Arbin CTI Instructions

Contents

Introduction

CTI is an abbreviation of Console TCP/IP Interface. It provides an interface for the third-party App to communicate with the Arbin battery testing system. It defines the protocol of information exchanges between Arbin MitsPro software and third-party applications. MitsPro software will serve as a server (right now, the server is embedded in Console.exe), and a third-party App will be a client.

Authorized Clients are allowed to send requests to MitsPro. For each legitimated request command, MitsPro will respond to the request and send the feedback command to the client to let the client know how MitsPro will process the request. Both the request and feedback commands are using TCP/IP communication protocol.

The commands that the Client send to MitsPro can be classified into two categories:

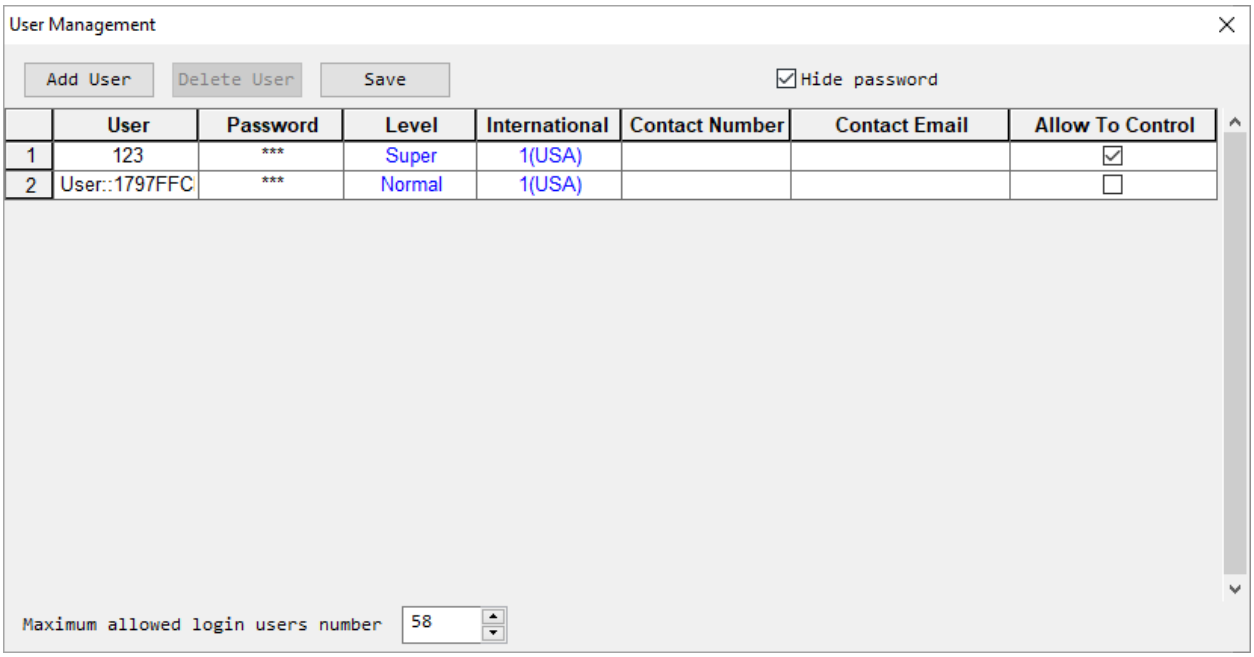
The first category is those need MitsPro to take action to control the cyclers, for example, to assign a schedule to channels, start, jump, stop, or resume channels. These commands will trigger MitsPro to send commands to microcontrollers in cyclers and eventually will change the running status of the related cyclers.

The second category is these require MitsPro to provide the status and other information about cyclers. These commands will not change the running status of cyclers.

GUI of CTI Server Side

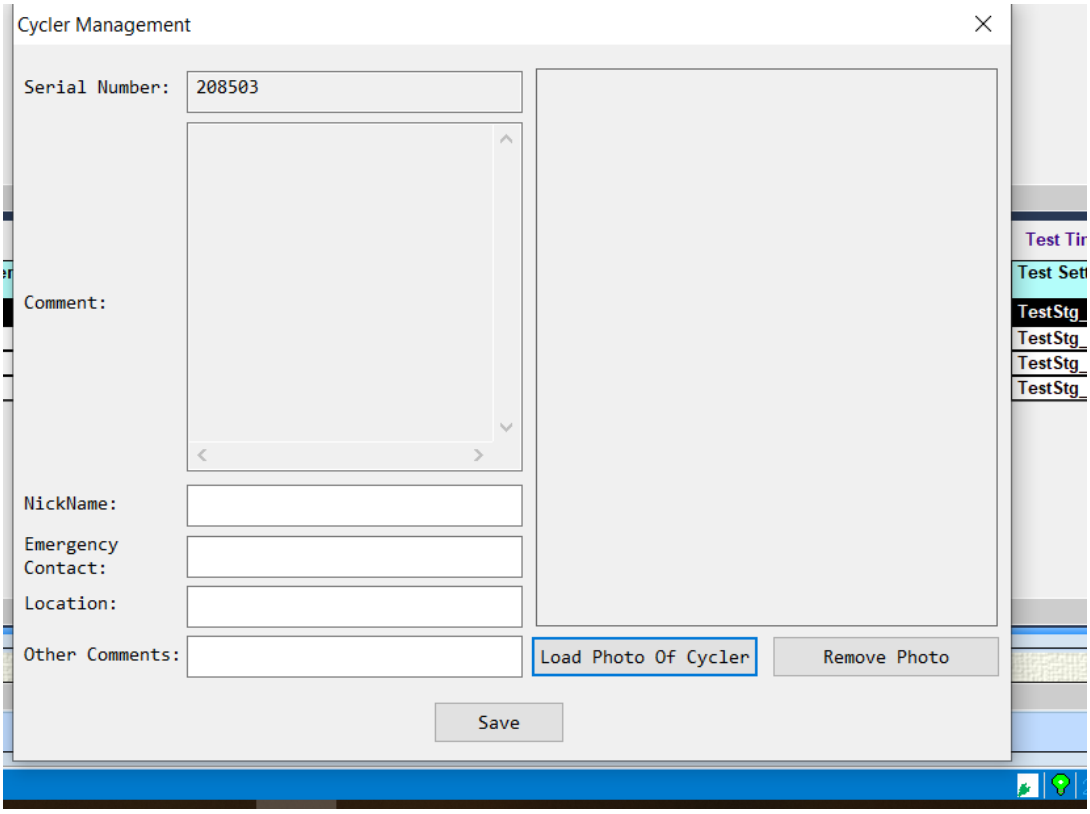
User management

It will support adding user's information such as user name, password, user-level, contact information used to notify by email and text message



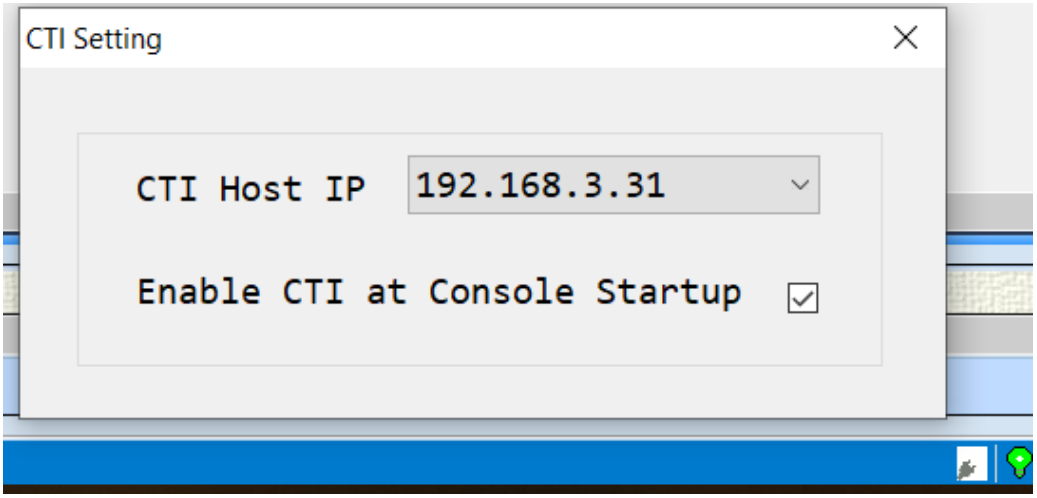
Cycler management

This is used for proving cycler's information, such as system specs, system serial number, the nickname for the cycler, contact information for the person in charge of the cycler, in case emergency contact is needed.



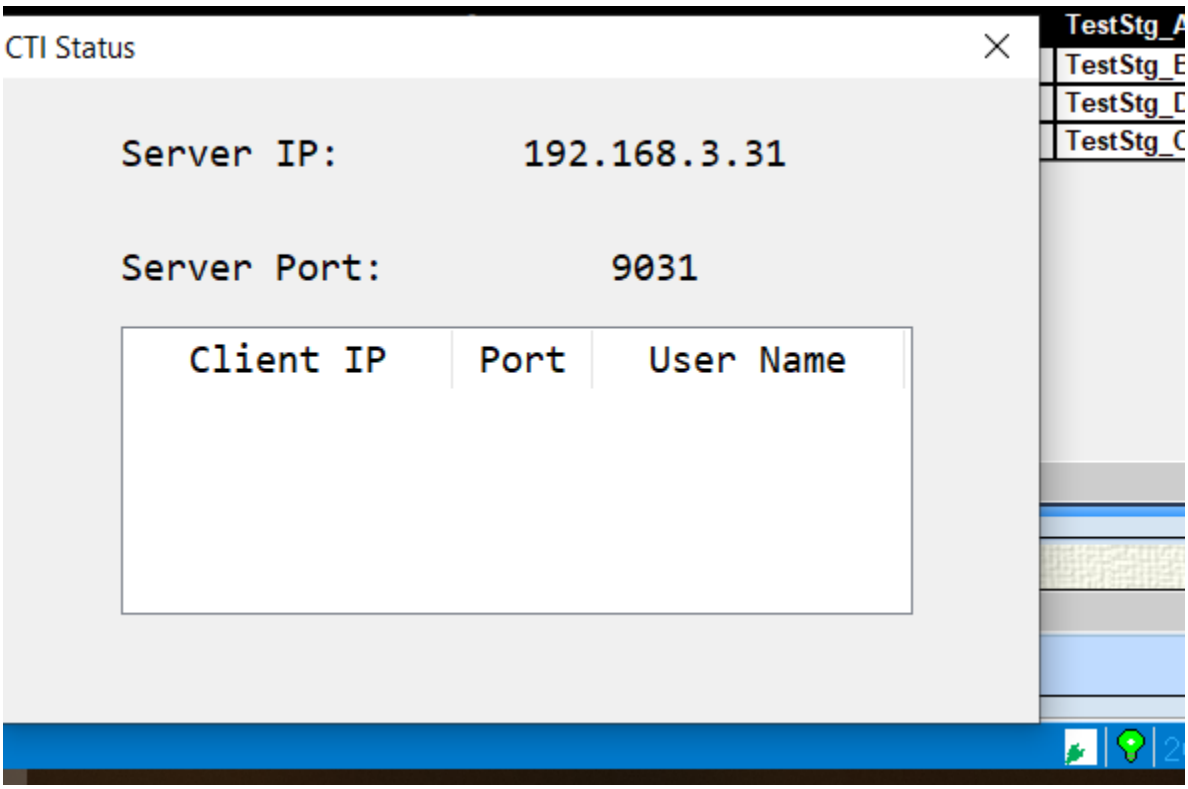
CTI server setting

This is used to set up the IP address of the CTI server so that the third-party can send out the connection request or other commands.



CTI server Status

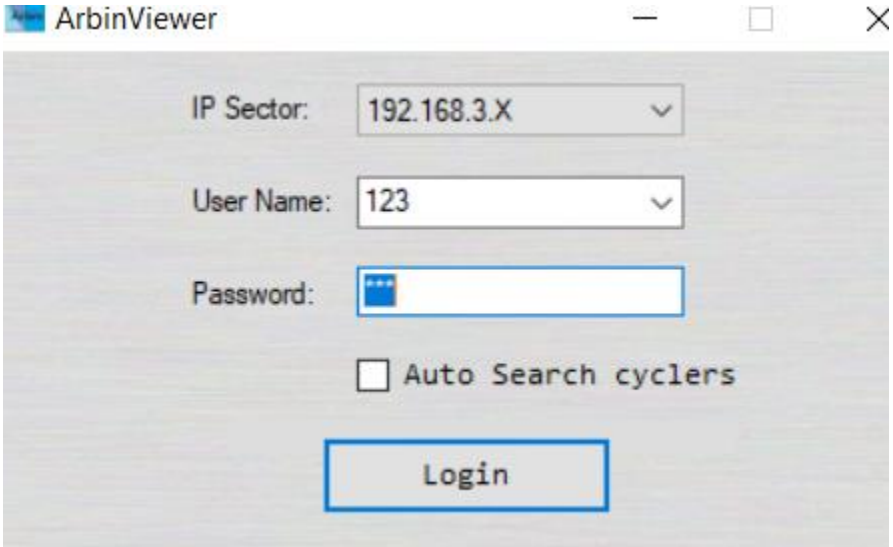
Once the CTI server is connected, the CTI status will show the server's IP address, port number, and current client's IP address and port number.



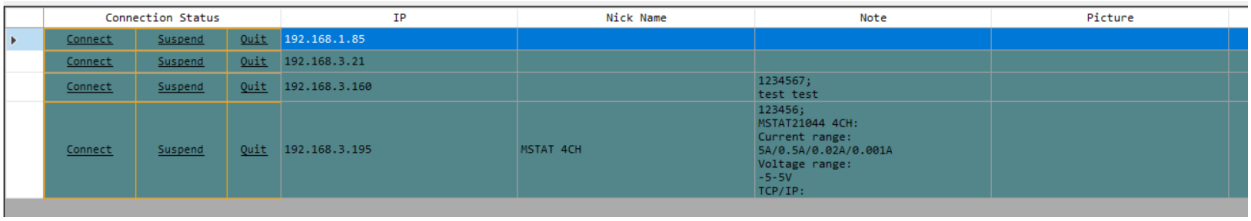
GUI of an Example Third-party App

To demonstrate the application of CTI, ArbinViewer is developed based on **ArbinCTI.dll** to monitor or control Arbin cyclers within the local network.

Login and password

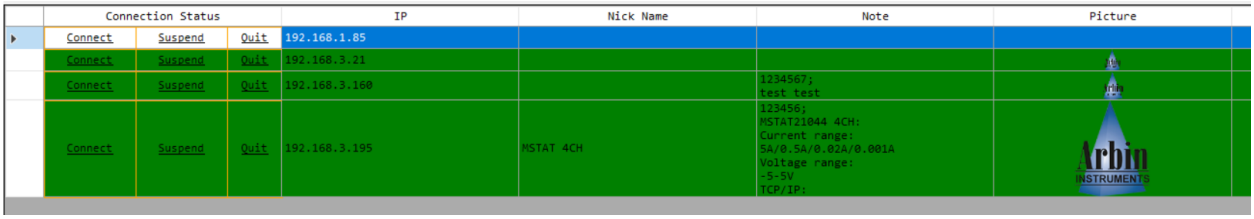


Cyclers list with the given user ID and password on the local network



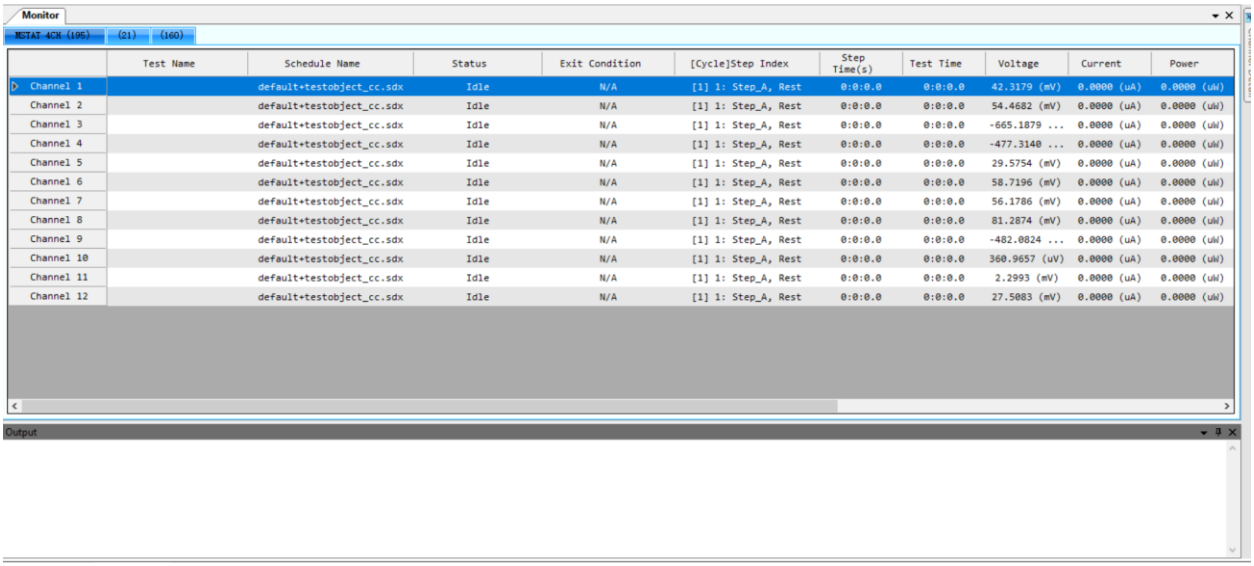
Connection Status			IP	Nick Name	Note	Picture
Connect	Suspend	Quit	192.168.1.85			
Connect	Suspend	Quit	192.168.3.21			
Connect	Suspend	Quit	192.168.3.160		1234567; test test	
Connect	Suspend	Quit	192.168.3.195	MSTAT 4CH	123456; MSTAT21044 4CH: Current range: 5A/0.5A/0.02A/0.001A Voltage range: -5-5V TCP/IP:	

After connected to the desired cyclers



Connection Status			IP	Nick Name	Note	Picture
Connect	Suspend	Quit	192.168.1.85			
Connect	Suspend	Quit	192.168.3.21			
Connect	Suspend	Quit	192.168.3.160		1234567; test test	
Connect	Suspend	Quit	192.168.3.195	MSTAT 4CH	123456; MSTAT21044 4CH: Current range: 5A/0.5A/0.02A/0.001A Voltage range: -5-5V TCP/IP:	

