



Programmer's Guide: Using the M3 ORANGE

Version: 1.0.0.

Release Date: February 24, 2011

Table of Contents

Copyright and Agreement.....	4
Development Tools and Requirements	4
Release History	4
1.0 Introduction	5
2.0 Tutorial.....	6
2.1 Bluetooth	7
2.1.1 Initialization	7
2.1.2 Create Connection	8
2.1.3 Connection	9
2.1.4 Find Connection.....	9
2.1.5 Find Device.....	11
2.1.6 Find Service.....	12
2.1.7 Delete Connection	15
2.1.8 Disconnect.....	16
2.1.9 Close	16
2.2 Camera.....	18
2.2.1 Open	18
2.2.2 Close	18
2.2.3 Capture Still Shot.....	19
2.2.4 Capture Video.....	19
2.2.5 Preview	20
2.3 GPS.....	21
2.3.1 Open GPS	21
2.3.2 Close GPS.....	21
2.3.3 Receive Message from GPS Module.....	22
2.4 RFID	23
2.4.1 Open	23
2.4.2 Close	24
2.4.3 Antenna on/off.....	25
2.4.4 Data Read	26
2.4.5 Data Write	27
2.5 Scanner 1D	28
2.5.1 Initialization	28
2.5.2 Default Settings.....	30
2.5.3 ScanRead and GetDecodeData	32
2.5.4 Close Scanner	35
2.6 Scanner 2D (Imager)	37
2.6.1 Initialization	37

2.6.2	Default Settings.....	38
2.6.3	ScanRead and GetDecodeData	38
2.6.4	Close Scanner	42
3.0	Samples.....	43
3.1	Bluetooth	43
3.2	Camera.....	43
3.3	GPS	43
3.4	RFID	43
3.5	Scanner 1D (Software Decoder only).....	43
3.6	Scanner 2D (Imager)	43
4.0	References (Function Lists)	44
4.1	Bluetooth	45
4.1.1	Reference and Function List for C++	46
4.1.2	Reference and Function List for C#	85
4.1.3	BTExplorer API Error Codes.....	95
4.2	Camera.....	96
4.2.1	Reference and Function List for C++	97
4.2.2	Reference and Function List for C#	107
4.3	GPS	116
4.3.1	Reference and Function List for C++	117
4.3.2	Reference and Function List for C#	122
4.4	RFID	126
4.4.1	Reference and Function List for C++.....	127
4.4.2	Command List for C++	133
4.4.3	Reference and Function List for C#.....	139
4.4.4	Command List for C#	146
4.5	Scanner 1D	153
4.5.1	Reference and Function List for C++	154
4.5.2	Reference and Function List for C#	166
4.6	Scanner 2D (Imager)	190
4.6.1	Reference and Function List for C++	191
4.6.2	Reference and Function List for C#	202

Copyright and Agreement

WARNING: All contents of this SDK manual are protected by the copyright laws and all rights are reserved. Unauthorized distribution or copying is strictly prohibited.

M3 Mobile does not guarantee the quality and performance of the programs written in unsupported programming language. For supported development tools and languages, please refer to Development Tool and Requirements section.

Development Tools and Requirements

Supported Development Tools and Languages

- Visual Studio 2005 (8.0) (.NET Framework 2.0) – Visual C++, Visual C#

Development System Requirements

- Pentium – compatible 500MHz processor or better
- Microsoft Windows 98 / ME / 2000 / XP / 2003
- ActiveSync

Development Platform Requirements

- M3 ORANGE Platform SDK must be installed on your computer to be able to develop softwares using this SDK.
 - Click [HERE](#) to download M3 ORANGE Platform for Windows Mobile 6
 - Click [HERE](#) to download M3 ORANGE Platform for Windows Mobile 5.0.

Release History

M3 ORANGE SDK Version 1.0.0

- First release of the M3 ORANGE SDK manual

1.0 Introduction

This document is a reference guide for the software developer's kit (SDK) for M3 ORANGE (**WM 6.5**). This handheld terminal may include up to 6 different modules depending on the specification of the device. For module list and brief description of each module, see below table.

Module Name	Description
Bluetooth	Mainly used to connect to a Bluetooth head set for phone calls or printer. It uses COM9 for BT connections.
Camera	Used to take a picture of an object.
GPS	Gathers information on current location through satellite. GPS can be used as an extension to the compatible vehicle cradle. COM0 is used for vehicle cradle serial.
RFID	Read data from tags or write to the tags through MOC1.
Scanner	Read 1D and / or 2D barcodes depend on the scanner module option. COM6 is reserved for scanner.
WLAN	Enable network connection using Wi-Fi. Summit WLAN (MSD10AG) module is used. (WLAN SDK is NOT included in this document)

This guide document provides comprehensive SDK manual by providing Tutorials, Samples and References including function lists.

2.0 Tutorial

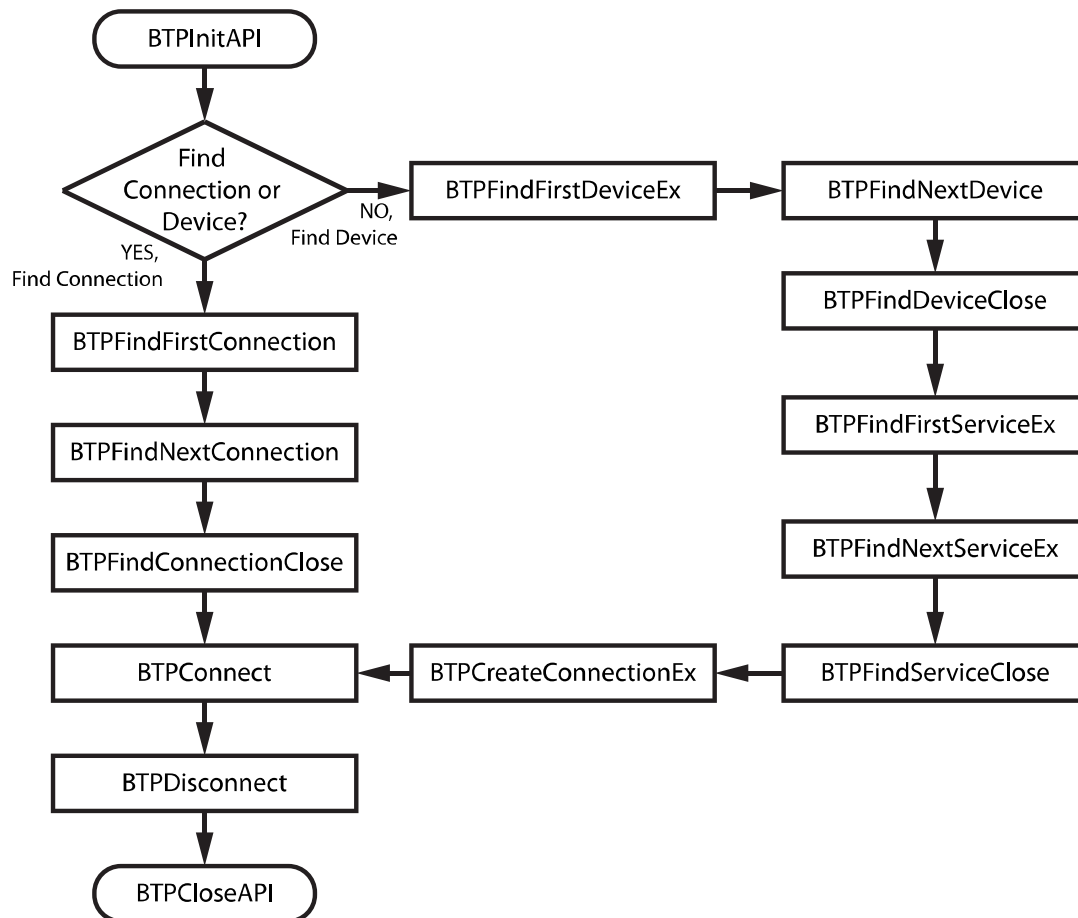
This chapter describes the basic usage of M3 Mobile SDK functions in a step-by-step manner. In this tutorial section, the following topics are treated.

Section	Topic
Bluetooth	Initialization Create Connection Connection Find Connection Find Device Find Service Delete Connection Disconnect Close
Camera	Open Close Capture Still Shot Capture Video Preview
GPS	Open GPS Close GPS Receive Message from GPS Module
RFID	Open Close Antenna on/off Data Read Data Write
Scanner 1D	Initialization Default Settings ScanRead and GetDecodeData Close Scanner
Scanner 2D	Initialization Default Settings ScanRead and GetDecodeData Close Scanner

2.1 Bluetooth

In general, Bluetooth module is included in the PDA as a default option.

Please refer to below flow chart for Bluetooth.



Bluetooth flow chart

2.1.1 Initialization

Initialization is the first step to use Bluetooth.

C++

```
#include "CustomerDLL.h"

if(BTPInitAPI())
{
    // Init success
}
else
{
    // Init Fail
}
```

C#

```
using BTEAPIdllNet;
```

```

M3BTEDllNet BteDll;

BteDll = new M3BTEDllNet();

if(BteDll.M3BTPInit())
{
    // Init success
}
else
{
    // Init Fail
}

```

2.1.2 Create Connection

The connection between the device founded by M3 Bluetooth module and the service to use can be configured as a favorite connection. The **BTPCreateConnectionEx** function can be used in BT pairing via COM9 by assigning the LocalComPort to **BTP_Connection_Info**.

C++

```

#include "CustomerDLL.h"

// Create Connection

BTP_Connection_Info_Ex_t connectionInfo;
HRESULT result;
/* Create a favorite association */
connectionInfo.ConnectionAttributes = BTP_CONNECTION_REMEMBERED |
BTP_CONNECTION_ACTIVE;
connectionInfo.LocalCOMPort = 9;
connectionInfo.ProfileType = BTP_PROFILE_SPP;
result = BTPCreateConnectionEx(&connectionInfo);

if (BTP_ERROR_SUCCESS == result)
    SetWindowText(L"create connection");
else
    AfxMessageBox(L"Create Connection_error");

```

C#

```

using BTEAPIdllNet;

M3BTEDllNet BteDll;

BteDll = new M3BTEDllNet();

uint nConnectionNum = 0; // First Connection

M3BTEDllNet._CONNECTINFO connectinfo;
byte[] Name_s = new byte[1026];
connectinfo.BD_ADDR = m_localinfo.BD_ADDR;

BteDll.M3BTPGetFindService(nConnectionNum, out connectinfo, Name_s);

SString szstr; szstr = String.Format("FIND service {0:x2} {1:x2} {2:x2} {3:x2} {4:x2} {5:x2} {6:G}", connectinfo.BD_ADDR.add5, connectinfo.BD_ADDR.add4, connectinfo.BD_ADDR.add3, connectinfo.BD_ADDR.add2, connectinfo.BD_ADDR.add1, connectinfo.BD_ADDR.add0, Encoding.Default.GetString(Name_s, 0, 1026));

MessageBox.Show(szstr);

if (BteDll.M3BTPCreateConnection(out connectinfo))
    MessageBox.Show("create success");

```



```
else
    MessageBox.Show("create fail");
```

2.1.3 Connection

Make a desired connection.

C++

```
#include "CustomerDLL.h"

// Delete Connection

BTP_Connection_Info_Ex_t connectionInfo;

HRESULT result;

result=BTPConnect(connectionInfo.ConnectionID);

if(result==BTP_ERROR_SUCCESS)
    SetWindowText(L"Connect");
else if(result ==BTP_ERROR_INVALID_PARAMETER)
    AfxMessageBox(L"ERROR_INVALID_PARAMETER");
else if(result ==BTP_ERROR)    AfxMessageBox(L"error");
```

C#

```
using BTEAPIdllNet;

M3BTEDllNet BteDll;

BteDll = new M3BTEDllNet();

uint nConnectionNum = 0;

M3BTEDllNet._CONNECTINFO connectinfo;
BBteDll.M3BTPGetFindConnection(nConnectionNum , out connectinfo);
BteDll.M3BTPConnect(connectinfo.ConnectionID)
```

2.1.4 Find Connection

When connection is created, the connection information is saved in the device. Then, by finding the connection, user can easily connect to the service of the device. This process is starting with **BTPFindFirstConnection**. If the user is not trying to establish a connection to the saved service, another connection can be searched with **BTPFindNextConnection**. To end searching BT devices, **BTPFindConnectionClose** is used.

C++

```
#include "CustomerDLL.h"

// Find Connection

m_ctrllist.DeleteAllItems();
m_Connectitem.RemoveAll();

m_listtype = 3;           //list type
HRESULT result;
CString stradd;
BTP_Connection_Find      connectFind;
BTP_Connection_Info_t    connectioinfo;
BTP_Connection_Query_t    connectQuery;

memset(&connectQuery, 0, sizeof(connectQuery));
```

```

connectQuery.ConnectionAttributes = BTP_CONNECTION_REMEMBERED;

BeginWaitCursor();

result = BTPFindFirstConnection(&connectFind, &connectioinfo, &connectQuery);
if (BTP_ERROR_SUCCESS == result)
{
    EndWaitCursor();
    do
    {
        BTP_Connection_Info_t connectInfo_temp;
        memcpy(&connectInfo_temp, &connectioinfo, sizeof(BTP_Connection_Info_t));
        m_Connectitem.AddTail(connectInfo_temp);

        result = BTPFindNextConnection(connectFind, &connectioinfo);
    } while (BTP_ERROR_SUCCESS == result);
}
BTPFindConnectionClose(connectFind);

int lm_count = m_Connectitem.GetCount();
int i = 0;

for(i = lm_count-1; i>=0; i--)
{
    connectioinfo = m_Connectitem.GetAt(m_Connectitem.FindIndex(i));
    stradd.Format(L "[%02X:%02X:%02X:%02X:%02X:%02X] id:%d",
        connectioinfo.BD_ADDR.BD_ADDR5,
        connectioinfo.BD_ADDR.BD_ADDR4,
        connectioinfo.BD_ADDR.BD_ADDR3,
        connectioinfo.BD_ADDR.BD_ADDR2,
        connectioinfo.BD_ADDR.BD_ADDR1,
        connectioinfo.BD_ADDR.BD_ADDR0,
        connectioinfo.ConnectionID);

    m_ctrllist.InsertItem(0, stradd, 0);
}

```

C#

```

using BTEAPIdllNet;

M3BTEDllNet BteDll;

BteDll = new M3BTEDllNet();

int connectioncount = BteDll.M3BTPFindConnection();

for (uint i = 1; i <= connectioncount ; i++)
{
    M3BTEDllNet._CONNECTINFO connectinfo;
    String szstr;

    BteDll.M3BTPGetFindConnection(i, out connectinfo);
    szstr = String.Format("{0:x2} {1:x2} {2:x2} {3:x2} {4:x2} {5:x2}",
        connectinfo.BD_ADDR.add5,
        connectinfo.BD_ADDR.add4,
        connectinfo.BD_ADDR.add3,
        connectinfo.BD_ADDR.add2,
        connectinfo.BD_ADDR.add1,
        connectinfo.BD_ADDR.add0);

    MessageBox.Show("szstr");
}

```

2.1.5 Find Device

Below code is an example of searching BT devices around M3 PDA and shows a list of found device in ListControl. First, find BT device with **BTPFindFistDeviceEx**. If the found device is not what you wish to connect, then **BTPFindNextDevice** is used to find the next BT device. If the device you wish to connect is found or even if the device is not found, user can terminate searching by **BTPFindDeviceClose**.

C++

```
#include "CustomerDLL.h"

// Device Find

m_listtype = 1;           //list type
m_ctrllist.DeleteAllItems();
m_Deviceitem.RemoveAll();
memset(&connectionInfo, 0, sizeof(connectionInfo));
connectionInfo.Size = sizeof(BTP_Connection_Info_Ex_t);
connectionInfo.ProfileType = BTP_PROFILE_SPP;
CString str,stradd,strname;

HRESULT result;
BTP_Device_Find          deviceFind;
BTP_Device_Info_t        deviceInfo;
BTP_Device_Query_Ex_t    deviceQuery;
memset(&deviceQuery, 0, sizeof(deviceQuery));
deviceQuery.Size = sizeof(BTP_Device_Query_Ex_t);

/* Looking for all devices either discovered or remembered */
deviceQuery.DeviceAttributes = BTP_DEVICE_ALL;
deviceQuery.DeviceType = (connectionInfo.ProfileType == BTP_PROFILE_SPP ? bdAll :
bdHID);

/* Tells the API to perform a ten second GAP inquiry, using zero (0) */

/* here will just return previously discovered devices. */
deviceQuery.InquiryTimeout = 10;

BeginWaitCursor();

result = BTPFindFirstDeviceEx(&deviceFind, &deviceInfo, &deviceQuery);

if (BTP_ERROR_SUCCESS == result)
{
    EndWaitCursor();
    do
    {
        BTP_Device_Info_t deviceInfo_temp;
        memcpy(&deviceInfo_temp,&deviceInfo,sizeof(BTP_Device_Info_t));

        m_Deviceitem.AddTail(deviceInfo_temp);
        result = BTPFindNextDevice(deviceFind, &deviceInfo);
    } while (BTP_ERROR_SUCCESS == result);
}

BTPFindDeviceClose(deviceFind);

int lm_count = m_Deviceitem.GetCount();
int i = 0;

for(i = lm_count-1;i>=0;i--)
{
    deviceInfo = m_Deviceitem.GetAt(m_Deviceitem.FindIndex(i));
    stradd.Format(L"%02X:%02X:%02X:%02X:%02X:%02X]Device: %d %s ",
```

```

        deviceInfo.BD_ADDR.BD_ADDR5,
        deviceInfo.BD_ADDR.BD_ADDR4,
        deviceInfo.BD_ADDR.BD_ADDR3,
        deviceInfo.BD_ADDR.BD_ADDR2,
        deviceInfo.BD_ADDR.BD_ADDR1,
        deviceInfo.BD_ADDR.BD_ADDR0,
        deviceInfo.DeviceAttributes,
        CString(deviceInfo.Name));
}

```

C#

```

using BTEAPIdllNet;

M3BTEDllNet BteDll;

BteDll = new M3BTEDllNet();

int devicecount = BteDll.M3BTPFindDevice();

String szstr;

ffor (uint i = 1; i <= devicecount ; i++)
{
    M3BTEDllNet._LOCALINFO localinfo;
    byte[] Name = new byte[1026];

    BteDll.M3BTPGetFindDevice(i, out localinfo, Name);
    szstr = String.Format("{0:x2} {1:x2} {2:x2} {3:x2} {4:x2} {5:x2} {6:G}",
        localinfo.BD_ADDR.add5,
        localinfo.BD_ADDR.add4,
        localinfo.BD_ADDR.add3,
        localinfo.BD_ADDR.add2,
        localinfo.BD_ADDR.add1,
        localinfo.BD_ADDR.add0,
        Encoding.Default.GetString(Name, 0, 1026));
    listBox1.Items.Add(szstr);
}

```

2.1.6 Find Service

BT enabled devices normally provide one or more services that can be used through BT. In such devices, the service provided by the device must be identified. **BTPFindFirstServiceEx** searches the first service available in the found device. If the device provides more than one service, **BTPFindNextServiceEx** can be used to search the next service. **BTPFindServiceClose** is used to end finding service.