Show the channel status of cyclers by tab or by block, Detail view or Brief View

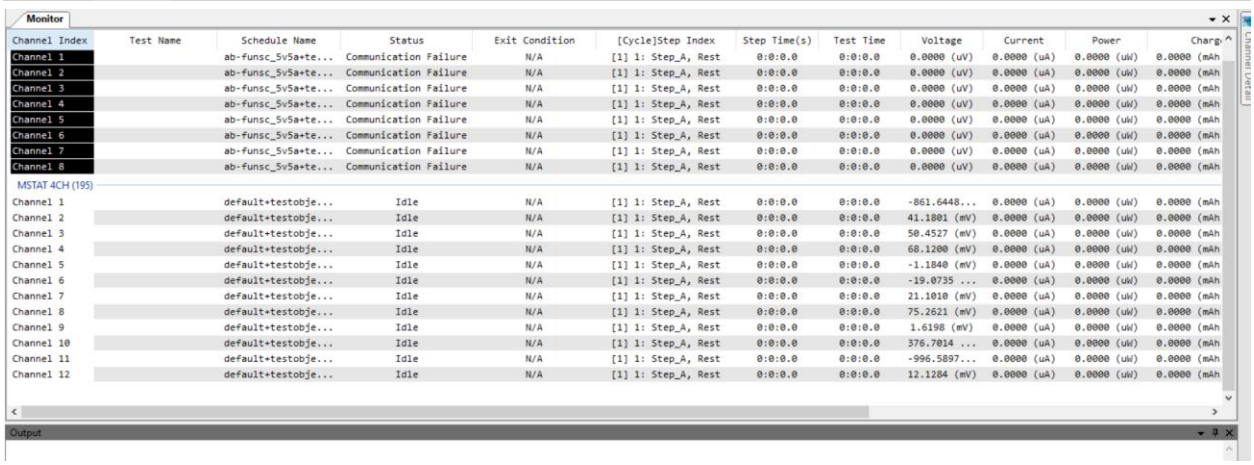


Monitor

MSTAT 4CH (195) (21) (160)

	Test Name	Schedule Name	Status	Exit Condition	[Cycle]Step Index	Step Time(s)	Test Time	Voltage	Current	Power
Channel 1		default+testobject_cc.sdx	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	42.3179 (mV)	0.0000 (uA)	0.0000 (uW)
Channel 2		default+testobject_cc.sdx	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	54.4682 (mV)	0.0000 (uA)	0.0000 (uW)
Channel 3		default+testobject_cc.sdx	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	-665.1879 ...	0.0000 (uA)	0.0000 (uW)
Channel 4		default+testobject_cc.sdx	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	-477.3140 ...	0.0000 (uA)	0.0000 (uW)
Channel 5		default+testobject_cc.sdx	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	29.5754 (mV)	0.0000 (uA)	0.0000 (uW)
Channel 6		default+testobject_cc.sdx	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	58.7196 (mV)	0.0000 (uA)	0.0000 (uW)
Channel 7		default+testobject_cc.sdx	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	56.1786 (mV)	0.0000 (uA)	0.0000 (uW)
Channel 8		default+testobject_cc.sdx	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	81.2874 (mV)	0.0000 (uA)	0.0000 (uW)
Channel 9		default+testobject_cc.sdx	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	-482.0824 ...	0.0000 (uA)	0.0000 (uW)
Channel 10		default+testobject_cc.sdx	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	360.9657 (uV)	0.0000 (uA)	0.0000 (uW)
Channel 11		default+testobject_cc.sdx	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	2.2993 (mV)	0.0000 (uA)	0.0000 (uW)
Channel 12		default+testobject_cc.sdx	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	27.5083 (mV)	0.0000 (uA)	0.0000 (uW)

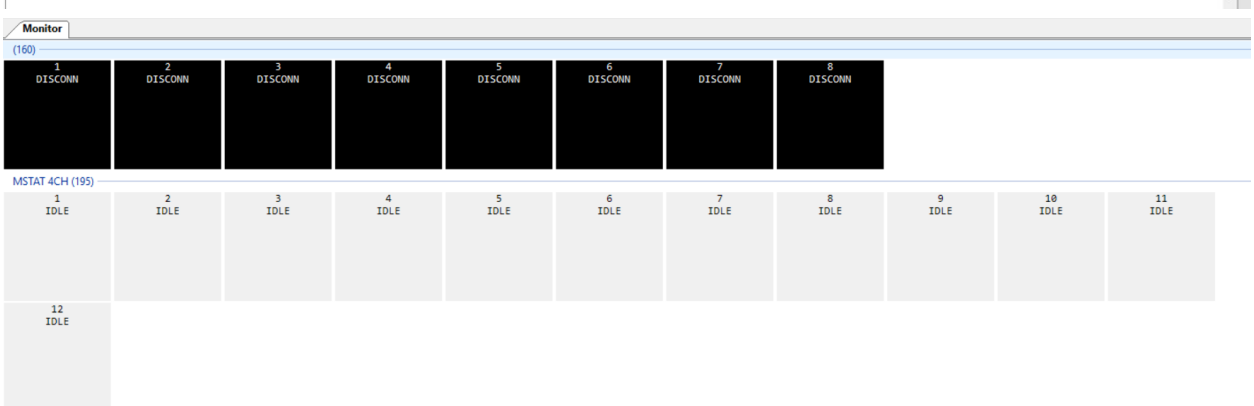
Output



Monitor

Channel Index	Test Name	Schedule Name	Status	Exit Condition	[Cycle]Step Index	Step Time(s)	Test Time	Voltage	Current	Power	Charg
Channel 1		ab-funsc_5v5a+te...	Communication Failure	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	0.0000 (uV)	0.0000 (uA)	0.0000 (uW)	0.0000 (mAh
Channel 2		ab-funsc_5v5a+te...	Communication Failure	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	0.0000 (uV)	0.0000 (uA)	0.0000 (uW)	0.0000 (mAh
Channel 3		ab-funsc_5v5a+te...	Communication Failure	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	0.0000 (uV)	0.0000 (uA)	0.0000 (uW)	0.0000 (mAh
Channel 4		ab-funsc_5v5a+te...	Communication Failure	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	0.0000 (uV)	0.0000 (uA)	0.0000 (uW)	0.0000 (mAh
Channel 5		ab-funsc_5v5a+te...	Communication Failure	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	0.0000 (uV)	0.0000 (uA)	0.0000 (uW)	0.0000 (mAh
Channel 6		ab-funsc_5v5a+te...	Communication Failure	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	0.0000 (uV)	0.0000 (uA)	0.0000 (uW)	0.0000 (mAh
Channel 7		ab-funsc_5v5a+te...	Communication Failure	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	0.0000 (uV)	0.0000 (uA)	0.0000 (uW)	0.0000 (mAh
Channel 8		ab-funsc_5v5a+te...	Communication Failure	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	0.0000 (uV)	0.0000 (uA)	0.0000 (uW)	0.0000 (mAh
MSTAT 4CH (195)											
Channel 1		default+testobje...	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	-861.6448...	0.0000 (uA)	0.0000 (uW)	0.0000 (mAh
Channel 2		default+testobje...	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	41.1801 (mV)	0.0000 (uA)	0.0000 (uW)	0.0000 (mAh
Channel 3		default+testobje...	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	50.4527 (mV)	0.0000 (uA)	0.0000 (uW)	0.0000 (mAh
Channel 4		default+testobje...	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	68.1200 (mV)	0.0000 (uA)	0.0000 (uW)	0.0000 (mAh
Channel 5		default+testobje...	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	-1.1840 (mV)	0.0000 (uA)	0.0000 (uW)	0.0000 (mAh
Channel 6		default+testobje...	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	-19.0735 ...	0.0000 (uA)	0.0000 (uW)	0.0000 (mAh
Channel 7		default+testobje...	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	21.1010 (mV)	0.0000 (uA)	0.0000 (uW)	0.0000 (mAh
Channel 8		default+testobje...	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	75.2621 (mV)	0.0000 (uA)	0.0000 (uW)	0.0000 (mAh
Channel 9		default+testobje...	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	1.6198 (mV)	0.0000 (uA)	0.0000 (uW)	0.0000 (mAh
Channel 10		default+testobje...	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	376.7014 ...	0.0000 (uA)	0.0000 (uW)	0.0000 (mAh
Channel 11		default+testobje...	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	-996.5897...	0.0000 (uA)	0.0000 (uW)	0.0000 (mAh
Channel 12		default+testobje...	Idle	N/A	[1] 1: Step_A, Rest	0:0:0.0	0:0:0.0	12.1284 (mV)	0.0000 (uA)	0.0000 (uW)	0.0000 (mAh

Output



Monitor

(160)

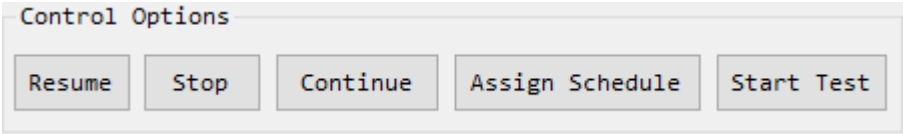
1	2	3	4	5	6	7	8
DISCONN	DISCONN	DISCONN	DISCONN	DISCONN	DISCONN	DISCONN	DISCONN

MSTAT 4CH (195)

1	2	3	4	5	6	7	8	9	10	11
IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE

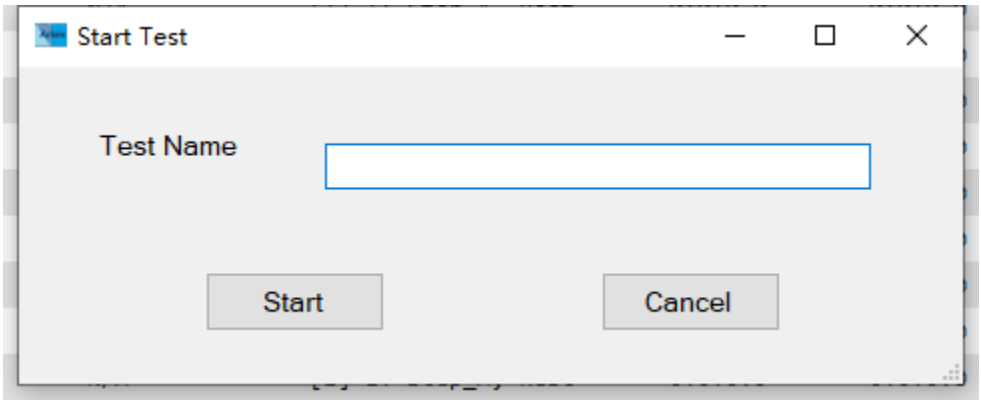
12
IDLE

Control Features



Basic control features include Stop, Continue and Resume channels. Advanced features include Assign Schedule and Start Test.

Start Test



Command Set

Control Commands Code

	Command	Command in bit code	Remark
	ASSIGN_SDU	0XBB210001	
	ASSIGN_SDU_FEED	0XBB120001	
	AUTOCALI_START_CODE	0XCD140001	
	AUTOCALI_START_FEED_BACK_CODE	0XCD410001	
	BROWSE_DIRECTORY	0XCC130001	
	BROWSE_DIRECTORY_FEED	0XCC310001	
	CONNECT_CODE	0XEEAB0002	
	CONNECT_FEED_CODE	0XEEBA0002	
	DELETE_FOLDER_CODE	0XCC130006	
	DELETE_FOLDER_FEED_CODE	0XCC310006	
	DOWNLOAD_CODE	0XCC130002	
	DOWNLOAD_FEED_CODE	0XCC310002	
	GET_CHANNELS_INFO	0XEEAB0003	
	GET_CHANNELS_INFO_FEED_CODE	0XEEBA0003	
	GET_RESUME_DATA_CODE	0XCD130003	
	GET_RESUME_DATA_FEED_CODE	0XCD310003	
	GET_SERIAL_CODE	0XBB340001	
	GET_SERIAL_FEED_CODE	0XBB430001	
	GET_START_DATA_CODE	0XCD130001	
	GET_START_DATA_FEED_CODE	0XCD310001	
	JUMP_CHANNEL_CODE	0XBB320005	
	JUMP_CHANNEL_FEED_CODE	0XBB230005	
	LOGIN_CODE	0XEEAB0001	
	LOGIN_FEED_CODE	0XEEBA0001	
	NEW_FOLDER_CODE	0XCC130005	
	NEW_FOLDER_FEED_CODE	0XCC310005	
	NEW_OR_DELETE_CODE	0XCC130004	Recommended use NEW_FOLDER_CODE or DELETE_FOLDER_CODE
	NEW_OR_DELETE_FEED_CODE	0XCC310004	Recommended use NEW_FOLDER_FEED_CODE or DELETE_FOLDER_FEED_CODE
	RESUME_EX_CODE	0XCD130004	
	RESUME_EX_FEED_BACK_CODE	0XCD310004	Not in used, use SCHEDULE_RESUME_FEED instead
	SCHEDULE_RESUME	0XBB310002	
	SCHEDULE_RESUME_FEED	0XBB130002	
	SCHEDULE_CONTINUE	0XBB320006	
	SCHEDULE_CONTINUE_FEED	0XBB230006	
	SCHEDULE_START	0XBB320004	
	SCHEDULE_START_FEED	0XBB230004	
	SEND_MSG_TO_CTI_CODE	0XCD140002	
	SEND_MSG_TO_CTI_FEED_BACK_CODE	0XCD410002	

	SET_MV_VAULE_CODE	0XBB150001	
	SET_MV_VAULE_FEED_CODE	0XBB510001	
	UPDATE_MV_ADVANCES_CODE	0XBB150002	
	UPDATE_MV_ADVANCES_FEED_CODE	0XBB510002	
	START_EX_CODE	0XCD130002	
	START_EX_FEED_CODE	0XCD310002	Not in used, use SCHEDULE_START_FEED instead
	STOP_CODE	0XBB310001	
	STOP_FEED_CODE	0XBB130001	
	UPLOAD_CODE	0XCC130003	
	UPLOAD_FEED_CODE	0XCC310003	
	Assign Name Rule	0xBB310007	Reserved, not implemented
	Assign Path	0xBB310008	
	Reset Local Database	0xBB310009	
	Restart DAQ	0xBB31000A	
	Restart Console	0xBB31000B	

```
public enum ArbinCommandCode :uint
{
    // Log in to the CTI server
    THIRD_PARTY_LOGIN_CODE                              =0XEEAB0001,
    // Login Result
    THIRD_PARTY_LOGIN_FEED_CODE                         =0XEEBA0001,

    // Set MetaVariable
    THIRD_PARTY_CMD_SET_MV_VAULE_CODE                   =0XBB150001,
    // Set MetaVariable result
    THIRD_PARTY_CMD_SET_MV_VAULE_FEED_CODE              =0XBB510001,

    // Set MetaVariable Advanced
    THIRD_PARTY_CMD_UPDATE_MV_ADVANCED_CODE             =0XBB150002,
    // Set MetaVariable Advanced Result
    THIRD_PARTY_CMD_UPDATE_MV_ADVANCED_FEED_CODE        =0XBB510002,

    // Stop Channel
    THIRD_PARTY_STOP_CODE                               =0XBB310001,
    // Stop Channel Result
    THIRD_PARTY_STOP_FEED_CODE                          =0XBB130001,

    // get cycelr serial number
    THIRD_PARTY_GET_SERIAL_CODE                         =0XBB340001,
    // get serial number result
    THIRD_PARTY_GET_SERIAL_FEED_CODE                    =0XBB430001,

    // jump steps
```

```
THIRD_PARTY_JUMP_CHANNEL_CODE                        =0XBB320005,
// get jump result
THIRD_PARTY_JUMP_CHANNEL_FEED_CODE                   =0XBB230005,

// Set kicked out
THIRD_PARTY_CMD_CONNECT_CODE                         =0XEEAB0002,
// kicked out result
THIRD_PARTY_CMD_CONNECT_FEED_CODE                    =0XEEBA0002,

// Get Channel info
THIRD_PARTY_CMD_GET_CHANNELS_INFO                    =0XEEAB0003,
// Get Channel info result
THIRD_PARTY_CMD_GET_CHANNELS_INFO_FEED_CODE          =0XEEBA0003,

// Start Channel
THIRD_PARTY_CMD_SCHEDULE_START                       =0XBB320004,
// Start Channel result
THIRD_PARTY_CMD_SCHEDULE_START_FEED                  =0XBB230004,

// Start Channel
THIRD_PARTY_CMD_SCHEDULE_CONTINUE                    =0XBB320006,
// Start Channel result
THIRD_PARTY_CMD_SCHEDULE_CONTINUE_FEED               =0XBB230006,

// Assign Schedule
THIRD_PARTY_CMD_ASSIGN_SDU                           =0XBB210001,
// Assign Schedule result
THIRD_PARTY_CMD_ASSIGN_SDU_FEED                      =0XBB120001,

// Resume Channel
THIRD_PARTY_CMD_SCHEDULE_RESUME                      =0XBB310002,
// Resumt Channel result
THIRD_PARTY_CMD_SCHEDULE_RESUME_FEED                 =0XBB130002,

// Browse folder
THIRD_PARTY_CMD_BROWSE_DIRECTORY                     =0XCC130001,
// Browse folder result
THIRD_PARTY_CMD_BROWSE_DIRECTORY_FEED                =0XCC310001,

// Download file
THIRD_PARTY_CMD_DOWNLOAD_CODE                        =0XCC130002,
// Download file result
THIRD_PARTY_CMD_DOWNLOAD_FEED_CODE                   =0XCC310002,

// Upload file
```

```
THIRD_PARTY_CMD_UPLOAD_CODE                          =0XCC130003,
// Upload file result
THIRD_PARTY_CMD_UPLOAD_FEED_CODE                     =0XCC310003,

//Create, delete files or folders
THIRD_PARTY_CMD_NEW_OR_DELETE_CODE                   =0XCC130004,
//Create, delete files or folders result
THIRD_PARTY_CMD_NEW_OR_DELETE_FEED_CODE              =0XCC310004,

// Create New Folder
THIRD_PARTY_CMD_NEW_FOLDER_CODE                      =0XCC130005,
// Create New Folder result
THIRD_PARTY_CMD_NEW_FOLDER_FEED_CODE                 =0XCC310005,

// Delete Folder
THIRD_PARTY_CMD_DELETE_FOLDER_CODE                   =0XCC130006,
// Delete Folder result
THIRD_PARTY_CMD_DELETE_FOLDER_FEED_CODE              =0XCC310006,

// Get the data before start channel
THIRD_PARTY_CMD_GET_START_DATA_CODE                  =0XCD130001,
// Get the data before start channel Result
THIRD_PARTY_CMD_GET_START_DATA_FEED_CODE             =0XCD310001,

// Advanced start channel
THIRD_PARTY_CMD_START_EX_CODE                        =0XCD130002,
// Advanced start channel result
THIRD_PARTY_CMD_START_EX_FEED_CODE                   =0XCD310002,

// Get data before resume channel
THIRD_PARTY_CMD_GET_RESUME_DATA_CODE                 =0XCD130003,
// Get data before resume channel result
THIRD_PARTY_CMD_GET_RESUME_DATA_FEED_CODE            =0XCD310003,

// Advanced resume channel
THIRD_PARTY_CMD_RESUME_EX_CODE                       =0XCD130004,
// Advanced resume channel Result
THIRD_PARTY_CMD_RESUME_EX_FEED_BACK_CODE             =0XCD310004,

// Start automatic calibration
THIRD_PARTY_CMD_AUTOCALI_START_CODE                  =0XCD140001,
// Start automatic calibration result
THIRD_PARTY_CMD_AUTOCALI_START_FEED_BACK_CODE        =0XCD410001,

// Send message to Mits Pro
```

```
    THIRD_PARTY_CMD_SEND_MSG_TO_CTI_CODE                =0XCD140002,
    // Send message to Mits Pro result
    THIRD_PARTY_CMD_SEND_MSG_TO_CTI_FEED_BACK_CODE      =0XCD410002
}
```