BT can provide following services;

- AVC — Audio/Video Control Transport Protocol Sample Application
- AVR — Audio/Video Remove Control Profile Sample Application
- BTC — Generic Bluetooth COM Profile Sample Application
- DUN — Dial-Up Networking Profile Sample Application
- FTP — OBEX File Transfer Profile Sample Application
- GAV — Generic Audio/Video Profile Sample Application
- HDS — Headset Profile Sample Application
- OBP — OBEX Object Push Profile Sample Application
- PAN — Personal Area Networking Profile Sample Application
- SDP — Sample SDP Application using Bluetopia
- SPP — Sample SPP Application using Bluetopia

C++

```

#include "CustomerDLL.h"

// Service Find

m_ctrlList.DeleteAllItems();

unsigned                serviceUUID = 16;
HRESULT result;
BTP_Service_Find       serviceFind;
BTP_Service_Info_Ex_t  serviceInfo;
BTP_Service_Query_t    serviceQuery;
SDP_UUID_Entry_t       service[1];
CString                strName;

serviceInfo.Size = sizeof(BTP_Service_Info_Ex_t);

memset(&serviceQuery, 0, sizeof(serviceQuery));

/* Set the BD ADDR of the device that we want to query */
serviceQuery.BD_ADDR = connectionInfo.BD_ADDR;
serviceQuery.NumberServiceUUID = 1;
serviceQuery.Service = service;

/* Normally, the service UUID will be a 16-bit UUID. There is only one */
/* custom case that we have encountered where a 128-bit UUID has been */
/* used for a custom SPP service. For this reason, we will only check */
/* for the special 128-bit case and assume all others are a 16-bit UUID. */
service[0].SDP_Data_Element_Type = deUUID_16;
switch(connectionInfo.ProfileType)
{
    case BTP_PROFILE_SPP:
        if (128==serviceUUID)
        {
            service[0].SDP_Data_Element_Type = deUUID_128;
            /* This is the special case of a custom 128-bit UUID SPP. */
            ASSIGN_UUID_128((service[0].UUID_Value.UUID_128),
                0x85, 0x60, 0xCA, 0x18,    // 0x8560CA18
                0x62, 0x3F,                // 0x623f
                0x4A, 0x77,                // 0x4a77
                0x9F, 0x5E, 0x3C, 0x52, 0x66, 0x68, 0x05, 0x1A);
                // 0x9f, 0x5e, 0x3c, 0x52, 0x66, 0x68, 0x05, 0x1a
        }
        else
        {
            /* We will only request services that support SPP. */
            /* FYI: The 16-bit UUID for SPP is 0x1101. Reference the Bluetooth */
            /* spec for additional UUIDs at http://www.bluetooth.org */
            ASSIGN_UUID_16((service[0].UUID_Value.UUID_16), 0x11, 0x01)
        }
        break;
    case BTP_PROFILE_HID_HOST:
    case BTP_PROFILE_HID_DEVICE:
        /* We will only request services that support HID. */
        /* FYI: The 16-bit UUID for HID is 0x1124. Reference the Bluetooth */
        /* spec for additional UUIDs at http://www.bluetooth.org */
        ASSIGN_UUID_16((service[0].UUID_Value.UUID_16), 0x11, 0x24)
        break;
}

BeginWaitCursor();
/* Search for the service */
result = BTPFindFirstServiceEx(&serviceFind, &serviceInfo, &serviceQuery);
if (BTP_ERROR_SUCCESS == result)
{

```

```

EndWaitCursor();
do
{
    /* Is this the service to which we want to connect? For this ex- */
    /* ample, we will limit our search to any service that at least */
    /* begins with the string passed in as the service name. */

    /* We found a service that is close to what we are seeking */
    /* Populate the Connection Info structure for this service. */
    connectionInfo.ProfileType = serviceInfo.ProfileType;
    connectionInfo.MajorVersion = serviceInfo.MajorVersion;
    connectionInfo.MinorVersion = serviceInfo.MinorVersion;
    switch(connectionInfo.ProfileType)
    {
        case BTP_PROFILE_UNKNOWN:
            /* Treat unknown profiles as RFCOMM based (SPP) profiles */
            connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.RFCOMMServerPort =
            serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.RFCOMMServerPort;

            connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.UseActiveSync =
            serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.UseActiveSync;
            break;
            case BTP_PROFILE_SPP:
                connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.RFCOMMServerPort =
                serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.RFCOMMServerPort;

                connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.UseActiveSync =
                serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.UseActiveSync;

                strname.Format(L"%s\n", CString(serviceInfo.ServiceName));

                m_ctrllist.InsertItem(0, strname, 0);
                break;
                case BTP_PROFILE_HID_HOST:
                case BTP_PROFILE_HID_DEVICE:
                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.DeviceAutomaticReconnect =
                    serviceInfo.ProfileInformation.RemoteHIDProfileInfo.DeviceAutomaticReconnect;

                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.DeviceNormallyConnectable =
                    serviceInfo.ProfileInformation.RemoteHIDProfileInfo.DeviceNormallyConnectable;

                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.DeviceSubclass =
                    serviceInfo.ProfileInformation.RemoteHIDProfileInfo.DeviceSubclass;

                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.L2CAPControlChannel =
                    serviceInfo.ProfileInformation.RemoteHIDProfileInfo.L2CAPControlChannel;

                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.L2CAPInterruptChannel =
                    serviceInfo.ProfileInformation.RemoteHIDProfileInfo.L2CAPInterruptChannel;

                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.VirtualCableSupported =
                    serviceInfo.ProfileInformation.RemoteHIDProfileInfo.VirtualCableSupported;
                    break;
            }

            result = BTPFindNextServiceEx(serviceFind, &serviceInfo);
        } while (BTP_ERROR_SUCCESS == result);

        if (BTP_ERROR_NO_MORE != result)
            fprintf(stderr, "%s - ERROR (line %d): Failed to find next service.
Result = %d\n", __FUNCTION__, __LINE__, result);

        BTPFindServiceClose(serviceFind);
    }
}

```

C#

```
using BTEAPIdllNet;

M3BTEDllNet BteDll;

BteDll = new M3BTEDllNet();

String szstr;
byte[] Name_d = new byte[1026];
uint nDeviceNum = 1;

BteDll.M3BTPGetFindDevice(nDeviceNum , out m_localinfo, Name_d);

szstr = String.Format("FIND service {0:x2} {1:x2} {2:x2} {3:x2} {4:x2} {5:x2} {6:G}",

    m_localinfo.BD_ADDR.add5,
    m_localinfo.BD_ADDR.add4,
    m_localinfo.BD_ADDR.add3,
    m_localinfo.BD_ADDR.add2,
    m_localinfo.BD_ADDR.add1,
    m_localinfo.BD_ADDR.add0,
    Encoding.Default.GetString(Name_d, 0, 1026));

uint nservicecount = BteDll.M3BTPFindService(out m_localinfo);

for (uint i = 1; i <= nservicecount ; i++)
{
    M3BTEDllNet._CONNECTINFO connectinfo;
    byte[] Name_s = new byte[1026];

    BteDll.M3BTPGetFindService(i, out connectinfo, Name_s);
    szstr = String.Format("{0:G}", Encoding.Default.GetString(Name_s, 0, 1026));

    MessageBox.Shwo(szstr);
}
```

2.1.7 Delete Connection

This example shows deleting created connection.
Deleted Connection cannot be found by the **FindConnection** function.

C++

```
#include "CustomerDLL.h"

// Delete Connection

BTP_Connection_Info_Ext connectionInfo;

HRESULT result;
/* Delete the favorite */
result = BTPDeleteConnection(connectionInfo.ConnectionID);
if (BTP_ERROR_SUCCESS != result)
    SetWindowText(L"delete connection");
else
    AfxMessageBox(L"delete Connection_error");
```

C#

```
using BTEAPIdllNet;

M3BTEDllNet BteDll;
```

```

BteDll = new M3BTEDllNet();

uint nConnectionNum = 0; // First Connection

M3BTEDllNet._CONNECTINFO connectinfo;
BteDll.M3BTPGetFindConnection(nConnectionNum , out connectinfo);

if (BteDll.M3BTPDeleteConnection(connectinfo.ConnectionID))
    MessageBox.Show("delete success");
else
    MessageBox.Show("delete fail");

```

2.1.8 Disconnect

Disconnect the connected Connection.