Detailed Command Data format

CTI_REQUEST_LOGIN:

User-defined datatype to hold username, password, other information when sending a login command to Server.

```
typedef struct
{
        BYTE      m_bPrefix[8];          // required to be filled with  eight 0xDD, mandatory.
        DWORD     m_dwLen;               // size in byte: m_dwCmd + m_dwCmd_Extend + m_user
                                         //     + m_password + m_btReserved1 + m_wCheckSum
        DWORD     m_dwCmd;               // required to be filled with 0XEEAB0001, mandatory, stating this is a login command
        DWORD     m_dwCmd_Extend;        // required to be filled with 0x000000, mandatory.

        BYTE      m_user[32]             // username, Insufficient length needs to fill 0x00
        BYTE      m_password[32]         // password, Insufficient length needs to fill 0x00

        WORD      m_wCheckSum;           //checksum: add all fields above byte by byte
} CTI_REQUEST_LOGIN;                     //187 bytes total
```

CTI_REQUEST_LOGIN_FEEDBACK:

The user-defined data type holds the status after login and response sent by the server such as IP Address, Note, Serial Number, and various information about the cycler.

```
typedef struct
{
        BYTE      m_bPrefix[8];          // Required to be filled with  eight 0xDD, mandatory.
        DWORD     m_dwLen;               /* Size in byte: m_dwCmd + m_dwCmd_Extend + Result + IP_Address + SN + Note
                                         + NickName + ... + dwPicLength + Picture + m_wCheckSum */
        DWORD     m_dwCmd;               // Required to be filled with 0XEEBA0001, mandatory, stating this is a login command
        DWORD     m_dwCmd_Extend;        // Required to be filled with 0x000000, mandatory.

        UINT      Result;                /* Return results, enumeration. 1: indicates that there is permission; 2: indicates no
                                         Permissions, cannot log in; 3: previously logged in, cannot modify information*/
        BYTE      IP_Address[4];         // CTI Server IP
        BYTE      SN[16];                // The SN number of the machine, Insufficient length needs to fill 0x00
        BYTE      Note[256];             // Customer Note in ArbinSys.cfg, Insufficient length needs to fill 0x00
        wchar_t   NickName[1024];        // Cycler's nickname, Insufficient length needs to fill 0x00
        wchar_t   Location[1024];        // Cycler's location, Insufficient length needs to fill 0x00
        wchar_t   EmergencyContactNameAndPhoneNumber[1024];     // Cycle's emergency contact, Insufficient length needs to fill 0x00
        wchar_t   OtherComments[1024];   // Cycler's comments, Insufficient length needs to fill 0x00
        wchar_t   Email[64];             // Cycler's email, Insufficient length needs to fill 0x00
        wchar_t   Call[16];              // Cycler's call, Insufficient length needs to fill 0x00
        UINT      ITAC;                  // Cycler's international telephone area code
        UINT      Version;               // Cycler's version
        UINT      IsAllowToControl;      // Whether to allow this ArbinViwer Client to control CTI, 0：Not allowed ,1:Allow
        UINT      dwChannelNum;          // The number of channels cycler has
        DWORD     dwUserType;            //0: Already set to normal user, 1: Already set to super user
        DWORD     dwPicLength;           // size of picture
        BYTE      Picture[];             // Image content (The actual length is determined by the number of dwPicLength)

        WORD      m_wCheckSum;           //checksum: add all fields above byte by byte
```

```
} CTI_REQUEST_LOGIN_FEEDBACK;
```

CTI_REQUEST_SCHEDULE_STOP :

The user-defined data type being sent to the cycler to stop the test. It contains information regarding channel number that needs to be stopped.

```
typedef struct
{
        BYTE        m_bPrefix[8];                   // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                        /* size in byte : m_dwCmd + m_dwCmd_Extend + m_dwIvChannelGlobalIndex
                                                     + m_btStopAll + m_btReserved1 + m_wCheckSum */
        DWORD       m_dwCmd;                        // required to be filled with 0xBB310001, mandatory, stating this is a stop command
        DWORD       m_dwCmd_Extend;                  // required to be filled with 0x000000, mandatory.

        DWORD       m_dwIvChannelGlobalIndex;       // the IV channel index need to be stopped, channel index start from 0
        BYTE        m_btStopAll;                    // filled with 0x00 to stop one channel; 0x01 to stop all channels
        BYTE        m_btReserved1[101];             // reserved, required to be filled with 0x00

        WORD        m_wCheckSum;                    //checksum: add all fields above byte by byte
} CTI_REQUEST_SCHEDULE_STOP;                        //128 bytes total
```

CTI_REQUEST_SCHEDULE_STOP_FEEDBACK :

The user-defined data type that reports the status of stop command operation.

```
typedef struct
{
        BYTE        m_bPrefix[8];                   // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                        /* size in byte : m_dwCmd + m_dwCmd_Extend + m_dwIvChannelGlobalIndex
                                                     + m_btResult + m_btReserved1 + m_wCheckSum */
        DWORD       m_dwCmd;                        // required to be filled with 0xBB310001, mandatory, stating this is a stop command
        DWORD       m_dwCmd_Extend;                 // required to be filled with 0x000000, mandatory.

        DWORD       m_dwIvChannelGlobalIndex;       // IV channel index, channel index start from 0
        BYTE        m_btResult;                     // Return result. , See ArbinCommandStopChannelFeed for more information
        BYTE        m_btReserved1[101];             // reserved, required to be filled with 0x00

        WORD        m_wCheckSum;                    //checksum: add all fields above byte by byte
} CTI_REQUEST_SCHEDULE_STOP_FEEDBACK;
```

CTI_REQUEST_SCHEDULE_START :

The user-defined data type that sends the command to cycler for starting the test. It contains information regarding channel number and test name.

```
typedef struct
{
        BYTE        m_bPrefix[8];                   // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                        /* size in byte: m_dwCmd + m_dwCmd_Extend + TestName
                                                    + ChannelNum + Channels + m_wCheckSum*/
        DWORD       m_dwCmd;                        // required to be filled with 0XBB320004, mandatory, stating this is a start command
        DWORD       m_dwCmd_Extend;                 // required to be filled with 0x000000, mandatory.

        Wchar_t     TestName[72];                   // The name assigned to the test, Insufficient length needs to fill 0x00
        DWORD       TotalChnToStart;                //  How  many channel to start
        Unsigned short ChnIndexList [];             /* Array, the IV channel number to run, the index starts from zero (The actual length is
```

```
                                        determined by TotalChnToStart  )*/

        WORD      m_wCheckSum;          // checksum: add all fields above byte by byte
} CTI_REQUEST_SCHEDULE_START;
```

CTI_REQUEST_SCHEDULE_START_FEEDBACK :

The user-defined data type that reports the status of the start test command.

```
typedef struct
{
        BYTE      m_bPrefix[8];         // required to be filled with  eight 0xDD, mandatory.
        DWORD     m_dwLen;              /* size in byte : m_dwCmd + m_dwCmd_Extend + m_dwIvChannelGlobalIndex
                                        + m_btStopAll + m_btReserved1 + m_wCheckSum */
        DWORD     m_dwCmd;              // required to be filled with 0XBB230004, mandatory, stating this is a start command
        DWORD     m_dwCmd_Extend;       // required to be filled with 0x000000, mandatory.

        DWORD     m_dwIvChannelGlobalIndex;  // the IV channel index need to be stopped, channel index start from 0
        BYTE      m_btResult;           // Return result. , See ArbinCommandStartChannelFeed for more information
        BYTE      m_btReserved1[101];   // reserved, required to be filled with 0x00

        WORD      m_wCheckSum;          //checksum: add all fields above byte by byte
} CTI_REQUEST_SCHEDULE_START_FEEDBACK;
```

CTI_REQUEST_SCHEDULE_CONTINUE :

The user-defined data type that sends the command to the cycler to continue testing. It contains information regarding channel number.

```
typedef struct
{
        BYTE      m_bPrefix[8];         // required to be filled with  eight 0xDD, mandatory.
        DWORD     m_dwLen;              /* size in byte: m_dwCmd + m_dwCmd_Extend
                                        + ChannelNum + Channels + m_wCheckSum*/
        DWORD     m_dwCmd;              /* required to be filled with 0XBB320006, mandatory, stating this is a continue
                                        command*/
        DWORD     m_dwCmd_Extend;       // required to be filled with 0x000000, mandatory.

        DWORD     TotalChnToContinue;         //  How  many channel to Continue
        Unsigned short ChnIndexList [];  /* Array, the IV channel number to run, the index starts from zero (The actual length is
                                        determined by TotalChnToContinue  )*/

        WORD      m_wCheckSum;          // checksum: add all fields above byte by byte
} CTI_REQUEST_SCHEDULE_CONTINUE;
```

CTI_REQUEST_SCHEDULE_CONTINUE_FEEDBACK :

The user-defined data type that reports the status of the continue test command.

```
typedef struct
{
        BYTE      m_bPrefix[8];         // required to be filled with  eight 0xDD, mandatory.
        DWORD     m_dwLen;              /* size in byte : m_dwCmd + m_dwCmd_Extend + m_dwIvChannelGlobalIndex
                                        + m_btStopAll + m_btReserved1 + m_wCheckSum */
        DWORD     m_dwCmd;              /* required to be filled with 0XBB230006, mandatory, stating this is a continue
                                        command*/
        DWORD     m_dwCmd_Extend;       // required to be filled with 0x000000, mandatory.

        DWORD     m_dwIvChannelGlobalIndex;  // the IV channel index need to be stopped, channel index start from 0
        BYTE      m_btResult;           // Return result. , See ArbinCommandContinueChannelFeed for more information
```

```
        BYTE        m_btReserved1[101];         // reserved, required to be filled with 0x00

        WORD        m_wCheckSum;                //checksum: add all fields above byte by byte
} CTI_REQUEST_SCHEDULE_CONTINUE_FEEDBACK;
```

CTI_REQUEST_SCHEDULE_RESUME :

The user-defined data type that holds the schedule resume commands and the channel number which needs to resume the test.

```
typedef struct
{
        BYTE        m_bPrefix[8];               // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                    /* size in byte : m_dwCmd + m_dwCmd_Extend + m_dwIvChannelGlobalIndex
                                                + m_btResumeAll + m_btReserved1 + m_wCheckSum */
        DWORD       m_dwCmd;                    // required to be filled with 0XBB310002, mandatory, stating this is a resume command
        DWORD       m_dwCmd_Extend;             // required to be filled with 0x000000, mandatory.

        DWORD       m_dwIvChannelGlobalIndex;   // the IV channel index need to be resume, channel index start from 0
        BYTE        m_btResumeAll;              // filled with 0x00 to resume one channel; 0x01 to resume all channels
        BYTE        m_btReserved1[101];         // reserved, required to be filled with 0x00

        WORD        m_wCheckSum;                //checksum: add all fields above byte by byte
} CTI_REQUEST_SCHEDULE_RESUME;
```

CTI_REQUEST_SCHEDULE_RESUME_FEED_BACK :

The user-defined data type that reports the status of the resume test command.

```
typedef struct
{
        BYTE        m_bPrefix[8];               // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                    /* size in byte : m_dwCmd + m_dwCmd_Extend + m_dwIvChannelGlobalIndex
                                                + m_btResult + m_btReserved1 + m_wCheckSum */
        DWORD       m_dwCmd;                    // required to be filled with 0XBB130002, mandatory, stating this is a resume command
        DWORD       m_dwCmd_Extend;             // required to be filled with 0x000000, mandatory.

        DWORD       m_dwIvChannelGlobalIndex;   // the IV channel index need to be resume, channel index start from 0
        BYTE        m_btResult;                 // Return result., See ArbinCommandResumChanneleFeed for more information
        BYTE        m_btReserved1[101];         // reserved, required to be filled with 0x00

        WORD        m_wCheckSum                 //checksum: add all fields above byte by byte
} CTI_REQUEST_SCHEDULE_RESUME_FEED_BACK;
```

CTI_REQUEST_CONNECT :

User-defined data type to hold the connect command and set/unset the kick-out flag.

```
typedef struct
{
        BYTE        m_bPrefix[8];               // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                    /* size in byte : m_dwCmd + m_dwCmd_Extend + dwSetKickOut
                                                + m_btReserved1 + m_wCheckSum */
        DWORD       m_dwCmd;                    // required to be filled with 0XEEAB0002, mandatory, stating this is a connect command
        DWORD       m_dwCmd_Extend;             // required to be filled with 0x000000, mandatory.

        DWORD       dwSetKickOut;               // 0: Can I be kicked, 1: Set to suggest not to kick

        BYTE        m_btReserved1[32];          // reserved, required to be filled with 0x00
```

```
        WORD      m_wCheckSum;                //checksum: add all fields above byte by byte
} CTI_REQUEST_CONNECT;
```

CTI_REQUEST_CONNECT_FEED_BACK :

The user-defined data type that reports the status of the connecting command.

```
typedef struct
{
        BYTE      m_bPrefix[8];               // required to be filled with  eight 0xDD, mandatory.
        DWORD     m_dwLen;                    /* size in byte : m_dwCmd + m_dwCmd_Extend + dwSetKickOut
                                              + dwConnectResult + m_btReserved1 + m_wCheckSum */
        DWORD     m_dwCmd;                    // required to be filled with 0XEEBA0002, mandatory, stating this is a connect command
        DWORD     m_dwCmd_Extend;             // required to be filled with 0x000000, mandatory.

        DWORD     dwSetKickOut;               // 0: Recommended for not kicking, 1: Recommended for kicking
        DWORD     dwConnectResult;            // Return result., 0: Accept; 1: Not Accept
        BYTE      m_btReserved1[32];          // reserved, required to be filled with 0x00

        WORD      m_wCheckSum                 //checksum: add all fields above byte by byte
} CTI_REQUEST_CONNECT_FEED_BACK;
```

CTI_REQUEST_GET_CHANNELS_INFO :

The user-defined data type that holds the command for getting channel info from the cycler.

```
typedef struct
{
        BYTE      m_bPrefix[8];               // required to be filled with  eight 0xDD, mandatory.
        DWORD     m_dwLen;                    /* size in byte : m_dwCmd + m_dwCmd_Extend + OnlyChannel
                                              + InfoType + NeedTypeSet + m_btReserved1 + m_wCheckSum */
        DWORD     m_dwCmd;                    /* required to be filled with 0XEEAB0003, mandatory, stating this is
                                              a get channel information command */
        DWORD     m_dwCmd_Extend;             // required to be filled with 0x000000, mandatory.

        short     SingleChnOnly;              // -1, non-single channel,> = 0 is the specified channel number
        short     ChnSelection;                       /* Enumeration, 1: indicates all channels, 2: indicates the running channel,
                                              3: indicates the Unsafe channel; */
        DWORD     AuxDataOptions;             /* Enumeration 0x00: means no auxiliary data is required, 0x100: Requires BMS, 0x200:
                                              Requires SMB, 0x400: Requires AUX */
        BYTE      m_btReserved1[32];          // reserved, required to be filled with 0x00
        WORD      m_wCheckSum;                //checksum: add all fields above byte by byte
} CTI_REQUEST_GET_CHANNELS_INFO;
```

CTI_REQUEST_GET_CHANNELS_INFO_FEED_BACK :

The user-defined data type holds the status from the get_channel_info command and all the channel information in the form of an array.

```
typedef struct
{
        BYTE      m_bPrefix[8];               // required to be filled with  eight 0xDD, mandatory.
        DWORD     m_dwLen;                    /* size in byte : m_dwCmd + m_dwCmd_Extend + ChannelNum
                                              + Channels + m_wCheckSum */
        DWORD     m_dwCmd;                    /* required to be filled with 0XEEBA0003, mandatory, stating this is
                                              a get channel information command */
        DWORD     m_dwCmd_Extend;             // required to be filled with 0x000000, mandatory.

        DWORD     ChannelNum;                 // Number of channels
        THIRD_PARTY_CHANNEL Channels[];       // The actual length of the array is determined by the number of channels
```

```
                                                //See ArbinCommandGetChannelDataFeed for more
        WORD      m_wCheckSum                   //checksum: add all fields above byte by byte
} CTI_REQUEST_GET_CHANNELS_INFO_FEED_BACK;
typedef struct
{
        UINT        ChannelIndex;               // the IV channel index need to be stopped, channel index start from 0
        THIRD_PARTY_STATUS_INFOMATION Info;     // Structure: contains channel data
} THIRD_PARTY_CHANNEL;
typedef struct
{
        short Status;                           // Present status of a channel.
        BYTE m_bCommFailure;                    /* network status.  false: No problem with the connection.
                                                true</b>: Can't Connect to the mcu. */
        wchar_t Schedule[200];                  //The file name of the schedule under running
        wchar_t Testname[72];                   // test name
        BYTE ExitCondition[100];                // The stop or exit condition of test
        BYTE StepAndCycleformat[64];            // Currently [active test cycle number] running step number in the active schedule
        wchar_t Barcode[72];                    // The Barcode of Battery
        wchar_t CANCfgName[200];                // BMS Config Name
        wchar_t SMBCfgName[200];                // SMB Config Name
        unsigned short MasterChannel;           // Parallel main channels, if no parallel occurs, the value is equal to ChannelIndex
        double TestTime;                        // Elapsed time counted from the starting point of present active test
        double StepTime;                        // Elapsed time counted from the starting point of present active step
        float  Voltage;                         // Measured value of present channel voltage
        float  Current;                         // Measured value of present channel current
        float  Power;                           // Calculated value of present channel power
        float  ChargeCapacity;                  // Cumulative value of present channel charge capacity
        float  DischargeCapacity;               // Cumulative value of present channel discharge capacity
        float  ChargeEnergy;                    // Cumulative value of present channel charge energy
        float  DishargeEnergy;                  // Cumulative value of present channel discharge energy
        float  InternalResistance;              // Calculated internal resistance
        float  dvdt;                            // The first-order change rate of voltage, Change in voltage over an interval of time,  dt
        float  ACR;                             // Measured value of battery internal resistance
        float  ACI;                             // Calculated value of impedance resulting from 1kHz imposed sine wave
        float  ACIPhase;                        // Phase angle value of the AC impedance in degree
        unsigned short nAuxVoltageCount;        // Number of Aux voltages
        unsigned short nAuxTemperatureCount;    // Number of Aux Temperature
        unsigned short nAuxPressureCount;       // Number of Aux voltages
        unsigned short nAuxExternalCount;       // Number of Aux External
        unsigned short nAuxFlowCount;           // Number of Aux Flow
        unsigned short nAuxAoCount;             // Number of Aux Ao
        unsigned short nAuxDiCount;             // Number of Aux Di
        unsigned short nAuxDoCount;             // Number of Aux Do
        unsigned short nAuxHumidityCount;       // Number of Aux Humidity
        unsigned short nAuxSafetyCount;         // Number of Aux Safety
        unsigned short nAuxPhCount;             // Number of Aux Ph
        unsigned short nAuxDensityCount;        // Number of Aux Density
        unsigned short BMSNum;                  // Number of BMS
        unsigned short SMBNum;                  // Number of SMB
        THIRD_PARTY_AUX_VALUE AuxValues[];      /* The actual length is determined by the number of nAuxVoltageCount +
                                                nAuxTemperatureCount + … + nAuxDensityCount*/
        THIRD_PARTY_BMS_VALUE BMSValues[];      // The actual length is determined by the number of ChannelNum
        THIRD_PARTY_SMB_VALUE SMBValues[];      // The actual length is determined by the number of ChannelNum
} THIRD_PARTY_STATUS_INFOMATION;
typedef struct
{
        float Value;                            // Auxiliary data
```

```
        float dtValue;                      // Auxiliary data differentiation of time, if available
} THIRD_PARTY_AUX_VALUE;
typedef struct
{
        UINT Index;                         // Entry index on BMS Config
        double Value;                       // Data
        BYTE Unit[];                        // The unit of the entry on BMS Config ('\0'End, variable length)
} THIRD_PARTY_BMS_VALUE;
typedef struct
{
        UINT Index;                         // Entry index on SMB Config
        UINT Type;                          //0:DOUBLE, 1: string
        union
        {
          double Value;                     // Data
          BYTE StringValue[];               // ('\0' End, variable length)
        }
        BYTE Unit[];                        // The unit of the entry on SMB Config ('\0' End, variable length)
} THIRD_PARTY_SMB_VALUE;
```

CTI_REQUEST_ASSIGN_SCHEDULE :

The user-defined data type to hold the assigned schedule command and custom user-defined test values and schedule name.

```
typedef struct
{
        BYTE      m_bPrefix[8];             // required to be filled with  eight 0xDD, mandatory.
        DWORD     m_dwLen;                  /* size in byte : m_dwCmd + m_dwCmd_Extend + dwIvChannelGlobalIndex
                                            + AssignSduAll + … + fMV_UD16 + m_btReserved1 + m_wCheckSum */
        DWORD     m_dwCmd;                  /* required to be filled with 0XBB210001, mandatory, stating this is
                                            a assign schedule command */
        DWORD     m_dwCmd_Extend;           // required to be filled with 0x000000, mandatory.

        int       dwIvChannelGlobalIndex;   // the IV channel index need to be stopped, channel index start from 0
        BYTE      AssignSduAll;             // filled with 0x00 to assign one channel; 0x01 to assign all channels
        wchar_t   ScheduleName[200];        // The file name of the schedule
        float     flCapacity;               // Capacity
        wchar_t   ItemId[72];               // Barcode
        float     fMV_UD1;                  // Custom user-defined variable values
        float     fMV_UD2;                  // Custom user-defined variable values
        float     fMV_UD3;                  // Custom user-defined variable values
        float     fMV_UD4;                  // Custom user-defined variable values
        float     fMV_UD5;                  // Custom user-defined variable values
        float     fMV_UD6;                  // Custom user-defined variable values
        float     fMV_UD7;                  // Custom user-defined variable values
        float     fMV_UD8;                  // Custom user-defined variable values
        float     fMV_UD9;                  // Custom user-defined variable values
        float     fMV_UD10;                 // Custom user-defined variable values
        float     fMV_UD11;                 // Custom user-defined variable values
        float     fMV_UD12;                 // Custom user-defined variable values
        float     fMV_UD13;                 // Custom user-defined variable values
        float     fMV_UD14;                 // Custom user-defined variable values
        float     fMV_UD15;                 // Custom user-defined variable values
        float     fMV_UD16;                 // Custom user-defined variable values

        BYTE      m_btReserved1[32];        // reserved, required to be filled with 0x00

        WORD      m_wCheckSum;              //checksum: add all fields above byte by byte
```

```
} CTI_REQUEST_ASSIGN_SCHEDULE;
```

CTI_REQUEST_ASSIGN_SCHEDULE_FEED_BACK :

The user-defined data type is by the cycler with the status of the Assign Schedule command.

```
typedef struct
{
        BYTE      m_bPrefix[8];          // required to be filled with  eight 0xDD, mandatory.
        DWORD     m_dwLen;               /* size in byte : m_dwCmd + m_dwCmd_Extend + dwSetKickOut
                                         + dwConnectResult + m_btReserved1 + m_wCheckSum */
        DWORD     m_dwCmd;               /* required to be filled with 0XBB120001, mandatory, stating this is
                                         a assign schedule command */
        DWORD     m_dwCmd_Extend;        // required to be filled with 0x000000, mandatory.

        int       dwIvChannelGlobalIndex;  //the IV channel index need to be stopped, channel index start from 0
        BYTE      btResult;              // Return result., See ArbinCommandAssignScheduleFeed for more information

        BYTE      Reserved1[101];        // reserved, required to be filled with 0x00

        WORD      m_wCheckSum            //checksum: add all fields above byte by byte
} CTI_REQUEST_ASSIGN_SCHEDULE_FEED_BACK;
```

CTI_REQUEST_BROWSE_DIRECTORY :

User-defined data type to hold the directory path in the target PC along with the Browse Directory command.

```
typedef struct
{
        BYTE      m_bPrefix[8];          // required to be filled with  eight 0xDD, mandatory.
        DWORD     m_dwLen;               /* size in byte : m_dwCmd + m_dwCmd_Extend + DirectoryPath
                                          m_wCheckSum */
        DWORD     m_dwCmd;               /* required to be filled with 0XCC130001, mandatory, stating this is
                                         a browse directory command */
        DWORD     m_dwCmd_Extend;        // required to be filled with 0x000000, mandatory.

        wchar_t   DirectoryPath[1024];   // Request the path to browse the directory

        WORD      m_wCheckSum;           //checksum: add all fields above byte by byte
} CTI_REQUEST_BROWSE_DIRECTORY;
```

CTI_REQUEST_BROWSE_DIRECTORY_FEED_BACK :

The user-defined data type that is returned after the Browse_request is performed. It holds the Filename and Directory name along with the status of the Browse Directory command.

```
typedef struct
{
        BYTE      m_bPrefix[8];          // required to be filled with  eight 0xDD, mandatory.
        DWORD     m_dwLen;               /* size in byte : m_dwCmd + m_dwCmd_Extend + Result
                                         + nDirFileCount + m_ DirFileList + m_wCheckSum */
        DWORD     m_dwCmd;               /* required to be filled with 0XCC310001, mandatory, stating this is
                                         a browse directory command */
        DWORD     m_dwCmd_Extend;        // required to be filled with 0x000000, mandatory.

        UINT      Result;                // Return result., See ArbinCommandBrowseDirectoryFeed for more information
        UINT      nDirFileCount;         // Total number of files and directories
        DirFileInfo DirFileList[];       // The length of the array is determined by the actual number of nDirFileCount.

        WORD      m_wCheckSum            //checksum: add all fields above byte by byte
```

```
} CTI_REQUEST_BROWSE_DIRECTORY_FEED_BACK;
typedef struct
{
        UINT Type;                          //0: directory; 1: file
        wchar_t DirFileName[64];            // Directory or file name
        DWORD dwSize;
                                            // File size (no need for directory)
        wchar_t wcModified[32];             // Directory or file modification time
} DirFileInfo;
```

CTI_REQUEST_DOWNLOAD_FILE :

The user-defined data type to store the file path of the file to be downloaded along with the timestamp of the request and command to download the file from the target PC.

```
typedef struct
{
        BYTE    m_bPrefix[8];       // required to be filled with  eight 0xDD, mandatory.
        DWORD   m_dwLen;            /* size in byte : m_dwCmd + m_dwCmd_Extend + FilePath
                                     + DownloadTime + m_wCheckSum */
        DWORD   m_dwCmd;            /* required to be filled with 0XCC130002, mandatory, stating this is
                                    a download file command */
        DWORD   m_dwCmd_Extend;     // required to be filled with 0x000000, mandatory.

        wchar_t FilePath[1024];     // File path to download
        double  DownloadTime;       // Time stamp (unique)

        WORD    m_wCheckSum;        //checksum: add all fields above byte by byte
} CTI_REQUEST_DOWNLOAD_FILE;
```

CTI_REQUEST_DOWNLOAD_FILE_FEED_BACK :

The user-defined data type to hold the status of file download request along with data packet and packet information.

```
typedef struct
{
        BYTE    m_bPrefix[8];       // required to be filled with  eight 0xDD, mandatory.
        DWORD   m_dwLen;            /* size in byte : m_dwCmd + m_dwCmd_Extend + Result
                                    + DownloadTime + … + dwFileLength + File + m_wCheckSum */
        DWORD   m_dwCmd;            /* required to be filled with 0XCC310002, mandatory, stating this is
                                    a download file command */
        DWORD   m_dwCmd_Extend;     // required to be filled with 0x000000, mandatory.

        UINT    Result;             // Return result, See ArbinCommandDownLoadFileFeed for more information
        double  DownloadTime;       // Time stamp (unique)
        UINT    uGeneralPackage;    // Split the file into multiple packages and send it multiple times (total package)
        UINT    uPackageIndex;      // Package index for easy splicing of files
        BYTE    m_MD5[16];          // Use MD5 check file
        UINT64  dwFileLength;       // File size
        BYTE[0] File;               // File content, the size of the array is determined by the dwFileLength

        WORD    m_wCheckSum         //checksum: add all fields above byte by byte
} CTI_REQUEST_DOWNLOAD_FILE_FEED_BACK;
```

CTI_REQUEST _ UPLOAD_FILE :

The user-defined data type to hold the command to upload the file along with file path and packet information.

```
typedef struct
```

```
{
        BYTE        m_bPrefix[8];           // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                /* size in byte : m_dwCmd + m_dwCmd_Extend + FilePath
                                            + dwFileLength + … + File + m_wCheckSum */
        DWORD       m_dwCmd;                /* required to be filled with 0XCC130003, mandatory, stating this is
                                            a upload file file command */
        DWORD       m_dwCmd_Extend;         // required to be filled with 0x000000, mandatory.

        wchar_t FilePath[1024];             // The path to which the file will be uploaded
        UINT64 dwFileLength;                // File size
        UINT uGeneralPackage;               //Split the file into multiple packages and send it multiple times (total package)
        UINT uPackageIndex;                 //Package index for easy splicing of files
        double UploadTime;                  // Time stamp (unique)
        BYTE m_MD5[16];                     //Use MD5 check file
        BYTE[0] File;                       // File content, the size of the array is determined by the dwFileLength

        WORD        m_wCheckSum;            //checksum: add all fields above byte by byte
} CTI_REQUEST_UPLOAD_FILE;
```

CTI_REQUEST_UPLOAD_FILE_FEED_BACK :

The user-defined data type to hold the status of the uploaded file along with packet information.

```
typedef struct
{
        BYTE        m_bPrefix[8];           // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                /* size in byte : m_dwCmd + m_dwCmd_Extend + Result
                                            + UploadTime + uGeneralPackage + uPackageIndex + m_wCheckSum */
        DWORD       m_dwCmd;                /* required to be filled with 0XCC310003, mandatory, stating this is
                                            a upload file command */
        DWORD       m_dwCmd_Extend;         // required to be filled with 0x000000, mandatory.

        UINT        Result;                 // Return result, See ArbinCommandUpLoadFileFeed for more information
        double      UploadTime;             // Time stamp (unique)
        UINT        uGeneralPackage;        // Split the file into multiple packages and send it multiple times (total package)
        UINT        uPackageIndex;          // Package index for easy splicing of files

        WORD       m_wCheckSum              //checksum: add all fields above byte by byte
} CTI_REQUEST_UPLOAD_FILE_FEED_BACK;
```

CTI_REQUEST_NEW_OR_DELETE_FOLDER :

The user-defined data type to hold the file path and flag for NEW/DELETE a folder along with the command.

```
typedef struct
{
        BYTE        m_bPrefix[8];           // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                /* size in byte : m_dwCmd + m_dwCmd_Extend + Type
                                            + FilePath + m_wCheckSum */
        DWORD       m_dwCmd;                /* required to be filled with 0XCC130004, mandatory, stating this is
                                            a new or delete folder/file file command */
        DWORD       m_dwCmd_Extend;         // required to be filled with 0x000000, mandatory.

        UINT        Type;                   // 0: Delete 1: New(New directory only)
        wchar_t     FilePath[1024];         // Request new directory or delete directory, file path

        WORD        m_wCheckSum;            //checksum: add all fields above byte by byte
} CTI_REQUEST_NEW_OR_DELETE_FOLDER;
```

CTI_REQUEST_NEW_OR_DELETE_FOLDER_FEED_BACK :

The user-defined data type to hold the status of the create new/ Delete folder command.

```
typedef struct
{
        BYTE      m_bPrefix[8];          // required to be filled with  eight 0xDD, mandatory.
        DWORD     m_dwLen;               /* size in byte : m_dwCmd + m_dwCmd_Extend + Result
                                         + m_wCheckSum */
        DWORD     m_dwCmd;               /* required to be filled with 0XCC310004, mandatory, stating this is
                                         a new or delete folder/file command */
        DWORD     m_dwCmd_Extend;        // required to be filled with 0x000000, mandatory.

        UINT      Result;                // Return result, See ArbinCommandNewOrDeleteFeed for more information

        WORD      m_wCheckSum            //checksum: add all fields above byte by byte
} CTI_REQUEST_NEW_OR_DELETE_FOLDER_FEED_BACK;
```

CTI_REQUEST_GET_START_DATA :

The user-defined data type to hold the channel number and command to start the test.

```
typedef struct
{
        BYTE      m_bPrefix[8];          // required to be filled with  eight 0xDD, mandatory.
        DWORD     m_dwLen;               /* size in byte : m_dwCmd + m_dwCmd_Extend + ChannelNum
                                         + channelNo + m_wCheckSum */
        DWORD     m_dwCmd;               /* required to be filled with 0XCD130001, mandatory, stating this is
                                         a get start channel data command */
        DWORD     m_dwCmd_Extend;        // required to be filled with 0x000000, mandatory.

        UINT ChannelNum;                 // Number of channels
        ushort channelNo[];              // The array size is determined by ChannelNum, the channel number (starting at 0)

        WORD      m_wCheckSum;           //checksum: add all fields above byte by byte
} CTI_REQUEST_GET_START_DATA;
```

CTI_REQUEST_GET_START_DATA_FEED_BACK :

The user-defined data type to hold the status of the start test command.

```
typedef struct
{
        BYTE      m_bPrefix[8];          // required to be filled with  eight 0xDD, mandatory.
        DWORD     m_dwLen;               /* size in byte : m_dwCmd + m_dwCmd_Extend + ChannelNum
                                         + channelNo + m_wCheckSum */
        DWORD     m_dwCmd;               /* required to be filled with 0XCD310001, mandatory, stating this is
                                         a get start channel data command */
        DWORD     m_dwCmd_Extend;        // required to be filled with 0x000000, mandatory.

        UINT      ChannelNum;            // Number of channels
        THIRD_PARTY_CHANNEL_START_DATA_DESC channelNo[];  // The array size is determined by ChannelNum, get start channel data

        WORD      m_wCheckSum            //checksum: add all fields above byte by byte
} CTI_REQUEST_GET_START_DATA_FEED_BACK;
typedef struct
{
        UINT      Channel;               //  the channel number (starting at 0)
```

```
        UINT        channelCode;                    //  Code 34: Database query Test Name failed; Code 35: Test Name is empty
                                                    Code 36: Failed to get resume table data
        UINT        TestNameNum;                    // Number of TestNames
        UINT        StepNum;                        // Number of step
        wchar_t     Schedule[200];                  // The file name of the schedule
        float       fMV_UD1;                        // Custom user-defined variable values
        float       fMV_UD2;                        // Custom user-defined variable values
        float       fMV_UD3;                        // Custom user-defined variable values
        float       fMV_UD4;                        // Custom user-defined variable values
        float       fMV_UD5;                        // Custom user-defined variable values
        float       fMV_UD6;                        // Custom user-defined variable values
        float       fMV_UD7;                        // Custom user-defined variable values
        float       fMV_UD8;                        // Custom user-defined variable values
        float       fMV_UD9;                        // Custom user-defined variable values
        float       fMV_UD10;                       // Custom user-defined variable values
        float       fMV_UD11;                       // Custom user-defined variable values
        float       fMV_UD12;                       // Custom user-defined variable values
        float       fMV_UD13;                       // Custom user-defined variable values
        float       fMV_UD14;                       // Custom user-defined variable values
        float       fMV_UD15;                       // Custom user-defined variable values
        float       fMV_UD16;                       // Custom user-defined variable values
        THIRD_PARTY_STRING TestNames[];             // The array size is determined by TestNameNum, Test name
        THIRD_PARTY_STRING Steps[];                 // The array size is determined by StepNum, Step label
} THIRD_PARTY_CHANNEL_START_DATA_DESC;
typedef struct
{
        UINT Length;                                // String length
        char stringData[];                          // The array size is determined by Length, String content
} THIRD_PARTY_STRING;
```

CTI_REQUEST_SCHEDULE_START_EX :