C++

```

#include "CustomerDLL.h"
// Delete Connection

BTP_Connection_Info_Ex_t connectionInfo;

HRESULT result;

result = BTPDisconnect(connectionInfo.ConnectionID);

if(result==BTP_ERROR_SUCCESS)
    SetWindowText(L"Disconnect");
else if(result ==BTP_ERROR_INVALID_PARAMETER)
    AfxMessageBox(L"ERROR_INVALID_PARAMETER");
else if(result ==BTP_ERROR)
    AfxMessageBox(L"error");

```

C#

```

using BTEAPIdllNet;

M3BTEDllNet BteDll;

BteDll = new M3BTEDllNet();

uint nConnectionNum = 0;

M3BTEDllNet._CONNECTINFO connectinfo;
BBteDll.M3BTPGetFindConnection(nConnectionNum , out connectinfo);

BteDll.M3BTPDisconnect(connectinfo.ConnectionID);

```

2.1.9 Close

Bluetooth Module must be closed after use by the BTPCloseAPI function.

C++

```

#include "CustomerDLL.h"

BTPCloseAPI();

```

C#

```

using BTEAPIdllNet;

```

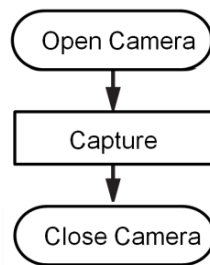


```
M3BTEDllNet BteDll;  
  
BteDll = new M3BTEDllNet();  
  
if (BteDll.M3BTPClose())  
    MessageBox.Show("close success");  
else  
    MessageBox.Show("fail");
```

2.2 Camera

This SDK is applicable to 3.0 mega pixel camera.

Please refer to below flow chart for Camera.



Simple Camera flow chart

2.2.1 Open

The module must be opened to initialize the Camera. Once the module is opened, preview screen will appear on registered window. Also, the OpenCamera function can change the Camera Mode—Still shot mode and Video mode. If the Video mode is selected, file type to be compressed must be assigned.

C++

```
#include "M3SkyCamera.h"

m_cameramode = STILL_MODE;    // still Mode
m_videotype = VIDEO_ASF;      // video File Type
OpenCamera(this->m_hWnd,m_cameramode,m_videotype);
```

C#

```
private Camera m_Cam;

private CAMERA_MODE m_cameramode;
private VIDEO_TYPE m_videotype;

m_Cam = new Camera();

m_cameramode = CAMERA_MODE.STILL_MODE;
m_videotype = VIDEO_TYPE.VIDEO_ASF;

m_Cam.Open(this.Handle, m_cameramode, m_videotype);
```

2.2.2 Close

CloseCamera in C++ and **Camera.Close** in C# close the camera module.

C++

```
#include "M3SkyCamera.h"

CloseCamera();
```

C#

```
Camera.Close();
```

2.2.3 Capture Still Shot

Captured picture can be named as user-defined, otherwise file name is assigned automatically according to the system date.

C++

```
#include "M3SkyCamera.h"

TCHAR name[1024] = {0x00};

if (bSaveDateMode)
{
    TCHAR      *Name;
    TCHAR      *reName;
    Name = NULL;
    rename = CaptureStill(Name);
}

else
{
    wcscpy(name, L"1.bmp");
    CaptureStill(name);
}
```

C#

```
private Camera m_Cam;

m_Cam = new Camera();

if (m_bStillDateSaveMode)
{
    StringBuilder path = new StringBuilder(100);
    m_Cam.Capture(null, path);
}

else
{
    filefullname = m_strStillFolder + m_strStillFileName;
    m_Cam.Capture(filefullname);
}
```

2.2.4 Capture Video

VideoStart takes file name as a parameter to save the video taken. If 0 is input to the function, current date is used for the file name. **VideoStop** can be used when finished.

C++

```
#include "M3SkyCamera.h"

TCHAR name[50] = {0x00};

if (name[0] == 0x00)
{
    VideoStart(0);    // file full name : \My Documents\My Pictures\\(DATE).wmv
}

else
{
    VideoStart(name);
}
```

C#

```
private Camera m_Cam;

m_Cam = new Camera();

if (m_bVideoDateSaveMode)
{
    m_Cam.Video_Start(null);    // file full name : \My Documents\My
    Pictures\\(DATE).wmv
}

else
{
    filefullname = m_strVideoFolder + m_strVideoFileName;
    m_Cam.Video_Start(filefullname);
}
```

2.2.5 Preview

PreviewControl controls the preview image that is currently seen by the camera.

C++

```
#include "M3SkyCamera.h"

void PreviewControl(BOOL bStart)
{
    if(bStart == TRUE)
    {
        PreviewStart();
    }

    else
    {
        PreviewStop();
    }
}
```

C#

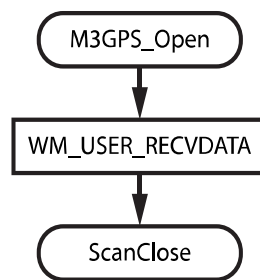
```
private Camera m_Cam;
m_Cam = new Camera();
void PreviewControl(bool bStart)
{
    if(bStart == true)
    {
        m_Cam.Preview_Start();
    }

    else
    {
        m_Cam.Preview_Stop();
    }
}
```

2.3 GPS

In M3 ORANGE, GPS can be used via COM2.

Please refer to below flow diagram for GPS.



GPS flow chart

2.3.1 Open GPS

GPS uses serial communication. To open GPS, input the COM port number, Baud Rate for serial communication and Windows HWND for getting GPS data.

C++

```
void OpenGps(TCHAR* tzCom, int nBaudRate)
{
    M3GPS_Open(m_hWnd, tzCom, nBaudRate);
}
```

C#

```
Class_Gps_Parse cGps;
cGps = new Class_Gps_Parse();

void OpenGps(String strCom, int nBaudRate)
{
    if (!cGps.Gps_Open(MsgWin.Hwnd, strCom, nBaudRate))
    {
        MessageBox.Show("Open Fail");
    }
}
```

2.3.2 Close GPS

GPS can be closed through simply closing the opened Serial Port.

C++

```
void CloseGps()
{
    M3GPS_Close();
}
```

C#

```
void CloseGps()
{
    cGps.Gps_Close();
}
```

2.3.3 Receive Message from GPS Module

When GPS Module downloads data from satellite, the data received when opening the GPS is sent to HWND. User can get information through getting WM_USER_RECVDATA message and transferring parameter to structure.

C++

```
#define WM_USER_RECVDATA    (WM_USER+10000)

// Receive WM_USER_RECVDATA
long OnRecGpsData(WPARAM wParam, LPARAM lParam)
{
    GPSParseInfo *pInf = (GPSParseInfo*)wParam;

    // GPS Information
    ParseGPSMsg(pInf);
    //

    return 0;
}
```

C#

```
long OnRecvGpsData(IntPtr wParam, IntPtr lParam)
{
    lass_Gps_Parse.GPS_PARSE_INFO info = new Class_Gps_Parse.GPS_PARSE_INFO();

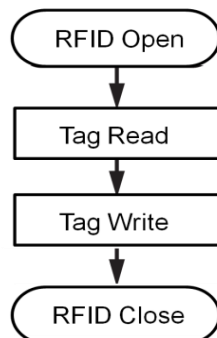
    try
    {
        info = (Class_Gps_Parse.GPS_PARSE_INFO)Marshal.PtrToStructure(wParam,
            typeof(Class_Gps_Parse.GPS_PARSE_INFO));

        GpsInfoParse(info);
    }
    catch(Exception e)
    {
        MessageBox.Show(e.Message);
        cGps.Gps_Close();
        this.Close();
        return 0;
    }
    return 0;;
}
```

2.4 RFID

In M3 ORANGE, the supported tag types are:

ISO 15693
etc...



Simple RFID flow chart

2.4.1 Open

For initializing RFID, supply power to the module through KernelIOControl. Note that RFID comport in M3 ORANGE series device is 'MOC1'. On successful power supply, open the reader through **RDR_OpenReader** function. Then, configure the reader as necessary.