The user-defined data type to hold the command to start the test with exception and the channel numbers where the test is to be started.

```
typedef struct
{
        BYTE        m_bPrefix[8];                   // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                        /* size in byte : m_dwCmd + m_dwCmd_Extend + ChannelNum
                                                    + channelNo + m_wCheckSum */
        DWORD       m_dwCmd;                        /* required to be filled with 0XCD130002, mandatory, stating this is
                                                    a start channel command */
        DWORD       m_dwCmd_Extend;                 // required to be filled with 0x000000, mandatory.

        UINT        ChannelNum;                     // Number of channels
        THIRD_PARTY_CHANNEL_START_DATA_EX channelNo[]          // The array size is determined by ChannelNum

        WORD        m_wCheckSum;                    //checksum: add all fields above byte by byte
} CTI_REQUEST_SCHEDULE_START_EX;
typedef struct
{
        UINT Channel;                               // the channel number (starting at 0)
        wchar_t TestName[72];                       // Test name
        wchar_t Schedule[200];                      // The file name of the schedule
        wchar_t Createor[64];                       // Creator of channel operation
        wchar_t Comment[64];                        // Run channel comment
        UINT nSelectStep;                           // Select the first few steps of Schedule to start running
        THIRD_PARTY_RESUME_DATA ResumeData;// Resume table data
```

```
}THIRD_PARTY_CHANNEL_START_DATA_EX;
typedef struct
{
        UINT Cycle;                     // Cycle number
        double TestTime;                // Running test total time
        double StepTime;                // Current step time
        double CCapacity;               // Charge capacity
        double DCapacity;               // Discharge capacity
        double CEnergy;                 // Charge energy
        double DEnergy;                 // Discharge energy

        double TC_Time1;                /* TC_Time1, TC_Time2, TC_Time3, and TC_Time4 are time counters.
        double TC_Time2;                Time counters can be used to count the total test time of a group of steps.
        double TC_Time3;                Further, the time counters can be used as the step termination limit a
        double TC_Time4;                or logging datlimit. */

        double TC_CCapacity1;           /* TC_Charge_Capacity1 and TC_Charge_Capacity2 are charge capacity counters.
        double TC_CCapacity2;           The charge capacity is the capacity when the current is positive */

        double TC_DCapacity1;           /* TC_Disharge_Capacity1 and TC_Disharge_Capacity2 are discharge capacity counters.
        double TC_DCapacity2;           The discharge capacity is the capacity when the current is negative. */

        double TC_CEnergy1;             /* TC_Charge_Energy1 and TC_Charge_Energy2 are charge energy counters.
        double TC_CEnergy2;             The charge energy is the positive energy value when calculated by the formula
                                        ∫I*V*dt */

        double TC_DEnergy1;             /* TC_Discharge_Energy1 and TC_Discharge_Energy2 are discharge energy counters.
        double TC_DEnergy2;             The discharge energy is the negative energy value when calculated by the formula
                                        ∫I*V*dt  */

        float  MV_Counter1;             // Universal Counter 1
        float  MV_Counter2;             // Universal Counter 2
        float  MV_Counter3;             // Universal Counter 3
        float  MV_Counter4;             // Universal Counter 4
        double ChargeCapacityTime;      // Charge capacity time
        double DischargeCapacityTime;   // discharge capacity time
        float  MVUD1;                   // Custom user-defined variable values
        float  MVUD2;                   // Custom user-defined variable values
        float  MVUD3;                   // Custom user-defined variable values
        float  MVUD4;                   // Custom user-defined variable values
        float  MVUD5;                   // Custom user-defined variable values
        float  MVUD6;                   // Custom user-defined variable values
        float  MVUD7;                   // Custom user-defined variable values
        float  MVUD8;                   // Custom user-defined variable values
        float  MVUD9;                   // Custom user-defined variable values
        float  MVUD10;                  // Custom user-defined variable values
        float  MVUD11;                  // Custom user-defined variable values
        float  MVUD12;                  // Custom user-defined variable values
        float  MVUD13;                  // Custom user-defined variable values
        float  MVUD14;                  // Custom user-defined variable values
        float  MVUD15;                  // Custom user-defined variable values
        float  MVUD16;                  // Custom user-defined variable values
} THIRD_PARTY_RESUME_DATA;
```

CTI_REQUEST_SCHEDULE_START_EX_FEED_BACK

User-defined data type to hold the start test status of a test that ran through the exception.

```
typedef struct
```

```
{
        BYTE        m_bPrefix[8];                   // required to be filled with  eight 0xDD, mandatory.
        DWORD   m_dwLen;                            /* size in byte : m_dwCmd + m_dwCmd_Extend + m_dwIvChannelGlobalIndex
                                                    + m_btStopAll + m_btReserved1 + m_wCheckSum */
        DWORD   m_dwCmd;                            // required to be filled with 0XBB230004, mandatory, stating this is a start command
        DWORD   m_dwCmd_Extend;                     // required to be filled with 0x000000, mandatory.

        DWORD   m_dwIvChannelGlobalIndex;           // the IV channel index need to be stopped, channel index start from 0
        BYTE        m_btResult;                     // Return result. , See ArbinCommandStartChannelFeed for more information
        BYTE        m_btReserved1[101];             // reserved, required to be filled with 0x00

        WORD        m_wCheckSum                     //checksum: add all fields above byte by byte
} CTI_REQUEST_SCHEDULE_START_EX_FEED_BACK;
```

CTI_REQUEST_GET_RESUME_DATA :

User-defined data type to hold the command and channel number where we can resume the test.

```
typedef struct
{
        BYTE        m_bPrefix[8];                   // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                        /* size in byte : m_dwCmd + m_dwCmd_Extend + ChannelNum
                                                    + channelNo + m_wCheckSum */
        DWORD       m_dwCmd;                        /* required to be filled with 0XCD130003, mandatory, stating this is
                                                    a get resume channel data command */
        DWORD       m_dwCmd_Extend;                 // required to be filled with 0x000000, mandatory.

        UINT        ChannelNum;                     // Number of channels
        ushort      channelNo[];                    // The array size is determined by ChannelNum, the channel number (starting at 0)

        WORD        m_wCheckSum;                    //checksum: add all fields above byte by byte
} CTI_REQUEST_GET_RESUME_DATA;
```

CTI_REQUEST_GET_RESUME_DATA_FEED_BACK :

User-defined data type to hold the status of the test that resumed.

```
typedef struct
{
        BYTE        m_bPrefix[8];                   // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                        /* size in byte : m_dwCmd + m_dwCmd_Extend + ChannelNum
                                                    + channelNo + m_wCheckSum */
        DWORD       m_dwCmd;                        /* required to be filled with 0XCD310003, mandatory, stating this is
                                                    a get resume channel data command */
        DWORD       m_dwCmd_Extend;                 // required to be filled with 0x000000, mandatory.

        UINT        ChannelNum;                     // Number of channels
        THIRD_PARTY_CHANNEL_RESUME_DATA_DESC channelNo[];  // The array size is determined by ChannelNum, get start channel data

        WORD        m_wCheckSum                     //checksum: add all fields above byte by byte
} CTI_REQUEST_GET_RESUME_DATA_FEED_BACK;

typedef struct
{
        UINT        Channel;                        //  the channel number (starting at 0)
        UINT        channelCode;                    //  Code 34: Database query Test Name failed; Code 35: Test Name is empty
                                                    Code 36: Failed to get resume table data
        UINT        StepNum;                        // Number of step
        wchar_t     TestName[72];
```

```
        wchar_t     Schedule[200];                  // The file name of the schedule
        wchar_t     Createor[64];                   // Creator of channel operation
        wchar_t     Comment[64];                    // Run channel comment
        wchar_t     StartTime[64];                  // Schedule start time
        THIRD_PARTY_RESUME_DATA ResumeData; // Resume table data
        THIRD_PARTY_STRING Steps[];                 // The array size is determined by StepNum, Step label
} THIRD_PARTY_CHANNEL_RESUME_DATA_DESC;
typedef struct
{
        UINT Length;                                // String length
        char stringData[];                          // The array size is determined by Length, String content
} THIRD_PARTY_STRING;
typedef struct
{
        UINT Cycle;                                 // Cycle number
        double TestTime;                            // Running test total time
        double StepTime;                            // Current step time
        double CCapacity;                           // Charge capacity
        double DCapacity;                           // Discharge capacity
        double CEnergy;                             // Charge energy
        double DEnergy;                             // Discharge energy

        double TC_Time1;                            /* TC_Time1, TC_Time2, TC_Time3, and TC_Time4 are time counters.
        double TC_Time2;                            Time counters can be used to count the total test time of a group of steps.
        double TC_Time3;                            Further, the time counters can be used as the step termination limit a
        double TC_Time4;                            or logging datlimit. */

        double TC_CCapacity1;                       /* TC_Charge_Capacity1 and TC_Charge_Capacity2 are charge capacity counters.
        double TC_CCapacity2;                       The charge capacity is the capacity when the current is positive */

        double TC_DCapacity1;                       /* TC_Disharge_Capacity1 and TC_Disharge_Capacity2 are discharge capacity counters.
        double TC_DCapacity2;                       The discharge capacity is the capacity when the current is negative. */

        double TC_CEnergy1;                         /* TC_Charge_Energy1 and TC_Charge_Energy2 are charge energy counters.
        double TC_CEnergy2;                         The charge energy is the positive energy value when calculated by the formula
                                                    ∫I*V*dt */

        double TC_DEnergy1;                         /* TC_Discharge_Energy1 and TC_Discharge_Energy2 are discharge energy counters.
        double TC_DEnergy2;                         The discharge energy is the negative energy value when calculated by the formula
                                                    ∫I*V*dt  */

        float  MV_Counter1;                         // Universal Counter 1
        float  MV_Counter2;                         // Universal Counter 2
        float  MV_Counter3;                         // Universal Counter 3
        float  MV_Counter4;                         // Universal Counter 4
        double ChargeCapacityTime;                  // Charge capacity time
        double DischargeCapacityTime;               // discharge capacity time
        float  MVUD1;                               // Custom user-defined variable values
        float  MVUD2;                               // Custom user-defined variable values
        float  MVUD3;                               // Custom user-defined variable values
        float  MVUD4;                               // Custom user-defined variable values
        float  MVUD5;                               // Custom user-defined variable values
        float  MVUD6;                               // Custom user-defined variable values
        float  MVUD7;                               // Custom user-defined variable values
        float  MVUD8;                               // Custom user-defined variable values
        float  MVUD9;                               // Custom user-defined variable values
        float  MVUD10;                              // Custom user-defined variable values
```

```
        float  MVUD11;                          // Custom user-defined variable values
        float  MVUD12;                          // Custom user-defined variable values
        float  MVUD13;                          // Custom user-defined variable values
        float  MVUD14;                          // Custom user-defined variable values
        float  MVUD15;                          // Custom user-defined variable values
        float  MVUD16;                          // Custom user-defined variable values
} THIRD_PARTY_RESUME_DATA;
```

CTI_REQUEST_SCHEDULE_RESUME_EX :

The user-defined data type to hold the command and channel number that resumed the test with the exception.

```
typedef struct
{
        BYTE      m_bPrefix[8];                 // required to be filled with  eight 0xDD, mandatory.
        DWORD     m_dwLen;                      /* size in byte : m_dwCmd + m_dwCmd_Extend + ChannelNum
                                                + channelNo + m_wCheckSum */
        DWORD     m_dwCmd;                      /* required to be filled with 0XCD130004, mandatory, stating this is
                                                a resume channel command */
        DWORD     m_dwCmd_Extend;               // required to be filled with 0x000000, mandatory.

        UINT      ChannelNum;                   // Number of channels
        THIRD_PARTY_CHANNEL_RESUME_DATA_EX channelNo[]        // The array size is determined by ChannelNum

        WORD      m_wCheckSum;                  //checksum: add all fields above byte by byte
} CTI_REQUEST_SCHEDULE_RESUME_EX;
typedef struct
{
        UINT Channel;                           // the channel number (starting at 0)
        wchar_t TestName[72];                   // Test name
        wchar_t Schedule[200];                  // The file name of the schedule
        UINT nSelectStep;                       // Select the first few steps of Schedule to start running
        THIRD_PARTY_RESUME_DATA ResumeData;// Resume table data
} THIRD_PARTY_CHANNEL_RESUME_DATA_EX;
typedef struct
{
        UINT Cycle;                             // Cycle number
        double TestTime;                        // Running test total time
        double StepTime;                        // Current step time
        double CCapacity;                       // Charge capacity
        double DCapacity;                       // Discharge capacity
        double CEnergy;                         // Charge energy
        double DEnergy;                         // Discharge energy

        double TC_Time1;                        /* TC_Time1, TC_Time2, TC_Time3, and TC_Time4 are time counters.
        double TC_Time2;                        Time counters can be used to count the total test time of a group of steps.
        double TC_Time3;                        Further, the time counters can be used as the step termination limit a
        double TC_Time4;                        or logging datlimit. */

        double TC_CCapacity1;                   /* TC_Charge_Capacity1 and TC_Charge_Capacity2 are charge capacity counters.
        double TC_CCapacity2;                   The charge capacity is the capacity when the current is positive */

        double TC_DCapacity1;                   /* TC_Disharge_Capacity1 and TC_Disharge_Capacity2 are discharge capacity counters.
        double TC_DCapacity2;                   The discharge capacity is the capacity when the current is negative. */

        double TC_CEnergy1;                     /* TC_Charge_Energy1 and TC_Charge_Energy2 are charge energy counters.
        double TC_CEnergy2;                     The charge energy is the positive energy value when calculated by the formula
```

```
                                        ∫I*V*dt */

        double TC_DEnergy1;             /* TC_Discharge_Energy1 and TC_Discharge_Energy2 are discharge energy counters.
        double TC_DEnergy2;             The discharge energy is the negative energy value when calculated by the formula
                                        ∫I*V*dt  */

        float  MV_Counter1;             // Universal Counter 1
        float  MV_Counter2;             // Universal Counter 2
        float  MV_Counter3;             // Universal Counter 3
        float  MV_Counter4;             // Universal Counter 4
        double ChargeCapacityTime;      // Charge capacity time
        double DischargeCapacityTime;   // discharge capacity time
        float  MVUD1;                   // Custom user-defined variable values
        float  MVUD2;                   // Custom user-defined variable values
        float  MVUD3;                   // Custom user-defined variable values
        float  MVUD4;                   // Custom user-defined variable values
        float  MVUD5;                   // Custom user-defined variable values
        float  MVUD6;                   // Custom user-defined variable values
        float  MVUD7;                   // Custom user-defined variable values
        float  MVUD8;                   // Custom user-defined variable values
        float  MVUD9;                   // Custom user-defined variable values
        float  MVUD10;                  // Custom user-defined variable values
        float  MVUD11;                  // Custom user-defined variable values
        float  MVUD12;                  // Custom user-defined variable values
        float  MVUD13;                  // Custom user-defined variable values
        float  MVUD14;                  // Custom user-defined variable values
        float  MVUD15;                  // Custom user-defined variable values
        float  MVUD16;                  // Custom user-defined variable values
} THIRD_PARTY_RESUME_DATA;
```

CTI_REQUEST_SCHEDULE_RESUME_EX_FEED_BACK :

The user-defined data type to hold the status of the test that had an exception, and that was resumed.

```
typedef struct
{
        BYTE     m_bPrefix[8];          // required to be filled with  eight 0xDD, mandatory.
        DWORD    m_dwLen;               /* size in byte : m_dwCmd + m_dwCmd_Extend + m_dwIvChannelGlobalIndex
                                        + m_btStopAll + m_btReserved1 + m_wCheckSum */
        DWORD    m_dwCmd;               // required to be filled with 0XBB130002, mandatory, stating this is a resume command
        DWORD    m_dwCmd_Extend;        // required to be filled with 0x000000, mandatory.

        DWORD    m_dwIvChannelGlobalIndex;  // the IV channel index need to be stopped, channel index start from 0
        BYTE     m_btResult;            // Return result., See ArbinCommandResumChanneleFeed for more information
        BYTE     m_btReserved1[101];    // reserved, required to be filled with 0x00

        WORD     m_wCheckSum            //checksum: add all fields above byte by byte
} CTI_REQUEST_SCHEDULE_RESUME_EX_FEED_BACK;
```

CTI_REQUEST_AUTOCALI_START :

The user-defined data type to hold the command to start Auto-calibration.

```
typedef struct
{
        BYTE     m_bPrefix[8];          // required to be filled with  eight 0xDD, mandatory.
        DWORD    m_dwLen;               /* size in byte : m_dwCmd + m_dwCmd_Extend + ChannelNum
                                        + channelNo + m_wCheckSum */
        DWORD    m_dwCmd;               /* required to be filled with 0XCD130004, mandatory, stating this is
```

```
                                        a resume channel command */
        DWORD    m_dwCmd_Extend;        // required to be filled with 0x000000, mandatory.

        BYTE   Reserved[102];           // reserved, required to be filled with 0x00

        WORD     m_wCheckSum;           //checksum: add all fields above byte by byte
} CTI_REQUEST_AUTOCALI_START;
```

CTI_REQUEST_AUTOCALI_START_FEED_BACK :

The user-defined data type to hold the status of the Auto-Calibration command.

```
typedef struct
{
        BYTE       m_bPrefix[8];        // required to be filled with  eight 0xDD, mandatory.
        DWORD   m_dwLen;                /* size in byte : m_dwCmd + m_dwCmd_Extend + m_dwIvChannelGlobalIndex
                                        + m_btStopAll + m_btReserved1 + m_wCheckSum */
        DWORD   m_dwCmd;                // required to be filled with 0XBB130002, mandatory, stating this is a resume command
        DWORD   m_dwCmd_Extend;         // required to be filled with 0x000000, mandatory.

        UINT       m_btResult;          // Return result., See ArbinCommandStartAutomaticCalibrationFeed for more
                                        // information

        WORD     m_wCheckSum            //checksum: add all fields above byte by byte
} CTI_REQUEST_AUTOCALI_START_FEED_BACK;
```

CTI_REQUEST_SET_METAVARIABLE_VALUE :

The user-defined data type to hold the command to Set metavariable value.

```
typedef struct
{
        BYTE       m_bPrefix[8];        // required to be filled with  eight 0xDD, mandatory.
        DWORD    m_dwLen;               /* size in byte : m_dwCmd + m_dwCmd_Extend + ChannelNum
                                        + channelNo + m_wCheckSum */
        DWORD    m_dwCmd;               /* required to be filled with 0XCD130004, mandatory, stating this is
                                        a resume channel command */
        DWORD    m_dwCmd_Extend;        // required to be filled with 0x000000, mandatory.

        DWORD    m_dwIvChannelGlobalIndex;  // IV channel index, channel index start from 0

        INT        MV_Type;             // Meta Code  Type: 1. IV Type
        INT        MV_MetaCode;         // MetaVariable Code
        BYTE       MV_btReserved1[16];  // reserved, required to be filled with 0x00
        INT        MV_ValueType;        // Value Type: 1.float
        float      MV_Data;             // MetaVariable value
        BYTE       MV_btReserved2[16];  // reserved, required to be filled with 0x00

        WORD     m_wCheckSum            //checksum: add all fields above byte by byte
} CTI_REQUEST_SET_METAVARIABLE_VALUE;
```

CTI_REQUEST_SET_METAVARIABLE_VALUE_FEED_BACK :

The user-defined data type to hold the status of Set metavariable value command.

```
typedef struct
{
        BYTE       m_bPrefix[8];        // required to be filled with  eight 0xDD, mandatory.
        DWORD   m_dwLen;                /* size in byte : m_dwCmd + m_dwCmd_Extend + m_dwIvChannelGlobalIndex
                                        + m_btStopAll + m_btReserved1 + m_wCheckSum */
```

```
        DWORD   m_dwCmd;                    // required to be filled with 0XBB130002, mandatory, stating this is a resume command
        DWORD   m_dwCmd_Extend;             // required to be filled with 0x000000, mandatory.

        DWORD   m_dwIvChannelGlobalIndex;   // IV channel index, channel index start from 0
        BYTE    m_btResult;                 // Return result, See ArbinCommandSetMetaVariableFeed for more info
        BYTE    m_btReserved1[101];         // reserved, required to be filled with 0x00

        WORD    m_wCheckSum                 //checksum: add all fields above byte by byte
} CTI_REQUEST_SET_METAVARIABLE_VALUE_FEED_BACK;
```

CTI_REQUEST_UPDATE_METAVARIABLE_ADVANCED

The user-defined data type to hold the command to Set metavariable value.

```
typedef struct
{
        BYTE    m_bPrefix[8];               // required to be filled with  eight 0xDD, mandatory.
        DWORD   m_dwLen;                    /* size in byte : m_dwCmd + m_dwCmd_Extend + ChannelNum
                                            + channelNo + m_wCheckSum */
        DWORD   m_dwCmd;                    /* required to be filled with 0XCD130004, mandatory, stating this is
                                            a resume channel command */
        DWORD   m_dwCmd_Extend;             // required to be filled with 0x000000, mandatory.

        DWORD   m_dwIvChannelGlobalIndex;   // IV channel index, channel index start from 0

        BYTE    m_btReserved1[18];          // reserved, required to be filled with 0x00
        METAVARIABLE_DATA_CH_CODE m_MV_Data[]       // Maximum support 160
        WORD    m_wCheckSum;                //checksum: add all fields above byte by byte
} CTI_REQUEST_UPDATE_METAVARIABLE_ADVANCED;
typedef struct
{
        ushort m_ChannelIndexInGlobal;      // the channel number (starting at 0)
        ushort m_MV_MetaCode;               // MetaVariable Code
        float_t fMV_Value;                  // MetaVariable value
} METAVARIABLE_DATA_CH_CODE;
```

CTI_REQUEST_UPDATE_METAVARIABLE_ ADVANCED _FEED_BACK

The user-defined data type to hold the status of Set metavariable value command.

```
typedef struct
{
        BYTE    m_bPrefix[8];               // required to be filled with  eight 0xDD, mandatory.
        DWORD   m_dwLen;                    /* size in byte : m_dwCmd + m_dwCmd_Extend + m_dwIvChannelGlobalIndex
                                            + m_btStopAll + m_btReserved1 + m_wCheckSum */
        DWORD   m_dwCmd;                    // required to be filled with 0XBB130002, mandatory, stating this is a resume command
        DWORD   m_dwCmd_Extend;             // required to be filled with 0x000000, mandatory.

        DWORD   m_dwIvChannelGlobalIndex;   // IV channel index, channel index start from 0
        BYTE    m_btResult;                 // Return result, See ArbinCommandSetMetaVariableFeed for more info
        BYTE    m_btReserved1[101];         // reserved, required to be filled with 0x00

        WORD    m_wCheckSum                 //checksum: add all fields above byte by byte
} CTI_REQUEST_UPDATE_METAVARIABLE_ADVANCED_FEED_BACK;
```

CTI_REQUEST_GET_SERIAL

The user-defined data type to hold the command to Get Serial number.

```
typedef struct
```

```
{
        BYTE        m_bPrefix[8];           // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                /* size in byte : m_dwCmd + m_dwCmd_Extend + ChannelNum
                                            + channelNo + m_wCheckSum */
        DWORD       m_dwCmd;                /* required to be filled with 0XCD130004, mandatory, stating this is
                                            a resume channel command */
        DWORD       m_dwCmd_Extend;         // required to be filled with 0x000000, mandatory.

        DWORD       m_dwGetSerialNum;       // Do not use

        BYTE        m_btResult;             // Do not use
        BYTE        m_btReserved1[101];     // reserved, required to be filled with 0x00

        WORD        m_wCheckSum;            //checksum: add all fields above byte by byte
} CTI_REQUEST_GET_SERIAL;
```

CTI_REQUEST_GET_SERIAL_FEED_BACK :

The user-defined data type to hold the status of the Get Serial number command.

```
typedef struct
{
        BYTE        m_bPrefix[8];           // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                /* size in byte : m_dwCmd + m_dwCmd_Extend + m_dwIvChannelGlobalIndex
                                            + m_btStopAll + m_btReserved1 + m_wCheckSum */
        DWORD       m_dwCmd;                // required to be filled with 0XBB130002, mandatory, stating this is a resume command
        DWORD       m_dwCmd_Extend;         // required to be filled with 0x000000, mandatory.

        DWORD       m_dwGetSerialNum;       // Serial number

        BYTE        m_btResult;             // Return result.
        BYTE        m_btReserved1[101];     // reserved, required to be filled with 0x00

        WORD        m_wCheckSum             //checksum: add all fields above byte by byte
} CTI_REQUEST_GET_SERIAL_FEED_BACK;
```

CTI_REQUEST_SCHEDULE_JUMP :

The user-defined data type to hold the command to Schedule Jump.

```
typedef struct
{
        BYTE        m_bPrefix[8];           // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                /* size in byte : m_dwCmd + m_dwCmd_Extend + ChannelNum
                                            + channelNo + m_wCheckSum */
        DWORD       m_dwCmd;                /* required to be filled with 0XCD130004, mandatory, stating this is
                                            a resume channel command */
        DWORD       m_dwCmd_Extend;         // required to be filled with 0x000000, mandatory.

        DWORD       stepIndex;              // Index of the step to jump to
        DWORD       ChannelIndex;           // IV channel index, channel index start from 0

        BYTE        m_btReserved1[101];     // reserved, required to be filled with 0x00

        WORD        m_wCheckSum;            //checksum: add all fields above byte by byte
} CTI_REQUEST_SCHEDULE_JUMP;
```

CTI_REQUEST_SCHEDULE_JUMP_FEED_BACK :

The user-defined data type to hold the status of the Schedule-Jump command.

```
typedef struct
{
        BYTE        m_bPrefix[8];           // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                /* size in byte : m_dwCmd + m_dwCmd_Extend + m_dwIvChannelGlobalIndex
                                            + m_btStopAll + m_btReserved1 + m_wCheckSum */
        DWORD       m_dwCmd;                // required to be filled with 0XBB130002, mandatory, stating this is a resume command
        DWORD       m_dwCmd_Extend;         // required to be filled with 0x000000, mandatory.

        DWORD       dwIvChannelGlobalIndex; // IV channel index, channel index start from 0

        BYTE        m_btResult;             // Return result.
        BYTE        m_btReserved1[101];     // reserved, required to be filled with 0x00

        WORD        m_wCheckSum             //checksum: add all fields above byte by byte
} CTI_REQUEST_SCHEDULE_JUMP_FEED_BACK;
```

CTI_REQUEST_NEW_FOLDER :

The user-defined data type to hold the file path and flag for NEW a folder along with the command.

```
typedef struct
{
        BYTE        m_bPrefix[8];           // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                /* size in byte : m_dwCmd + m_dwCmd_Extend + Type
                                            + FilePath + m_wCheckSum */
        DWORD       m_dwCmd;                /* required to be filled with 0XCC130005, mandatory, stating this is
                                            a new folder command */
        DWORD       m_dwCmd_Extend;         // required to be filled with 0x000000, mandatory.

        wchar_t     FilePath[1024];         // Request new directory.

        WORD        m_wCheckSum;            //checksum: add all fields above byte by byte
} CTI_REQUEST_NEW_FOLDER;
```

CTI_REQUEST_NEW_FOLDER_FEED_BACK :

The user-defined data type to hold the status of the create new folder command.

```
typedef struct
{
        BYTE        m_bPrefix[8];           // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                /* size in byte : m_dwCmd + m_dwCmd_Extend + Result
                                            + m_wCheckSum */
        DWORD       m_dwCmd;                /* required to be filled with 0XCC310005, mandatory, stating this is
                                            a new or delete folder/file command */
        DWORD       m_dwCmd_Extend;         // required to be filled with 0x000000, mandatory.

        UINT        Result;                 // Return result, See ArbinCommandNewFeed for more information

        WORD        m_wCheckSum             //checksum: add all fields above byte by byte
} CTI_REQUEST_NEW_FOLDER_FEED_BACK;
```

CTI_REQUEST_DELETE_FOLDER :

The user-defined data type to hold the file path and flag for DELETE a folder along with the command.

```
typedef struct
```

```
{
        BYTE        m_bPrefix[8];           // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                /* size in byte : m_dwCmd + m_dwCmd_Extend + Type
                                            + FilePath + m_wCheckSum */
        DWORD       m_dwCmd;                /* required to be filled with 0XCC130006, mandatory, stating this is
                                            a delete folder/file file command */
        DWORD       m_dwCmd_Extend;         // required to be filled with 0x000000, mandatory.

        wchar_t     FilePath[1024];         // Request delete directory, file path

        WORD        m_wCheckSum;            //checksum: add all fields above byte by byte
} CTI_REQUEST_DELETE_FOLDER;
```

CTI_REQUEST_DELETE_FOLDER_FEED_BACK :

User-defined data type to hold the status of the Delete folder command.

```
typedef struct
{
        BYTE        m_bPrefix[8];           // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                /* size in byte : m_dwCmd + m_dwCmd_Extend + Result
                                            + m_wCheckSum */
        DWORD       m_dwCmd;                /* required to be filled with 0XCC310006, mandatory, stating this is
                                            a delete folder/file command */
        DWORD       m_dwCmd_Extend;         // required to be filled with 0x000000, mandatory.

        UINT        Result;                 // Return result, See ArbinCommandDeleteFeed for more information

        WORD       m_wCheckSum              //checksum: add all fields above byte by byte
} CTI_REQUEST_NEW_OR_DELETE_FOLDER_FEED_BACK;
```

CTI_REQUEST_SEND_MSG_TO_CTI :

User-defined data type to hold the file path and flag for DELETE a folder along with the command.

```
typedef struct
{
        BYTE        m_bPrefix[8];           // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                /* size in byte : m_dwCmd + m_dwCmd_Extend + Type
                                            + FilePath + m_wCheckSum */
        DWORD       m_dwCmd;                /* required to be filled with 0XCC130006, mandatory, stating this is
                                            a delete folder/file file command */
        DWORD       m_dwCmd_Extend;         // required to be filled with 0x000000, mandatory.

        wchar_t     Msg[1024];              // Rich Text Message, Text format: [DateTime]<-->[RichText]

        WORD        m_wCheckSum;            //checksum: add all fields above byte by byte
} CTI_REQUEST_SEND_MSG_TO_CTI;
```

CTI_REQUEST_SEND_MSG_TO_CTI_FEED_BACK :

User-defined data type to hold the status of the Delete folder command.

```
typedef struct
{
        BYTE        m_bPrefix[8];           // required to be filled with  eight 0xDD, mandatory.
        DWORD       m_dwLen;                /* size in byte : m_dwCmd + m_dwCmd_Extend + Result
                                            + m_wCheckSum */
        DWORD       m_dwCmd;                /* required to be filled with 0XCC310006, mandatory, stating this is
```

```
                                        a delete folder/file command */
        DWORD     m_dwCmd_Extend;       // required to be filled with 0x000000, mandatory.

        UINT      Result;               // Return result, See ArbinCommandSendMsgToCTIFeed for more information

        WORD      m_wCheckSum           //checksum: add all fields above byte by byte
} CTI_REQUEST_SEND_MSG_TO_CTI_FEED_BACK;
```

ArbinCTI.dll Major Function List

Connecting

Connecting to CTI is done through the use of ArbinClient. In order to establish a connection, the Monitor and Control window must be open on the cycler's PC. Below is a listing of the functions available:

int ConnectAsync(string strIPV4, int Port, int TimeOut, out int ErrorCode)

Explanation

Establish a connection to the cycler through the instance.

Arguments

string strIPV4: IP Address of cycler to connect to. Must be supplied in the format "xxx.xxx.xxx.xxx"
int port: Port number on which to connect.
Use port 9031 for commands like: Get Channel Info, Start Channel, Stop, Assign Sdu, Resume, Set MV
Use port 9032 for commands like: Browse Directory, Upload, Download, New Or Delete
int Timeout: Connection timeout length, in milliseconds. It is recommended to set this to 30 seconds.
int ErrorCode: Error code corresponding to System.Net.Sockets.SocketException.SocketErrorCode. Check https://docs.microsoft.com/en-us/windows/win32/winsock/windows-sockets-error-codes-2 for the error code to diagnose the problem.

Return Values

Value	Explanation
0	Successfully connected to the cycler
-1	A connection is already established
-2	Error returned. Check ErrorCode
-3	Encountered an unknown error
-4	Connection attempt timed out

bool IsConnected()

Explanation

Returns whether or not the ArbinClient instance is connected to a cycler.

Arguments

None

Return Values

Returns true if this instance is connected to a cycler and returns false otherwise.

void ShutDown()

Explanation

Shut down the connection with the cycler.

Arguments

None

Return values

None

Wrapper Library (ArbinControl)

An instance of ArbinControl may be used to issue commands to the cycler. ArbinControl will create the command and use an instance of ArbinClient to send the command to the cycler and has callback functions for retrieving feedback from the cycler to check for success or failure. ArbinControl acts as a wrapper on SendPackHelper, which creates the command to send to the cycler. In order for any command to succeed, the monitor and control window must be open on the cycler's PC, and CTI must be enabled.

NOTE: All channels passed as arguments are 0-indexed. I.E., physical IV channel 1 would be passed as 0.

Initialization and Exiting

These functions are used to set up for issuing CTI commands and for receiving messages from the cycler in response to CTI commands.

void Start()

Explanation

Begins CTI processing, allowing for communication between the cycler and CTI. It creates a new thread for processing commands sent by a cycler.

Arguments

None

Return Values

None

void ListenSocketRecv(IArbinSocket socket)

Explanation

Sets *socket* to use ArbinControl's internal handler for receiving messages from the client.

Arguments

IArbinSocket socket: An instance of ArbinClient

Return Values

None

void Exit()

Explanation

Shuts down the thread which receives commands from the cycler. Called by the finalizer for ArbinControl.

Arguments

None

Return Values

None

Login Commands

These commands are used for logging in to the CTI function on the cycler. You must log in in order to issue commands to the cycler.

bool PostLogicConnect(IArbinSocket socket, bool bSetKickOut)

Explanation

Sets whether or not the logged-in user is able to be kicked out of the cycler.

Arguments

IArbinSocket socket: an instance of ArbinClient

bool bSetKickOut: Set true to allow the current user to be kicked out.

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

abstract void OnLogicConnectFeedBack(ArbinCommandLogicConnectFeed cmd)

Explanation

The user-defined handler for the message returned by the cycler in response to PostLogicConnect. Check the message to determine if user privileges have been correctly set.

bool PostUserLogin(IArbinSocket socket, string strUser, string strPassword)

Explanation

Log in to the cycler's CTI function using the supplied username and password.

Arguments

IArbinSocket socket: an instance of ArbinClient

String strUser: Username for the CTI account to use

String strPassword: Password for the CTI account to use

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

abstract void OnUserLoginFeedBack(ArbinCommandLoginFeed cmd)

Explanation

The user-defined callback function for the cycler's response to PostUserLogin. Use this to determine if logging in was successful.

Arguments

ArbinCommandLoginFeed cmd: Message returned by the cycler in response to PostUserLogin. See **错误!未找到引用源。** for more information.

Return Values

None

Arguments

ArbinCommandLogicConnectFeed cmd: The message sent by the cycler in response to PostLogicConnect. See **错误!未找到引用源。** for more information.

Return Values

None

Schedule Commands

These commands are used for beginning schedules on the cycler. They allow for addressing individual channels, groups of specific channels or all channels depending on the parameters passed.

bool PostAssignSchedule(IArbinSocket socket, string ScheduleName, string Barcode, float Capacity, float MVUD1, float MVUD2, float MVUD3, float MVUD4, bool AllAssign, int ChannelIndex)

Explanation

Sends the command to assign the named schedule to the listed channel, or to all channels, with the provided barcode, capacity, and user-defined variables.

Arguments

IArbinSocket socket: an instance of ArbinClient
string ScheduleName: the name of the schedule to assign
string Barcode: the barcode to assign
float Capacity: the capacity to assign
float MVUD1, MVUD2, MVUD3, MVUD4: user-defined meta-variables to assign
bool AllAssign: If true, assigns these schedule settings to all channels. Defaults to true.
int ChannelIndex: If AllAssign is false, assigns the schedule settings to the selected channel. ChannelIndex is the 0-indexed

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

abstract void OnAssignScheduleFeedBack(ArbinCommandAssignScheduleFeed cmd)

Explanation

The user-defined callback function for the message returned by the cycler after sending the PostAssignSch command. Check the result stored in cmd to determine the status of the Assign-Schedule command.

Arguments

ArbinCommandAssignScheduleFeed cmd: The message sent from the cycler in response to the assign schedule command. See ArbinCommandAssignScheduleFeedfor more information on handling this object.

Return Values

None

bool PostJumpChannel(IArbinSocket socket, int stepNum, int channelIndex)

Explanation

Send a command to the cycler to jump the step in the schedule.

Arguments

IArbinSocket socket: an instance of ArbinClient

int stepNum: step number need to jump to
int channelIndex: channel need to jump

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

abstract void OnJumpChannelFeedBack(ArbinCommandJumpChannelFeed cmd)

Explanation

The user-defined callback function for handling the message returned by the cycler in response to PostJumpChannel.

Arguments

ArbinCommandJumpChannelFeed cmd: The message returned by the cycler in response to the PostJumpChannel command. See ArbinCommandJumpChannelFeed for more information.

bool PostResumeChannel(IArbinSocket socket, bool AllResume, int ChannelIndex)

Explanation

Send a command to resume all channels or a single specific channel.

Arguments

IArbinSocket socket: an instance of ArbinClient
bool AllResume: Set to true to resume all channels, or false to resume a single channel.
int ChannelIndex: Specify a single channel to resume. Only takes effect is AllResume is false.

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

abstract void OnResumeChannelFeedBack(ArbinCommandResumChanneleFeed cmd)

Explanation

The user-defined callback function for handling messages sent from the cycler following a PostResumeChannel command. Evaluate cmd to find specific information on the status resume command.

Arguments

ArbinCommandResumChanneleFeed cmd: Message returned by the cycler. See ArbinCommandResumChanneleFeed for more information.

Return Values

None

bool PostResumeChannelEx(IArbinSocket socket, List<StartResumeEx> resumeExs)

Explanation

Send a resume command to the cycler to start the channels in Channels using the arguments passed in resumeExs.

Arguments

IArbinSocket socket: an instance of ArbinClient

List<StartResumeEx> resumeExs: a list containing information for each channel, in the same order as Channels. All channels start under the first item's TestNames string, but each channel begins using the corresponding Schedules string.

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

bool PostSetMetaVariable (IArbinSocket socket, int nChannel, int mvType, int mvMetaCode, int mvValueType, object mvValue)

Explanation

Sends a command to change the value of Meta Variables(Example: MV_UD1, MV_UD2)

Arguments

IArbinSocket socket: an instance of ArbinClient
int nChannel: 0 indexed channels to set Meta Variable.
Int mvType: Type of the MetaVariable
Int mvMetaCode: Metacode of the Meta Variable needs to be changed
Int mvValueType: Meta variable value type.
Object mvValue: Value of the Meta Variable

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

abstract void OnSetMVFeedBack (ArbinCommandSetMetaVariableFeed cmd)

Explanation

Feedback is returned by the cycler when a PostSetMetaVariable command is issued.