C++

```
#include "CFReaderLib.h"

#define CPLD_ID_RFID_ON          0x24
#define CPLD_ID_RFID_RST        0x25

UINT nCPLDID = 0, uOut;
BOOL rc;
char com_port[10];
char buffer[514];

uOut = 1;

nCPLDID = CPLD_ID_RFID_RST;
rc = KernelIoControl(IOCTL_HAL_CPLD_CTRL_WRITE, &nCPLDID, sizeof(UINT), &uOut,
sizeof(UINT), NULL);
Sleep(100);

nCPLDID = CPLD_ID_RFID_ON;
rc = KernelIoControl(IOCTL_HAL_CPLD_CTRL_WRITE, &nCPLDID, sizeof(UINT), &uOut,
sizeof(UINT), NULL);
Sleep(50);

sprintf(com_port, "MOC%s", "1"); // M3Sky Rfid port
if(RDR_OpenComm(com_port, 1, NULL) == 0)
{
    AfxMessageBox(_T("OpenComm Failed"));
    return;
}

if(RDR_OpenReader(1, 0) == 0)
{
    AfxMessageBox(_T("OpenReader Failed"));
    return;
}
```

C#

```
using CFReaderDLLWrapper;

[DllImport("Coredll.dll")]
public static extern int KernelIoControl(uint dwIoControlCode, ref uint lpInBuf,
int nInBufSize, out uint lpOutBuf, int nOutBufSize, ref uint lpBytesReturned);

private const uint FILE_DEVICE_HAL = 0x00000101;
private const uint CPLD_ID_RFID_ON = 0x24;
private const uint CPLD_ID_RFID_RST = 0x25;
private const uint METHOD_BUFFERED = 0;
private const uint FILE_ANY_ACCESS = 0;

public ACG_CFReader dllACG;
dllACG = new ACG_CFReader();

public static uint CTL_CODE(uint DeviceType, uint Function, uint Method, uint
Access)
{
    return ((DeviceType << 16) | (Access << 14) | (Function << 2) | Method);
}

uint csel;
uint cout;

csel = CPLD_ID_RFID_ON;
cout = 1;
uint bytesReturned = 0;

uint IOCTL_HAL_CPLD_CTRL_WRITE = CTL_CODE(FILE_DEVICE_HAL, 2078, METHOD_BUFFERED,
FILE_ANY_ACCESS);

KernelIoControl(IOCTL_HAL_CPLD_CTRL_WRITE, ref csel, 0, out cout, 0, ref
bytesReturned);
Thread.Sleep(100);

csel = CPLD_ID_RFID_RST;

KernelIoControl(IOCTL_HAL_CPLD_CTRL_WRITE, ref csel, 0, out cout, 0, ref
bytesReturned);
Thread.Sleep(100);

string buffer = "";
ACG_CFReader.presetSettings settings;
settings.baudRate = 9600;
settings.protocol = 0;
int i = dllACG.RDR_OpenComm("MOC1", 6, settings);
if (i == 0)
    MessageBox.Show("OpenComm failed");
else
{
    i = dllACG.RDR_OpenReader(1, 0);
    if (i == 0)
        MessageBox.Show("OpenReader failed");
}
}
```

2.4.2 Close

RFID can be closed by closing COM port and cut the power to RFID through KernelIoControl so that Open and Close can not be used at the same program repeatedly.

C++

```
#include "CFReaderLib.h"
```



```

#define CPLD_ID_RFID_ON          0x24
#define CPLD_ID_RFID_RST        0x25

UINT nCPLDID = 0, uOut;
BOOL rc;
char com_port[10];
char buffer[514];

uOut = 0;

nCPLDID = CPLD_ID_RFID_ON;
rc = KernelIoControl(IOCTL_HAL_CPLD_CTRL_WRITE, &nCPLDID, sizeof(UINT), &uOut,
sizeof(UINT), NULL);
Sleep(50);

nCPLDID = CPLD_ID_RFID_RST;
rc = KernelIoControl(IOCTL_HAL_CPLD_CTRL_WRITE, &nCPLDID, sizeof(UINT), &uOut,
sizeof(UINT), NULL);
Sleep(100);

```

C#

```

using CFReaderDLLWrapper;

[DllImport("Coredll.dll")]
public static extern int KernelIoControl(uint dwIoControlCode, ref uint lpInBuf,
int nInBufSize, out uint lpOutBuf, int nOutBufSize, ref uint lpBytesReturned);

private const uint FILE_DEVICE_HAL = 0x00000101;
private const uint CPLD_ID_RFID_ON = 0x24;
private const uint CPLD_ID_RFID_RST = 0x25;
private const uint METHOD_BUFFERED = 0;
private const uint FILE_ANY_ACCESS = 0;

public static uint CTL_CODE(uint DeviceType, uint Function, uint Method, uint
Access)
{
    return ((DeviceType << 16) | (Access << 14) | (Function << 2) | Method);
}

uint csel;
uint cout;
uint bytesReturned = 0;

uout = 0;
uint IOCTL_HAL_CPLD_CTRL_WRITE = CTL_CODE(FILE_DEVICE_HAL, 2078, METHOD_BUFFERED,
FILE_ANY_ACCESS);

csel = CPLD_ID_RFID_RST;
KernelIoControl(IOCTL_HAL_CPLD_CTRL_WRITE, ref csel, 0, out cout, 0, ref
bytesReturned);
Thread.Sleep(100);

csel = CPLD_ID_RFID_ON;
KernelIoControl(IOCTL_HAL_CPLD_CTRL_WRITE, ref csel, 0, out cout, 0, ref
bytesReturned);
Thread.Sleep(100);

```

2.4.3 Antenna on/off

RFID antenna is a key point of communication with the tag, and draws high current. To efficiently use battery, the antenna control is an important issue. RFID device communicates with RFID module and the tag. The device sends command to the module, and the module performs action corresponding to the command. The command is sent by **RDR_SendCommand** and it sends the received data through **RDR_GetData** to the device. The **RDR_SendCommandGetData** can send commands to the module and obtain the received data

at the same time.

C++

```
#include "CFReaderLib.h"

void AntennaControl(BOOL bOn)
{
    char buffer[512] = {0x00};
    if(bOn == TRUE)
    {
        RDR_SendCommandGetData("pon", "", buffer);
    }
    else
    {
        RDR_SendCommandGetData("poff","",bufffer);
    }
}
```

C#

```
using CFReaderDLLWrapper;

public ACG_CFReader dllACG;
dllACG = new ACG_CFReader();

public void AntennaControl(bool bOn)
{
    string buffer = "";
    If(bOn == true)
    {
        dllACG.RDR_SendCommandGetData("pon", "", ref buffer);
    }
    Else
    {
        dllACG.RDR_SendCommandGetData("poff", "", ref buffer);
    }
}
```

2.4.4 Data Read

Reading data from tag starts with selecting tag to read. Once the tag to read is selected, the module gets Serial Data from the tag. And then, module can get Block Data if necessary.

C++

```
#include "CFReaderLib.h"

char buffer[514];
memset(buffer, 0x00, 514);

RDR_SendCommandGetData("s", "", buffer);
// buffer have a serial number of tag

memset(buffer, 0x00, 514);
RDR_EmptyCommRcvBuffer();

RDR_SendCommandGetData("rb", "01", buffer);
// buffer have the 01 block data

RDR_EmptyCommRcvBuffer();
```

C#

```

using CFReaderDLLWrapper;

public ACG_CFReader dllACG;
dllACG = new ACG_CFReader();

string buffer = "";

dllACG.RDR_SendCommandGetData("s", "", ref buffer);
// buffer have a serial number of tag

dllACG.RDR_EmptyCommRcvBuffer();

dllACG.RDR_SendCommandGetData("rb", "01", ref buffer);
// buffer have the 01 block data

dllACG.RDR_EmptyCommRcvBuffer();

```

2.4.5 Data Write

As in reading data, first, the tag must be selected before writing data. After that, input data corresponding to tag type or memory structure. For example, in general case of ISO15693 tag, hexadecimal of 4 bytes is input to 1 Block. The command is 'wb', and input data to write with Block number. The Serial Number written when the tag is manufactured can not be changed and/or re-write.

C++

```

#include "CFReaderLib.h"

char buffer[514];
memset(buffer, 0x00, 514);

RDR_SendCommandGetData("s", "", buffer);
// buffer have a serial number of tag

memset(buffer, 0x00, 514);
RDR_EmptyCommRcvBuffer();

RDR_SendCommandGetData("wb", "0101234567", buffer);
// buffer have the 01 block data

RDR_EmptyCommRcvBuffer();

```

C#

```

using CFReaderDLLWrapper;

public ACG_CFReader dllACG;
dllACG = new ACG_CFReader();

string buffer = "";

dllACG.RDR_SendCommandGetData("s", "", ref buffer);
// buffer have a serial number of tag

dllACG.RDR_EmptyCommRcvBuffer();

dllACG.RDR_SendCommandGetData("wb", "0101234567", ref buffer);
// buffer have the 01 block data

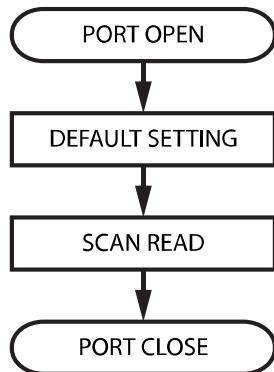
dllACG.RDR_EmptyCommRcvBuffer();

```

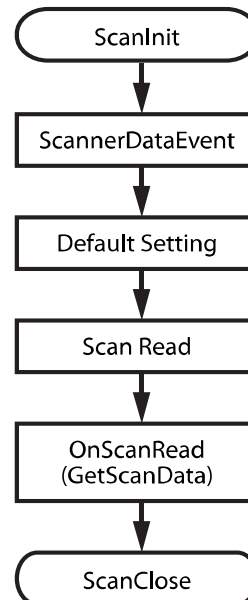
2.5 Scanner 1D

This SDK is applicable to 1D scanner modules.

Please refer to below flow charts for C++ and C# usage.