Arguments

ArbinCommandSetMetaVariableFeed cmd: Information sent by the cycler about the Set Metavaiable result.

Return Values

None

bool PostUpdateMetaVariableAdvanced (IArbinSocket socket, List<MetaVariableInfo> metaVariableList, out int error)

Explanation

Sends a command to change the value of Meta Variables(Example: MV_UD1, MV_UD2)

Arguments

IArbinSocket socket: an instance of ArbinClient
List<MetaVariableInfo> a list containing information for each metaVariable.
int error: Return error code: 1:Means more than 160 data

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

abstract void OnUpdateMetaVariableAdvancedFeedBack (ArbinCommandUpdateMetaVariableAdvancedFeed cmd)

Explanation

Feedback is returned by the cycler when a PostUpdateMetaVariableAdvanced command is issued.

Arguments

ArbinCommandUpdateMetaVariableAdvancedFeed cmd: Information sent by the cycler about the Update Metavaiable Advanced result.

Return Values

None

bool PostStartChannel(IArbinSocket socket, string TestName, List<ushort> Channels)

Explanation

Arguments

IArbinSocket socket: an instance of ArbinClient

String TestName: The name assigned to the test

List<ushort> Channels: A list of channels to start

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

abstract void OnStartChannelFeedBack(ArbinCommandStartChannelFeed cmd)

Explanation

The user-defined callback function for the message returned by the cycler after sending the PostStartChannel command. Check the result stored in cmd to determine the status of the channels following the start channels command.

Arguments

ArbinCommandStartChannelFeed cmd: The message sent from the cycler in response to the start channel command. See ArbinCommandStartChannelFeed for more information on handling this object.

Return Values

None

bool PostStartChannelEx(IArbinSocket socket, List<StartResumeEx> resumeEx, string Creators, string Comments)

Explanation

Starts the channels in the Channels list, using the information stored in resumeEx to initialize each channel's test information. Use **错误!未找到引用源。** to get the feedback from the cycler.

Arguments

IArbinSocket socket: an instance of ArbinClient

List<StartResumeEx> resumeEx: a list containing information for each channel, in the same order as Channels. All channels start under the first item's TestNames string, but each channel begins using the corresponding Schedules string to determine the schedule to run. See StartResumeEx for more information on usage.

string Creators: string containing the list of creators of this test/schedule
string Comments: string containing comments about this test/schedule

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

bool PostContinueChannel(IArbinSocket socket, List<ushort> Channels)

Explanation

Arguments

IArbinSocket socket: an instance of ArbinClient
List<ushort> Channels: A list of channels to continue

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

abstract void OnContinueChannelFeedBack(ArbinCommandContinueChannelFeed cmd)

Explanation

The user-defined callback function for the message returned by the cycler after sending the PostContinueChannel command. Check the result stored in cmd to determine the status of the channels following the start channels command.

Arguments

ArbinCommandStartChannelFeed cmd: The message sent from the cycler in response to the continue channel command. See ArbinCommandContinueChannelFeed for more information on handling this object.

Return Values

None

bool PostStopChannel(IArbinSocket socket, int nChannel, bool bStopAll)

Explanation

Send a command to the cycler to stop the selected channel or to stop all channels.

Arguments

IArbinSocket socket: An instance of ArbinClient
int nChannel: The 0-indexed channel number to stop
bool bStopAll: Set to true to stop all channels, or set to false to stop only the channel specified by nChannel

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

void OnStopChannelFeedBack(ArbinCommandStopChannelFeed cmd)

Explanation

The user-defined callback for handling the message sent by the cycler in response to PostStopMsg.

Arguments

ArbinCommandStopChannelFeed cmd: The message sent by the cycler in response to PostStopMsg. See ArbinCommandStopChannelFeed for more information.

Return Values

None

Request Info Commands

These commands are used for requesting information from cycler. They would reuturn value such as channel status.

bool PostGetChannelsData(IArbinSocket socket, uint NeedType, short OnlyGetChannelNumber, ArbinCommandGetChannelFeed.GET_CHANNEL_TYPE GetChannelType)

Explanation

Send a command to the cycler to get the status of its channels.

Arguments

IArbinSocket socket: An instance of ArbinClient
uint NeedType: Specifies that the cycler should the information on the chosen types. A list of types to return is built by OR-ing the possible options. Defaults to include CAN-BMS, SMB, or AUX. Possible valid values are THIRD_PART_GET_CHANNELS_INFO_NEED_TYPE_BMS, THIRD_PART_GET_CHANNELS_INFO_NEED_TYPE_SMB, and THIRD_PART_GET_CHANNELS_INFO_NEED_TYPE_AUX, and values obtained by OR-ing these together.
short OnlyGetChannelNumber: Only retrieve the status of the specified channel
GET_CHANNEL_TYPE GetChannelType: Gets the status of only the type of channel specified. Possible types are GET_CHANNEL_TYPE.ALLCHANNEL for all channels, GET_CHANNEL_TYPE.RUNNING for only running channels, and GET_CHANNEL_TYPE.UNSAFE for only unsafe channels.

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

void OnGetChannelsDataFeedBack(ArbinCommandGetChannelDataFeed cmd)

Explanation

The user-defined callback for handling the message received from the cycler in response to PostGetChannels. Use this to process the state of the channels requested.

Arguments

ArbinCommandGetChannelDataFeed cmd: Message returned by the cycler in response to PostGetChannelsData. See ArbinCommandGetChannelDataFeed for more information.

Return Values

None

bool PostGetResumeData(IArbinSocket socket, List<ushort> channels)

Explanation

Send a request to the cycler to retrieve the resume status of the channels.

Arguments

IArbinSocket socket: an instance of ArbinClient

List<ushort> channels: List of channels to retrieve resume status information from the cycler.

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

abstract void OnGetResumeDataBack(ArbinCommandGetResumeDataFeed cmd)

Explanation

The user-defined callback function for PostGetResumeData. Gives the results of the command to get the resume status. Currently only gives the channel number.

Arguments

ArbinCommandGetResumeDataFeed cmd: Message sent by the cycler containing the requested resume information. See 错误!未找到引用源。 for more information.

Return Values

None

bool PostGetSerialNumber(IArbinSocket socket)

Explanation

Send a command to the cycler to get the serial number of cycler.

Arguments

IArbinSocket socket: an instance of ArbinClient

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

Abstract void OnGetSerialNumberFeedBack(ArbinCommandGetSerialNumberFeed cmd);

Explanation

The user-defined callback function for handling the message returned by the cycler in response to PostGetSerialNumber.

Arguments

ArbinCommandGetSerialNumberFeed cmd: The message returned by the cycler in response to the PostGetSerialNumbercommand. See ArbinCommandGetSerialNumberFeed for more information.

bool PostGetStartData(IArbinSocket socket, List<ushort> channels)

Explanation

Sends a command to the cycler to return the information about the starting status of each channel in the channels list. Use abstract void OnGetStartDataBack(ArbinCommandGetStartDataFeed cmd) to process the returned information.

Arguments

IArbinSocket socket: an instance of ArbinClient

List<ushort> channels: 0 indexed channels to get start status data.

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

abstract void OnGetStartDataBack(ArbinCommandGetStartDataFeed cmd)

Explanation

Feedback is returned by the cycler when a PostGetStartData command is issued. Currently only returns information on channel number.

Arguments

ArbinCommandGetStartDataFeed cmd: Information sent by the cycler about the start channel status.

Return Values

None

File Operation Commands

These commands are used for browsing the file system on the cycler's PC and sending files to the cycler's PC, and retrieving files from the cycler's PC.

bool PostBrowseDirectory(IArbinSocket socket, string strPath)

Explanation

Sends a command to the cycler to retrieve a list of files stored at strPath, if the path exists.

Arguments

IArbinSocket socket: an instance of ArbinClient

String strPath: the directory path in which the console will search for files.

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

abstract void OnBrowseDirectoryBack(ArbinCommandBrowseDirectoryFeed cmd)

Explanation

The user-defined callback for handling the message from the cycler in response to PostBrowseFile. Use this to process the information gathered from the cycler.

Arguments

ArbinCommandBrowseDirectoryFeed cmd: The message sent by the cycler in response to PostBrowseFile. See ArbinCommandBrowseDirectoryFeed for more information.

Return Values

None

bool PostDeleteFolder (IArbinSocket socket, string strPath)

Explanation

Send a command to the cycler to delete the file or folder specified by strPath.

Arguments

IArbinSocket socket: an instance of ArbinClient

string strPath: the path of the folder to be deleted.

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

abstract void OnDeleteFolderBack(ArbinCommandDeleteFolderFeed cmd)

Explanation

The user-defined callback function for handling the message returned by the cycler in response to PostNewFolder.

Arguments

ArbinCommandDeleteFolderFeed cmd: The message returned by the cycler in response to the PostDeleteFolder command. See ArbinCommandDeleteFolderFeed for more information.

bool PostDownloadFile(IArbinSocket socket, string strPath, double time)

Explanation

Send a request to the cycler to send the file at strPath to the PC running CTI.

Arguments

IArbinSocket socket: an instance of ArbinClient
string strPath: The path of the file being retrieved from the cycler.
double time: The timestamp associated with this command.

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

abstract void OnDownLoadFileBack(ArbinCommandDownLoadFileFeed cmd)

Explanation

The user-defined callback function for handling the cycler's response to PostDownload. Use this to handle the file data sent by the cycler's PC.
NOTE: The file data is sent in chunks. cmd contains information on the chunk index and the number of chunks being sent. cmd allows for checking the validity of the data sent by the cycler. This callback must be implemented to save the file to the CTI computer.

Arguments

ArbinCommandDownLoadFileFeed cmd: The message sent from the cycler in response to PostDownload. It contains metadata about the file retrieved from the cycler's PC, such as MD5, file length, and the data itself. See ArbinCommandDownLoadFileFeed for more information.

Return Values

None

bool PostNewFolder(IArbinSocket socket, string strPath)

Explanation

Send a command to the cycler to delete the file or folder specified by strPath, or create a file or folder at strPath.

Arguments

IArbinSocket socket: an instance of ArbinClient
string strPath: the path of the file or folder to be deleted or of the file or folder to create.

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

abstract void OnNewFolderBack(ArbinCommandNewFolderFeed cmd)

Explanation

The user-defined callback function for handling the message returned by the cycler in response to PostNewFolder.

Arguments

ArbinCommandNewFolderFeed cmd: The message returned by the cycler in response to the PostNewFolder command. See ArbinCommandNewFolderFeed for more information.

bool PostNewOrDelete(IArbinSocket socket, string strPath, ArbinCommandNewOrDeleteFeed.NEW_OR_DELETE_TYPE uType)

Explanation

Send a command to the cycler to delete the file or folder specified by strPath, or create a file or folder at strPath.

Arguments

IArbinSocket socket: an instance of ArbinClient

string strPath: the path of the file or folder to be deleted or of the file or folder to create.

ArbinCommandNewOrDeleteFeed.NEW_OR_DELETE_TYPE uType: Specify whether to create or delete. Use NEW_OR_DELETE_TYPE.CTI_DELETE to delete a file or folder, or CTI_NEW to create a file or folder.

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

abstract void OnNewOrDeleteBack(ArbinCommandNewOrDeleteFeed cmd)

Explanation

The user-defined callback function for handling the message returned by the cycler in response to PostNewOrDelete.

Arguments

ArbinCommandNewOrDeleteFeed cmd: The message returned by the cycler in response to the PostNewOrDelete command. See **错误!未找到引用源。** for more information.

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

bool PostUpLoadFile(IArbinSocket socket, string strPath, byte[] Filedata, double time, uint uGeneralPackage, uint PackageIndex)

Explanation

Sends a command to the cycler to allow CTI to begin uploading a file to the cycler. The file data is passed as a byte array.

NOTE: When uploading a file, it is necessary to chunk the file manually.

Arguments

IArbinSocket socket: An instance of ArbinClient.

string strPath: The path to save the file data to.

byte[] Filedata: The binary data of the file to send, in byte array form.

double time: The timestamp of the upload operation.

uint uGeneralPackage: The number of data packages to send to transmit the complete file.

uint PackageIndex: The current package index.

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

abstract void OnUpLoadFileBack(ArbinCommandUpLoadFileFeed cmd)

Explanation

The user-defined callback for the message sent from the cycler in response to PostUpLoad. Use this to check the success or failure of the file upload command.

Arguments

ArbinCommandUpLoadFileFeed cmd: The message sent by the cycler. See ArbinCommandUpLoadFileFeed for more information.

Return Values

None

Calibration Commands

These commands are used to begin autocalibration on the cycler's PC.

NOTE: When issuing these commands, the calibration window must already be opened on the cycler's PC.

bool PostStartAutomaticCalibration(IArbinSocket socket)

Explanation

Sends a command to begin autocalibration according to a configuration file stored on the cycler's PC.

Arguments

IArbinSocket socket: an instance of ArbinClient

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

abstract void OnStartAutomaticCalibrationBack(ArbinCommandStartAutomaticCalibrationFeed cmd)

Explanation

The user-defined callback function for the message returned by the cycler in response to PostAutomaticCalibration.

Arguments

ArbinCommandStartAutomaticCalibrationFeed cmd: The message returned by the cycler. See ArbinCommandStartAutomaticCalibrationFeed for more information.

Return Values

None

Messaging Commands

These commands are used to send text messages to the cycler. The cycler will display the message in a dialog box.

bool PostSendMsgToCTI(IArbinSocket socket, string msg)

Explanation

It opens a dialog window on the cycler's PC with the formatted text provided by msg. Msg must be provided in rich-text format. Otherwise, garbage will be displayed. Allows for changing font, size, and color of the text.

Arguments

IArbinSocket socket: an instance of ArbinClient

String msg: the message to send to the cycler. Must be provided in a rich-text format. Otherwise, garbage will be displayed. Using rich-text allows for changing the font, color, and size of the text displayed.

Return Values

Returns false if the *socket* was unable to send the command to the cycler.

abstract void OnSendMsgToCTIBack(ArbinCommandSendMsgToCTIFeed cmd)

Explanation

Callback function for PostSendMsgToCTI. Provide a callback function to execute when the command returns in order to do some processing.

Arguments

ArbinCommandSendMsgToCTIFeed cmd: Command returned by console to CTI.

Return Values

None

Command and Feedback Structures and Enumerations

This section contains information on the classes and structures that are sent by the client to indicate success/failure and to communicate information relevant to the command.

Command Function Arguments

StartResumeEx

```
public struct StartResumeEx
{
    public uint   nChannelIndex;
    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 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;
  }
```

Usage:

TestNames is the test name to use when starting the test. Schedules is the schedule to use for the test. nSelectSteps specifies the step on which to begin.

NOTE: When passing as an argument to PostStartEx, ONLY TestNames, Schedules, and nSelectSteps are used in beginning the test.

MetaVariableInfo

```
  public class MetaVariabldeInfo
  {
    public ushort   m_ChannelIndexInGlobal;
    public ushort   m_MV_MetaCode;
    public float     fMV_Value;

    public MetaVariableInfo()
    {
    }
  }
```

Usage:

ChannelIndexInGlobal is which channel's meta variables to update. m_MV_MetaCode is update the code of the meta variable. fMV_Value is update the value of the meta variable.

NOTE: When passing as an argument to PostUpdateMetaVariableAdvanced.

Feedback from cycler

ArbinCommandAssignScheduleFeed

```
  public class ArbinCommandAssignScheduleFeed : ArbinCommand
  {
        public enum ASSIGN_TOKEN
    {
      CTI_ASSIGN_SUCCESS,
      CTI_ASSIGN_INDEX = 0x10,
      CTI_ASSIGN_ERROR,
      CTI_ASSIGN_SCHEDULE_NAME_EMPTY_ERROR,
      CTI_ASSIGN_SCHEDULE_NOT_FIND_ERROR,
      CTI_ASSIGN_CHANNEL_RUNNING_ERROR,
```

```
        CTI_ASSIGN_CHANNEL_DOWNLOAD_ERROR,
        CTI_ASSIGN_BACTH_FILE_OPENED,
        CTI_ASSIGN_SDU_CANNOT_ASSDIGN_SCHEDULE,
        CTI_ASSIGN_SDU_SAVE_FAILED,
    };

    public ASSIGN_TOKEN Result;
  }
```

Usage:

The result contains the success message from the cycler. Refer to ArbinCommandAssignScheduleFeed.ASSIGN_TOKEN for the meaning of the value.

ArbinCommandBrowseDirectoryFeed

```
  public class ArbinCommandBrowseDirectoryFeed : ArbinCommand
  {
    public enum BROWSE_DIRECTORY_RESULT
    {
        CTI_BROWSE_DIRECTORY_SUCCESS = 1,
        CTI_BROWSE_SCHEDULE_SUCCESS,
        CTI_BROWSE_DIRECTORY_FAILED
    };

    public struct DirFileInfo
    {
        public uint Type;
        public string DirFileName;
        public int dwSize;
        public string wcModified;
    };

    public BROWSE_DIRECTORY_RESULT Result;
    public List<DirFileInfo> DirFileInfoList = null;
  }
```

Usage:

Result contains the success message from the cycler. Refer to ArbinCommandBrowseDirectoryFeed.BROWSE_DIRECTORY_RESULT for the meaning of the value. DirFileInfoList is a list containing information on everything listed under the requested directory, including other directories.

ArbinCommandDeleteFolderFeed

```
public class ArbinCommandDeleteFolderFeed : ArbinCommand
{
    /// <summary>
    /// Result Type
    /// </summary>
    public enum RESULT_TYPE
    {
        CTI_NEW_SUCCESS = 1,
        CTI_DELETE_SUCCESS,
        CTI_NEW_FAILED,
        CTI_NEW_FAILED_ADD_FOLDER,
        CTI_DELETE_FAILED,
        CTI_DELETE_FAILED_EXTENSION,
        CTI_DELETE_FAILED_TEXT_RUNNING,
        CTI_DELETE_FAILED_EXIST
    };

    public RESULT_TYPE Result;    //(RESULT_TYPE) Return result
}
```

Usage:

Result contains the success message from the cycler. Refer to ArbinCommandDeleteFolderFeed. RESULT_TYPE for the meaning of the value.