1D Scanner flow chart for C++



1D Scanner flow chart for C#

2.5.1 Initialization

Unlike M3 GREEN, separate power control is not required.
Scanner power will automatically turned on when scanner com port is open.
Comport Number : 6 / Baudrate : 230400

In Windows Mobile, registry value of side key must be set.
M3 ORANGE have different registry path.
Reg.h/Reg.cpp file related to registry setting is provided. (Reg Class is available.)

C++

[Header File]

```
#include "KScanBar.h"
```

...

```
CKScan m_KScan;
```

[CPP File]

```
#define DEVICE_M3SKY      1  
#define DEVICE_MM3       2  
#define DEVICE_M3ORANGE  3  
#define KEYPAD_NUMERIC   0  
#define KEYPAD_QWERTY    1
```

```
int      g_nDeviceType = 0;  
int      g_nKeyPadType = 0;  
int      g_nRightDownKey = 0;  
TCHAR    g_szSideKeyReg[1024] = {0, };
```

```

// Device Type

TCHAR szModel[33]= {0, };

SystemParametersInfo(SPI_GETOEMINFO, 64, szModel, SPIF_SENDCHANGE);

if(wcsncmp(szModel, L"MM3") == 0)
{
    g_nDeviceType = DEVICE_MM3;
}

else if((wcsncmp(szModel, L"M3ORANGE") == 0) || (wcsncmp(szModel, L"MC7101") == 0)
|| (wcsncmp(szModel, L"MC7X01") == 0))
{
    g_nDeviceType= DEVICE_M3ORANGE;
    g_nKeyPadType= m_Reg.GetRegValue(HKEY_LOCAL_MACHINE, L"ControlPanel\\keypad",
L"KeypadType");
}

else
{
    g_nDeviceType = DEVICE_M3SKY;
}

if(g_nDeviceType == DEVICE_M3ORANGE) // M3SKY Summit & M3ORANGE
{
    if(g_nKeyPadType == KEYPAD_QWERTY)
        wsprintf(g_szSideKeyReg, L"ControlPanel\\Keypad\\Qwerty");

    else
        wsprintf(g_szSideKeyReg, L"ControlPanel\\Keypad\\Numeric");
}

else
    wsprintf(g_szSideKeyReg, L"ControlPanel\\Keypad\\SideKey");

// Set SideKey

g_nRightDownKey = m_Reg.GetRegValue(HKEY_LOCAL_MACHINE, g_szSideKeyReg,
L"RightDownKey");

m_Reg.SetRegValue(HKEY_LOCAL_MACHINE, g_szSideKeyReg, L"RightDownKey", 3);


// Scanner Open

BOOL bRet = KScanOpen(6, FALSE, 230400, FALSE, NULL);

if(bRet == FALSE)
{
    ::MessageBox(NULL, L"Scanner Open Failed", NULL, MB_TOPMOST);
}

```

In C#, unlike in C++, ScanCtrl.ScanOpen() function does all necessary steps at once. ScanCtrl.ScanOpen() opens the com port.

Additionally, an event must be created to receive scan data message.
ScanCtrl.ScannerDataEvent += new ScannerDataDelegate(OnScanRead);

Receiving the data can be done in OnScanRead() function.

C#

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Runtime.InteropServices;    // DllImport
using Microsoft.Win32;                  // Registry
using System.Threading;                  // Sleep
using KScanbarNet;                       // KScanBarNet

namespace M3ScanTest_Net
{
    public partial class M3Scanner : Form
    {
        private KScanbarNet.ScannerControl ScanCtrl;
        public M3Scanner()
        {
            InitializeComponent();
            ScanCtrl = new ScannerControl();
            ScanCtrl.ScannerDataEvent += new ScannerDataDelegate(OnScanRead);

// SideKey Setting
            RegistryKey rk =
Registry.LocalMachine.OpenSubKey("ControlPanel\\KeyPad\\SideKey", true);

            if (rk == null)
            {
                rk =
Registry.LocalMachine.CreateSubKey("ControlPanel\\KeyPad\\SideKey");
            }

            else
            {
                rk.SetValue("RightDownKey", 3);
            }

// Scanner Open
            ScanCtrl.ScanOpen();
        }
    }
}

```

2.5.2 Default Settings

KSCANREAD structure that has scanner options must be initialized.

Initialize TimeOut, MinLen and SecurityLevel.

Receiving type of ScanData: Initialize to Callback type.

Initialize options for each barcodes.

C++

```

void CM3ScanTestDlg::SetDefaultOption(void)
{
    memset(&kRead, 0, sizeof(kRead));
    kRead.nSize = sizeof(kRead);

    memset(&kReadEx2, 0, sizeof(kReadEx2));
    strcat(kReadEx2.Signature, "KSCANEX2");

    kRead.nTimeInSeconds = 10;           // time out
    kRead.nMinLength = 2;                 // minimum data length
    kRead.nSecurity = 1;                  // security
}

```

```

    kReadEx2.XmitAIMID = DISABLE;
    kReadEx2.hwnd = this->m_hWnd;
    kReadEx2.UserMsg= WM_SCANDATA;
    kRead.fnCallBack = NULL;
    kRead.dwFlags = KSCAN_FLAG_REVERSEDIRECTION;
    kRead.dwFlags = KSCAN_FLAG_WIDESCANGLE;

//UPCA
    kReadEx2.UpcA.Enable = ENABLE;
    kReadEx2.UpcA.Format = AS_UPCA;
    kReadEx2.UpcA.XmitNumber = XMIT_NUMBER;
    kReadEx2.UpcA.XmitCheckDigit = XMIT_CHECK_DIGIT;
    kReadEx2.UpcA.Supp = NO_Supp;

//UPCE
    kReadEx2.UpcE.Enable = ENABLE;
    kReadEx2.UpcE.Format = AS_UPCE;
    kReadEx2.UpcE.XmitNumber = XMIT_NUMBER;
    kReadEx2.UpcE.XmitCheckDigit = XMIT_CHECK_DIGIT;
    kReadEx2.UpcE.Supp = NO_Supp;          // Not Supported

//EAN13
    kReadEx2.Ean13.Enable = ENABLE;
    kReadEx2.Ean13.Format = AS_BOOKLAND;    // Including AS_EAN13;
    kReadEx2.Ean13.XmitNumber = NO_XMIT_NUMBER;
    kReadEx2.Ean13.XmitCheckDigit = XMIT_CHECK_DIGIT;
    kReadEx2.Ean13.Supp = NO_Supp;

//EAN8
    kReadEx2.Ean8.Enable = ENABLE;
    kReadEx2.Ean8.Format = AS_EAN8;
    kReadEx2.Ean8.XmitNumber = NO_XMIT_NUMBER;
    kReadEx2.Ean8.XmitCheckDigit = XMIT_CHECK_DIGIT;
    kReadEx2.Ean8.Supp = NO_Supp;          // Not Supported

//Code39
    kReadEx2.Code39.Enable = ENABLE;
    kReadEx2.Code39.MinLength = 4;
    kReadEx2.Code39.MaxLength = 30;
    kReadEx2.Code39.SetAscii = STD_ASCII;
    kReadEx2.Code39.CDV = DISABLE;
    kReadEx2.Code39.XmitCheckDigit = NO_XMIT_CHECK_DIGIT;
    kReadEx2.Code39.AsCode32 = ENABLE;
    kReadEx2.Code39.AsPZN = ENABLE;

//Code128
    kReadEx2.Code128.Enable = ENABLE;
    kReadEx2.Code128.AsUCCEAN128 = ENABLE;
    kReadEx2.Code128.FNC1_ASCII = FNC1_Ascii;    //NULL: No FNC1 conversion
    kReadEx2.Code128.MinLength = 4;
    kReadEx2.Code128.MaxLength = 30;

//Code93
    kReadEx2.Code93.Enable = ENABLE;
    kReadEx2.Code93.MinLength = 4;
    kReadEx2.Code93.MaxLength = 30;

//Code35
    kReadEx2.Code35.Enable = ENABLE;

//Code11
    kReadEx2.Code11.Enable = ENABLE;
    kReadEx2.Code11.MinLength = 4;

```

```

    kReadEx2.Code11.MaxLength = 30;
    kReadEx2.Code11.CheckDigit = DIGIT1;
    kReadEx2.Code11.XmitCheckDigit = NO_XMIT_CHECK_DIGIT;

//Interleaved 2of5
    kReadEx2.Code25.Enable = ENABLE;
    kReadEx2.Code25.MinLength = 4;
    kReadEx2.Code25.MaxLength = 30;
    kReadEx2.Code25.CDV = DISABLE;
    kReadEx2.Code25.XmitCheckDigit = NO_XMIT_CHECK_DIGIT;
    kReadEx2.Code25.KindofDecode = (CODE25KIND_INTER |
CODE25KIND_ITF14|CODE25KIND_MATRIX | CODE25KIND_INDUSTRY | CODE25KIND_DLOGIC |
CODE25KIND_IATA);

//Codabar
    kReadEx2.Codabar.Enable = ENABLE;
    kReadEx2.Codabar.XmitStartStop = NO_XMIT;
    kReadEx2.Codabar.MinLength = 4;
    kReadEx2.Codabar.MaxLength = 30;

//MSI
    kReadEx2.Msi.Enable = ENABLE;
    kReadEx2.Msi.CDV = ENABLE;
    kReadEx2.Msi.XmitCheckDigit = NO_XMIT_CHECK_DIGIT;
    kReadEx2.Msi.MinLength = 4;
    kReadEx2.Msi.MaxLength = 30;

//Plessey
    kReadEx2.Plessey.Enable = ENABLE;
    kReadEx2.Plessey.CDV = ENABLE;
    kReadEx2.Plessey.XmitCheckDigit = NO_XMIT_CHECK_DIGIT;
    kReadEx2.Plessey.MinLength = 4;
    kReadEx2.Plessey.MaxLength = 30;

//GS1
    kReadEx2.Gs1.Enable = ENABLE;

//GS1 Limited
    kReadEx2.Gs1Limited.Enable = ENABLE;

//GS1 Expanded
    kReadEx2.Gs1Expanded.Enable = ENABLE;

//Telepen
    kReadEx2.Telepen.Enable = ENABLE;
    kReadEx2.Telepen.OldStyle = DISABLE;
    kRead.pReadEx=&kReadEx2;

    SetOption(&kRead);
    SetReturn(0, NULL);
}

```

C#

None.

2.5.3 ScanRead and GetDecodeData

This function reads and gets decoded data.

m_KScan.Read() function is called to Trigger On scanner.
ScanRead() function acquires the actual scan data.

C++

```
BEGIN_MESSAGE_MAP(CM3ScanTestDlg, CPropertySheet)
    ON_MESSAGE(WM_SCANDATA, OnScanRead)
END_MESSAGE_MAP()

void CM3ScanTestDlg::ScanReadNB()
{
    BOOL      bRet;

    if(m_bKeyFlag == FALSE)
    {
        if (m_bReading)
        {
            // Reading already in progress, now cancel it.
            bRet = KScanReadCancel();

            if (bRet)
            {
                m_bReading = FALSE;
                return;
            }

            else
            {
                ::MessageBox(NULL, KScanGetLastErrorMsg(), NULL, NULL);
            }
        }

        m_bKeyFlag = TRUE;

        if(m_bCon)
            m_bReading = TRUE;

        else
            m_bReading = FALSE;

        bRet = KScanRead();

        if (!bRet)
        {
            m_bReading = FALSE;
        }
    }
}

LRESULT CM3ScanTestDlg::OnScanRead(WPARAM wParam, LPARAM lParam)
{
    ScanRead((LPVOID)wParam);
    return FALSE;
}

void CM3ScanTestDlg::ScanRead(LPVOID pRead)
{
    pRead = &kRead;

    int      Status = kRead.out_Status;
    int      Type;
    CString  strData;

    HINSTANCE hInst = AfxGetInstanceHandle();

    CString aa;
```

```

switch(Status) {
    case KSCAN_RET_TIMEOUT:           // Timed out.
    case KSCAN_RET_USER_CANCEL:       // User called stop
    case KSCAN_RET_NORMAL:             // Barcode was read
    case KSCAN_RET_TYPE_UNKNOWN:

        Status = 0;

        m_bReading = FALSE;

        break;

    case KSCAN_RET_BAR_NOTFOUND:       // Not yet found - Continue.
    case KSCAN_RET_NORMAL_SWEEP:       // barcode was read, and security criteria
was not met yet

        Status = 1;

        break;

    default:                           // Other error.

        Status = 0;                   // just continue reading until barcode was
read with security criteria met

        m_bReading = FALSE;

        break;
}

if (Status == 0)
{
    if(((PKSCANREAD)pRead)->out_Status == KSCAN_RET_NORMAL)
    {
        Type = ((PKSCANREAD)pRead)->out_Type;
        g_strType = KScanSetBarCodeString(Type);
        strData.Format(_T("%S"), ((PKSCANREAD)pRead)->out_Barcode);
        m_ScanPage.Write_ListCtrl(g_strType, strData);

        if(m_SymbologyPage.m_bVibrate == TRUE)
        {
            VibrateOn(TRUE);
            Sleep(100);
            VibrateOn(FALSE);
        }

        if(m_SymbologyPage.m_nSound == 0)
            PlaySound(MAKEINTRESOURCE(IDR_SCAN_WAVE), hInst,
SND_RESOURCE|SND_ASYNC);

        else if(m_SymbologyPage.m_nSound == 1)
            PlaySound(MAKEINTRESOURCE(IDR_WAVE_BEEP), hInst,
SND_RESOURCE|SND_ASYNC);
        }
    }
}

```

Create an event for OnScanRead() which acquires the scan data when the program launch. Barcode type and data of a barcode is obtained from parameter, ScannerDataArgs e.

C#

```
private KScanbarNet.ScannerControl ScanCtrl;
```

```

public M3Scanner()
{
    InitializeComponent();
    ScanCtrl = new ScannerControl();
    ScanCtrl.ScannerDataEvent += new ScannerDataDelegate(OnScanRead);
}

public void ScanRead()
{
    if (m_bReading == true)
    {
        ScanCtrl.ScanReadCancel();
        m_bReading = false;
        return;
    }

    if (m_bContinue == false)
        m_bReading = false;

    else
        m_bReading = true;

    ScanCtrl.ScanRead();
}

public void OnScanRead(object sender, ScannerDataArgs e)
{
    if (LV_ScanData.Items.Count > 7)
        LV_ScanData.Items.Clear();

    if (e.ScanData != "")
    {
        ListViewItem ScanData = new ListViewItem();
        ScanData.Text = e.ScanType;
        ScanData.SubItems.Add(e.ScanData);
        LV_ScanData.Items.Add(ScanData);
        PlaySound(@"\windows\Alarm1.wav", 0, (int)(SND.SND_ASYNC |
SND.SND_FILENAME));
    }

    if (m_bContinue == false)
        m_bReading = false;
}

```

2.5.4 Close Scanner

Terminating the scanner is done by closing the com port.
SideKey Registry value set when scanner opens is returned to initial value.

C++

```

int i = 0;
m_Reg.SetRegValue(HKEY_LOCAL_MACHINE, g_szSideKeyReg, L"RightDownKey",
g_nRightDownKey);

for(i=0;i<3;i++)
{
    if(KScanClose())
        break;
    Sleep(100);
}

```

C#

```

// Scanner Close

for (int i = 0; i < 3; i++)
{
    m_bResult = ScanCtrl.ScanClose();
    Thread.Sleep(300);

    if (m_bResult == true)
        break;
}

// SideKey
RegistryKey rk =
Registry.LocalMachine.OpenSubKey("ControlPanel\\KeyPad\\SideKey", true);

if (rk == null)
{
    rk = Registry.LocalMachine.CreateSubKey("ControlPanel\\KeyPad\\SideKey");
}

else
{
    rk.SetValue("RightDownKey", 0);
}

Close();

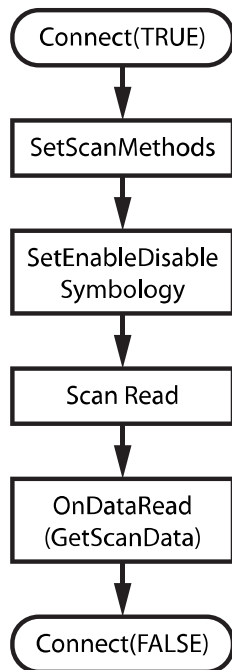
Application.Exit();

```

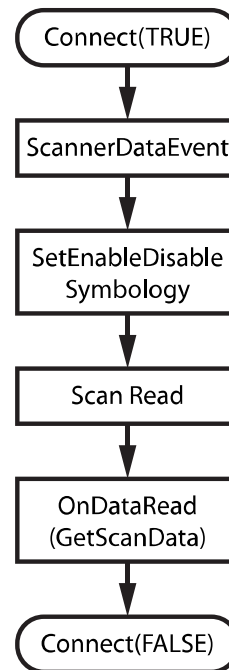
2.6 Scanner 2D (Imager)

This SDK is applicable to 2D scanner (imager) modules.

Please refer to below flow charts for C++ and C# usage.



2D Scanner flow chart for C++



2D Scanner flow chart for C#

2.6.1 Initialization

NOTE: Imager driver and Camera driver are sharing the same CPU interface. Consequently, 2D scanner and Camera cannot be used at the same time.

C++

```
if (Connect (TRUE) != TRUE)
{
    MessageBox (L"Connect Failed");
}
```

C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using M3MobileImagerNet;
using System.Runtime.InteropServices;
using System.Threading;
using System.Reflection;
using System.IO;

namespace Scan3Net
{
    public partial class Form1 : Form
    {
        private Scanner m_scan;
```

```

private ScannerControl ScanCtrl;
public Form1()
{
    InitializeComponent();
    m_scan = new Scanner();
    ScanCtrl = new ScannerControl();
    ScanCtrl.ScannerDataEvent += new ScannerDataDelegate(OnScanData);

    if (!m_scan.Connect(true))
        MessageBox.Show("Connect failed");
}
}
}

```

2.6.2 Default Settings

Registering an event and Symbology settings.