ArbinCommandDownLoadFileFeed

```
public class ArbinCommandDownLoadFileFeed : ArbinCommand
{
    public enum DOWNLOAD_RESULT
    {
        CTI_DOWNLOAD_SUCCESS = 1,
        CTI_DOWNLOAD_FAILED,
        CTI_DOWNLOAD_MD5_ERR,
        CTI_DOWNLOAD_MAX_LENGTH_ERR
    };

    public DOWNLOAD_RESULT Result;
    public string m_MD5;
    public ulong dwFileLength;
    public byte[] byData = null;
    public double DownloadTime;
    public double UploadTime;
```

```
    public uint uGeneralPackage;
    public uint uPackageIndex;
  }
```

Usage:

Result contains the success message from the cycler. Refer to ArbinCommandDownLoadFileFeed.DOWNLOAD_RESULT for the meaning of the value.

m_MD5 contains the MD5 checksum value of the chunk. Compare this with the computed MD5 for the chunk to ensure correct data transmission. If the MD5 values do not match, reject the transmission, and try again.

byData contains the data for this chunk.

uGeneralPackage is the number of packages being sent to construct the file.

uPackageIndex is the index of the current chunk within the file.

ArbinCommandGetChannelDataFeed

```
  public class ArbinCommandGetChannelDataFeed: ArbinCommand
  {
    public enum GET_CHANNELS_INFO_NEED_TYPE
    {
      THIRD_PARTY_GET_CHANNELS_INFO_NEED_TYPE_BMS = 0x100,
      THIRD_PARTY_GET_CHANNELS_INFO_NEED_TYPE_SMB = 0x200,
      THIRD_PARTY_GET_CHANNELS_INFO_NEED_TYPE_AUX = 0x400,
    };

    public enum GET_CHANNEL_TYPE
    {
      ALLCHANNEL = 1,
      RUNNING = 2,
      UNSAFE = 3
    };

    public enum ChannelStatus
    {
      Idle,
      Transition,
      Charge,
      Discharge,
      Rest,
      Wait,
      External_Charge,
      Calibration,
      Unsafe,
```

```
    Pulse,
    Internal_Resistance,
    AC_Impedance,
    ACI_Cell,
    Test_Settings,
    Error,
    Finished,
    Volt_Meter,
    Waiting_for_ACS,
    Pause,
    EMPTY,
    Idle_from_MCU,
    Start,
    Runing,
    Step_Transfer,
    Resume,
    Go_Pause,
    Go_Stop,
    Go_Next_Step,
    Online_Update,
    Daq_Memory_Unsafe,
    ACR,
};

public struct AuxData
{
    public float Value;
    public float dtValue;
}

public struct CANInfo
{
    public uint nIndex;
    public double Value;
    public string Unit;
}

public struct SMBInfo
{
    public uint nIndex;
    public uint nType;
    public string Unit;
```

```
    public object Value;
}

public class ChannelInfo
{
    public enum AUX_TYPE
    {
        AuxV,
        T,
        P,
        pH,
        FR,
        Conc,
        DI,
        DO,
        EC,
        Safety,
        Humidity,
        AO,
        MAX_NUM,
    };

    public static readonly ChannelInfo Empty = new ChannelInfo();
    public uint Channel = uint.MaxValue;
    public ChannelStatus Status = ChannelStatus.Idle;
    public bool   CommFailure = true;
    public string Schedule;
    public string CANCfg;
    public string SMBCfg;
    public string Testname;
    public string ExitCondition;
    public string StepAndCycle;
    public string  Barcode;
    public uint   MasterChannel;
    public double TestTime;
    public double StepTime;
    public float  Voltage;
    public float  Current;
    public float  Power;
    public float  ChargeCapacity;
    public float  DischargeCapacity;
    public float  ChargeEnergy;
```

```
        public float  DishargeEnergy;
        public float  InternalResistance;
        public float  dvdt;
        public float  ACR;
        public float  ACI;
        public float ACIPhase;
        public List<CANInfo> CAN = null;
        public List<SMBInfo> SMB = null;
        public List<AuxData>[] Auxs = new List<AuxData>[(int)AUX_TYPE.MAX_NUM];
    }
    public List<ChannelInfo> m_Channels = new List<ChannelInfo>();
  }
```

Usage:

m_Channels is a list containing the information on all requested IV channels.

ArbinCommandGetResumeDataFeed

```
  public class ArbinCommandGetResumeDataFeed : ArbinCommand
  {
    public class ResumeDataIInfo
    {
        public static readonly ResumeDataIInfo Empty = new ResumeDataIInfo();

        public uint Channel = uint.MaxValue;

        public struct RESUME_DATA
        {
            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 TC_Counter1;
            public float TC_Counter2;
            public float TC_Counter3;
            public float TC_Counter4;
            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 uint channelCode;
        public string TestName;
        public string Schedule;
        public string Createor;
        public string Comment;
        public string StartTime;
        public RESUME_DATA ResumeData;

        public List<string> Steps = null;
    }

    public List<ResumeDataIInfo> m_Channels = new List<ResumeDataIInfo>();
}
```

Usage:

m_Channels is a list containing the resume status of every channel requested.

ArbinCommandGetSerialNumberFeed

```
public class ArbinCommandGetSerialNumberFeed : ArbinCommand
  {
    public enum ASSIGN_TOKEN
    {
      CTI_GET_SERIAL_SUCCESS,
      CTI_ASSIGN_ERROR = 0x10,
    };
    public double SerialNum;
    public ASSIGN_TOKEN Result;
    }
```

Usage:

Result contains the success message from the cycler. Refer to ArbinCommandGetSerialNumberFeed. ASSIGN_TOKEN for the meaning of the value.

ArbinCommandGetStartDataFeed

```
public class ArbinCommandGetStartDataFeed : ArbinCommand
  {
    public class StartDataIInfo
    {
      public static readonly StartDataIInfo Empty = new StartDataIInfo();
      public uint Channel = uint.MaxValue;
      public uint channelCode;
      public string Schedule;
      public float fMV_UD1;
      public float fMV_UD2;
      public float fMV_UD3;
      public float fMV_UD4;
      public float fMV_UD5;
      public float fMV_UD6;
      public float fMV_UD7;
      public float fMV_UD8;
      public float fMV_UD9;
      public float fMV_UD10;
      public float fMV_UD11;
      public float fMV_UD12;
      public float fMV_UD13;
```

```
        public float fMV_UD14;
        public float fMV_UD15;
        public float fMV_UD16;
        public List<string> TestNames = null;
        public List<string> Steps = null;
    }
    public List<StartDataIInfo> m_Channels = new List<StartDataIInfo>();
  }
```

Usage:

m_Channels is a list containing the information on all requested IV channels.

ArbinCommandJumpChannelFeed

```
  public class ArbinCommandJumpChannelFeed : ArbinCommand
  {
    public enum JUMP_TOKEN
    {
        CTI_JUMP_SUCCESS,
        CTI_JUMP_INDEX = 0x10,
        CTI_JUMP_ERROR,
        CTI_JUMP_CHANNEL_RUNNING,
        CTI_JUMP_CHANNEL_NOT_CONNECT,
        CTI_JUMP_SCHEDULE_VALID,
        CTI_JUMP_NO_SCHEDULE_ASSIGNED,
        CTI_JUMP_SCHEDULE_VERSION,
        CTI_JUMP_POWER_PROTECTED,
        CTI_JUMP_RESULTS_FILE_SIZE_LIMIT,
        CTI_JUMP_STEP_NUMBER,
        CTI_JUMP_NO_CAN_CONFIGURATON_ASSIGNED,
        CTI_JUMP_AUX_CHANNEL_MAP,
        CTI_JUMP_BUILD_AUX_COUNT,
        CTI_JUMP_POWER_CLAMP_CHECK,
        CTI_JUMP_AI,
        CTI_JUMP_SAFOR_GROUPCHAN,
        CTI_JUMP_BT6000RUNNINGGROUP,
        CTI_JUMP_CHANNEL_DOWNLOADING_SCHEDULE,
        CTI_JUMP_DATABASE_QUERY_TEST_NAME_ERROR,
        CTI_JUMP_TEXTNAME_EXITS,
        CTI_JUMP_GO_STEP,
        CTI_JUMP_INVALID_PARALLEL,
    };
    public JUMP_TOKEN Result;
```

```
    public int errorChannel;
    }
```

Usage:

Result contains the success message from the cycler. Refer to ArbinCommandJumpChannelFeed. JUMP_TOKEN for the meaning of the value.

ArbinCommandLogicConnectFeed

```
  public class ArbinCommandLogicConnectFeed : ArbinCommand
  {
    // 0: Can be kicked
    // 1: Cannot be kicked
    public uint dwSetKickOut;
    // 0: Connection Accepted
    // 1: Connection not accepted
    public uint dwConnectResult;
  }
```

Usage:

It contains information on user privileges. dwSetKickOut specifies whether the user can be kicked off the cycler (0) or not (1). dwConnectResult specifies whether or not the connection request has been accepted by the cycler. A value of 0 indicates the connection has been accepted, and 1 indicates it has not.

ArbinCommandLoginFeed

```
  public class ArbinCommandLoginFeed : ArbinCommand
  {
    public enum CTI_VERSION
    {
      CTI_PRO7_1,
    };

    public enum LOGIN_RESULT
    {
      CTI_LOGIN_SUCCESS = 1,
      CTI_LOGIN_FAILED,
      CTI_LOGIN_BEFORE_SUCCESS
    };
    public LOGIN_RESULT Result;
    public int UserType;
    public string SN;
    public string Note;
```

```
    public string NickName;
    public string Location;
    public string EmergencyContactNameAndPhoneNumber;
    public string OtherComments;
    public string Email;
    public uint ITAC;
    public string CALL;
    public uint ChannelNum;
    public CTI_VERSION Version;
    public Image Img = null;
  }
```

Usage:

Result contains the success message from the cycler. Refer to ArbinCommandLoginFeed.LOGIN_RESULT for the meaning of the value.

SN contains the serial number of the cycler.

Location contains information on the cycler's location.

EmergencyContactNameAndPhoneNumber contains information on contact information for the person in charge of emergencies regarding the cycler.

ChannelNum contains the number of channels on the system.

Img contains an image set for the cycler.

ArbinCommandNewFolderFeed

```
  public class ArbinCommandNewFolderFeed : ArbinCommand
  {
    public enum RESULT_TYPE
    {
      CTI_NEW_SUCCESS = 1,
      CTI_DELETE_SUCCESS,
      CTI_NEW_FAILED,
      CTI_NEW_FAILED_ADD_FOLDER,
      CTI_DELETE_FAILED,
      CTI_DELETE_FAILED_EXTENSION,
      CTI_DELETE_FAILED_TEXT_RUNNING,
      CTI_DELETE_FAILED_EXIST
    };

    public RESULT_TYPE Result;
  }
```

Usage:

Result contains the success message from the cycler. Refer to ArbinCommandNewFolderFeed.RESULT_TYPE for the meaning of the value.

ArbinCommandNewOrDeleteFeed

```
public class ArbinCommandNewOrDeleteFeed : ArbinCommand
{
    public enum NEW_OR_DELETE_RESULT
    {
        CTI_NEW_SUCCESS = 1,
        CTI_DELETE_SUCCESS,
        CTI_NEW_FAILED,
        CTI_NEW_FAILED_ADD_FOLDER,
        CTI_DELETE_FAILED,
        CTI_DELETE_FAILED_EXTENSION,
        CTI_DELETE_FAILED_TEXT_RUNNING,
        CTI_DELETE_FAILED_EXIST
    };

    public enum NEW_OR_DELETE_TYPE
    {
        CTI_DELETE = 0,
        CTI_NEW = 1
    };
    public NEW_OR_DELETE_RESULT Result;
}
```

Usage:

Result contains the success message from the cycler. Refer to ArbinCommandNewOrDeleteFeed.NEW_OR_DELETE_RESULT for the meaning of the value.

ArbinCommandResumChanneleFeed

```
public class ArbinCommandResumChanneleFeed : ArbinCommand
{
    public enum RESUME_TOKEN
    {
        RESUME_SUCCESS,
        RESUME_INDEX = 0x10,
        RESUME_ERROR,
        RESUME_CHANNEL_RUNNING,
        RESUME_CHANNEL_NOT_CONNECT,
        RESUME_SCHEDULE_VALID,
        RESUME_NO_SCHEDULE_ASSIGNED,
        RESUME_SCHEDULE_VERSION,
```

```
        RESUME_POWER_PROTECTED,
        RESUME_RESULTS_FILE_SIZE_LIMIT,
        RESUME_STEP_NUMBER,
        RESUME_NO_CAN_CONFIGURATON_ASSIGNED,
        RESUME_AUX_CHANNEL_MAP,
        RESUME_BUILD_AUX_COUNT,
        RESUME_POWER_CLAMP_CHECK,
        RESUME_AI,
        RESUME_SAFOR_GROUPCHAN,
        RESUME_BT6000RUNNINGGROUP,
        RESUME_CHANNEL_DOWNLOADING_SCHEDULE,
        RESUME_DATABASE_QUERY_TEST_NAME_ERROR,
        RESUME_NO_TEST_NAME,
        RESUME_LOAD_RESUME,
        RESUME_MAX_MULTIPLE_RESULT,
    },
    public RESUME_TOKEN Result;
  }
```

Usage:

Result contains the success message from the cycler. Refer to ArbinCommandResumChanneleFeed. RESUME_TOKEN for the meaning of the value.

ArbinCommandSendMsgToCTIFeed

```
public class ArbinCommandSendMsgToCTIFeed : ArbinCommand
  {
    public enum SEND_MSG_TO_CTI_RESULT
    {
      SEND_MSG_TO_CTI_SUCCESS = 1,
      SEND_MSG_TO_CTI_FAILED
    };
    public SEND_MSG_TO_CTI_RESULT Result;
    }
```

Usage:

Result contains the success message from the cycler. Refer to ArbinCommandSendMsgToCTIFeed. SEND_MSG_TO_CTI_RESULT for the meaning of the value.

ArbinCommandSetMetaVariableMVFeed

```
  public class ArbinCommandSetMetaVariableFeed: ArbinCommand
  {
    public enum SET_MV_RESULT
```

```
    {
      CTI_SET_MV_SUCCESS,
      CTI_SET_MV_FAILED=16,
      CTI_SET_MV_METACODE_NOTEXIST,
      CTI_SET_MV_CHANNEL_NOT_STARTED
    };
    public SET_MV_RESULT Result;
  }
```

Usage:

Result contains the success message from the cycler. Refer to ArbinCommandSetMVFeed.SET_MV_RESULT for the meaning of the value.

ArbinCommandUpdateMetaVariableAdvancedFeed

```
public class ArbinCommandUpdateMetaVariableAdvancedFeed: ArbinCommand
{
    public enum SET_MV_RESULT
    {
        CTI_SET_MV_SUCCESS,
        CTI_SET_MV_FAILED=16,
        CTI_SET_MV_METACODE_NOTEXIST,
        CTI_SET_MV_CHANNEL_NOT_STARTED,
        CTI_SET_MV_METACODE_UPDATE_TOO_FREQUENTLY_200MS
    };
    public SET_MV_RESULT Result;
}
```

Usage:

Result contains the success message from the cycler. Refer to ArbinCommandUpdateMVAdvancedFeed. SET_MV_RESULT for the meaning of the value.

ArbinCommandStartAutomaticCalibrationFeed

```
public class ArbinCommandStartAutomaticCalibrationFeed : ArbinCommand
{
    public enum AUTOCALI_START_RESULT
    {
        CTI_AUTOCALI_START_SUCCESS = 1,
        CTI_AUTOCALI_START_FAILED
    };

    public AUTOCALI_START_RESULT Result;
}
```

Usage:

The result contains the success message from the cycler.

ArbinCommandStartChannelFeedBack

```
public class ArbinCommandStartChannelFeed : ArbinCommand
{
    public enum START_TOKEN
    {
        CTI_START_SUCCESS,
        CTI_START_INDEX = 0x10,
        CTI_START_ERROR,
        CTI_START_CHANNEL_RUNNING,
        CTI_START_CHANNEL_NOT_CONNECT,
```

```
        CTI_START_SCHEDULE_VALID,
        CTI_START_NO_SCHEDULE_ASSIGNED,
        CTI_START_SCHEDULE_VERSION,
        CTI_START_POWER_PROTECTED,
        CTI_START_RESULTS_FILE_SIZE_LIMIT,
        CTI_START_STEP_NUMBER,
        CTI_START_NO_CAN_CONFIGURATON_ASSIGNED,
        CTI_START_AUX_CHANNEL_MAP,
        CTI_START_BUILD_AUX_COUNT,
        CTI_START_POWER_CLAMP_CHECK,
        CTI_START_AI,
        CTI_START_SAFOR_GROUPCHAN,
        CTI_START_BT6000RUNNINGGROUP,
        CTI_START_CHANNEL_DOWNLOADING_SCHEDULE,
        CTI_START_DATABASE_QUERY_TEST_NAME_ERROR,
        CTI_START_TEXTNAME_EXITS,
        CTI_START_GO_STEP,
        CTI_START_INVALID_PARALLEL,
    };

    public START_TOKEN Result;
  }
```

Usage:

The result contains the success message from the cycler. Refer to ArbinCommandStartChannelFeed.START_TOKEN for the meaning of the value.

ArbinCommandContinueChannelFeedBack

```
  public class ArbinCommandContinueChannelFeed : ArbinCommand
  {
    public enum CONTINUE_TOKEN
    {
        CTI_CONTINUE_SUCCESS,
        CTI_CONTINUE_INDEX = 0x10,
        CTI_CONTINUE_ERROR,
        CTI_CONTINUE_CHANNEL_RUNNING,
        CTI_CONTINUE_CHANNEL_NOT_CONNECT,
        CTI_CONTINUE_CHANNEL_CALIBRATING,
        CTI_CONTINUE_NOT_PAUSE_NORMAL,
        CTI_CONTINUE_CHANNEL_UNSAFE
    };
```

```
    public CONTINUE_TOKEN Result;
  }
```

Usage:

The result contains the success message from the cycler. Refer to ArbinCommandContinueChannelFeed.CONTINUE_TOKEN for the meaning of the value.

ArbinCommandStopChannelFeed

```
  public class ArbinCommandStopChannelFeed : ArbinCommand
  {
    public enum STOP_TOKEN
    {
      SUCCESS = 0,
      STOP_INDEX = 0x10,
      STOP_ERROR,
      STOP_NOT_RUNNING,
      STOP_CHANNEL_NOT_CONNECT
    };
    public STOP_TOKEN Result;
  }
```

Usage:

The result contained information success or failure information on issuing the PostStopMsg command. See ArbinCommandStopChannelFeed.STOP_TOKEN for the different result types.

ArbinCommandUpLoadFileFeed

```
  public class ArbinCommandUpLoadFileFeed : ArbinCommand
  {
    public enum UPLOAD_RESULT
    {
      CTI_UPLOAD_SUCCESS = 1,
      CTI_UPLOAD_FAILED,
      CTI_UPLOAD_MD5_ERR,
      CTI_UPLOAD_FAILED_TEXT_RUNNING
    };
    public UPLOAD_RESULT Result;
    public double UploadTime;
    public uint uGeneralPackage;
    public uint uPackageIndex;
  }
```

Usage:

Result contains the success message from the cycler. Refer to ArbinCommandUpLoadFileFeed.UPLOAD_RESULT for the meaning of the value.