C++

```

SetScanMethods(NULL, m_hWnd, NULL);
SetEnableDisableSymbology(ID_ALL, TRUE);

```

C#

```

m_scan.SetEnableDisableSymbology(SYMID.ID_ALL, true);

```

2.6.3 ScanRead and GetDecodeData

This function reads and gets decoded data.

ScanRead() function is called to Trigger On scanner.
OnDataRead() function acquires the actual scan data.

C++

```

BEGIN_MESSAGE_MAP(CMainSheet, CPropertySheet)
//{{AFX_MSG_MAP(CMainSheet)
ON_MESSAGE(WM_SCAN_DATA, OnDataRead)
ON_WM_DESTROY()
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

BOOL CMainSheet::ScanRead()
{
    return ::ScanRead(m_nTimeOut, NULL);
}

void CMainSheet::OnDataRead()
{
    ScanLed(TRUE);

    DECODE_MSG decodeInfo;
    EventType_t eventType;
    GetScanResult(&eventType, &decodeInfo);

    Check_Barcode(decodeInfo.chCodeID, decodeInfo.chSymLetter,
decodeInfo.chSymModifier, decodeInfo.pchMessage);

    Sleep(150);
    ScanLed(FALSE);
}

```

Create an event for OnScanData(), which acquires scan data, when the program starts.

Barcode type and data is received through ScannerDataArgs which is a parameter of OnScanData().

C#

```
public Form1()
{
    InitializeComponent();

    m_scan = new Scanner();
    ScanCtrl = new ScannerControl();

    ScanCtrl.ScannerDataEvent += new ScannerDataDelegate(OnScanData);
}

private void Form1_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.F22)
    {
        if (m_bKeyFlag == false)
        {
            m_bKeyFlag = true;
            Scanner.DECODE_MSG msg = new Scanner.DECODE_MSG();
            int nTimeout = int.Parse(TimoutTextBox.Text);
            m_scan.ScanRead(nTimeout, ref msg);
        }
    }
}

private void Form1_KeyUp(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.F22)
    {
        if (m_bKeyFlag == true)
        {
            if (m_bSyncMode == false)
                m_scan.CancelIO();

            m_bKeyFlag = false;
        }
    }
}

private void OnScanData(object sender, ScannerDataArgs e)
{
    m_scan.GetScanResult(ref m_scan.m_event, ref m_scan.m_msg);
    ListViewItem BarcodeItem = new ListViewItem();
    switch (m_scan.m_msg.chSymLetter)
    {
        case '<':
            BarcodeItem.Text = "Code 32";
            break;
        case 'l':
            BarcodeItem.Text = "Code 49";
            break;
        case 'M':
            BarcodeItem.Text = "Code 4CB";
            break;
        case 'w':
            BarcodeItem.Text = "DATA Matrix";
            break;
        case 'j':
            if (m_scan.m_msg.chSymModifier == '4')
                BarcodeItem.Text = "ISBT 128";
    }
}
```

```

        else
            BarcodeItem.Text = "Code 128";
        break;
    case 'J':
        BarcodeItem.Text = "Japan POST";
        break;
    case 'K':
        BarcodeItem.Text = "KIX POST";
        break;
    case 'm':
        BarcodeItem.Text = "MATRIX 2of5";
        break;
    case 'x':
        BarcodeItem.Text = "MaxiCode";
        break;
    case 'g':
        BarcodeItem.Text = "MSI";
        break;
    case 'R':
        BarcodeItem.Text = "Micro PDF417";
        break;
    case 'L':
        BarcodeItem.Text = "Planet Code";
        break;
    case 'n':
        BarcodeItem.Text = "Plessey Code";
        break;
    case 'W':
        BarcodeItem.Text = "PosiCode";
        break;
    case 'P':
        BarcodeItem.Text = "Postnet";
        break;
    case 's':
        BarcodeItem.Text = "QR Code";
        break;
    case 'f':
        if (m_scan.m_msg.chCodeID == 'R')
            BarcodeItem.Text = "Straight 2of5 IATA";
        else
            BarcodeItem.Text = "Straight 2of5 Industria";
        break;
    case 'T':
        BarcodeItem.Text = "TCIF Linked Code39";
        break;
    case 't':
        BarcodeItem.Text = "Telepen";
        break;
    case '=':
        BarcodeItem.Text = "Trioptic Code";
        break;
    case 'A':
        BarcodeItem.Text = "Austria POST";
        break;
    case 'z':
        BarcodeItem.Text = "Aztec Code";
        break;
    case 'Z':
        BarcodeItem.Text = "Aztec Mesa";
        break;
    case 'B':
        BarcodeItem.Text = "British POST";
        break;
    case 'C':

```



```

        BarcodeItem.Text = "Canada POST";
        break;
    case 'Q':
        BarcodeItem.Text = "China POST";
        break;
    case 'q':
        BarcodeItem.Text = "Codablock F";
        break;
    case 'a':
        BarcodeItem.Text = "Codabar";
        break;
    case 'h':
        BarcodeItem.Text = "Code11";
        break;
    case 'o':
        BarcodeItem.Text = "Code 16K";
        break;
    case 'b':
        BarcodeItem.Text = "Code39";
        break;
    case 'D':
        BarcodeItem.Text = "EAN8";
        break;
    case 'd':
        BarcodeItem.Text = "EAN13";
        break;
    case 'e':
        BarcodeItem.Text = "Interleaved 2of5";
        break;
    case '?':
        BarcodeItem.Text = "KOREA POST";
        break;
    case 'r':
        BarcodeItem.Text = "PDF417";
        break;
    case 'c':
        BarcodeItem.Text = "UPC-A";
        break;
    case 'E':
        if (m_scan.m_msg.chCodeID == 'E')
            BarcodeItem.Text = "UPC-E";
        else
            BarcodeItem.Text = "UPC-E1";
        break;
    case 'i':
        BarcodeItem.Text = "Code93";
        break;
    case 'y':
        BarcodeItem.Text = "EAN/UCC Composite or Reduced Space Symbology";
        break;
    case 'I':
        BarcodeItem.Text = "GS1-128";
        break;
    case 'O':
        BarcodeItem.Text = "OCR";
        break;
    case 'N':
        BarcodeItem.Text = "UPU 4 State ID Tag";
        break;
    default:
        BarcodeItem.Text = "ETC";
        break;
}
}

```

2.6.4 Close Scanner

Terminates Imager and unload the imager driver.

C++

```
Connect (FALSE) ;
```

C#

```
private void Form1_Closing(object sender, CancelEventArgs e)
{
    m_scan.Connect(false);
}
```

3.0 Samples

This chapter illustrates the demo programs included in the SDK and current version of the SDK as well as the development tool used to write the sample program.

3.1 Bluetooth

Type	Demo	Source	Version	Date	Tool
C++	BTE_sample_1.exe	BTE_sample_1.exe	1.0.1	2011-01-03	Visual Studio 2005
C#.Net	BTE_sample_CS_1.exe	BTE_sample_CS_1.exe	1.0.1	2011-01-03	Visual Studio 2005

3.2 Camera

Type	Demo	Source	Version	Date	Tool
C++	Camera.exe M3SkyCamera.dll	Camera.exe	1.2.3	2011-01-20	Visual Studio 2005
C#.Net	CameraNet.exe M3SkyCamera.dll	CameraNet.exe	1.0.3	2011-01-04	Visual Studio 2005

Note that M3 ORANGE shares the same camera DLL with M3 SKY.

3.3 GPS

Type	Demo	Source	Version	Date	Tool
C++	GpsParseDemo.exe M3GpsParse.dll	GpsParseDemo.exe	1.0.3	2010-11-05	Visual Studio 2005
C#.Net	GpsParseDemoNet.exe M3GpsParse.dll	GpsParseDemoNet.exe	1.0.2	2010-06-07	Visual Studio 2005

3.4 RFID

Type	Demo	Source	Version	Date	Tool
C++	RF_ID_Demo.exe CFReader.dll	RF_ID_Demo.exe	1.1.1	2010-12-16	Visual Studio 2005
C#.Net	RF_ID_Demo_Net.exe CFReader.dll CFReaderDLLWrapper.dll	RF_ID_Demo_Net.exe	1.0.1	2011-01-03	Visual Studio 2005

3.5 Scanner 1D (Software Decoder only)

Type	Demo	Source	Version	Date	Tool
C++	M3ScanTest.exe ScanEmul.exe KScanBar.dll	M3ScanTest.exe	2.6.0	2010-10-25	Visual Studio 2005
C#.Net	M3ScanTest_Net.exe KScanBar.dll KScanBarNet.dll	M3ScanTest_Net.exe	3.1.0	2010-11-19	Visual Studio 2005

3.6 Scanner 2D (Imager)

Type	Demo	Source	Version	Date	Tool
C++	M3ScanTest.exe ScanEmul.exe TestCam.exe M3MobileImager.dll	M3ScanTest.exe	2.3.0	2010-10-25	Visual Studio 2005
		TestCam.exe	2.2.1	2011-01-14	Visual Studio 2005
C#.Net	M3ScanTest_Net.exe TestCam_Net.exe M3MobileImager.dll M3MobileImagerNet.dll	M3ScanTest_Net.exe	2.0.0	2010-03-08	Visual Studio 2005
		TestCam.exe	2.0.0	2010-03-08	Visual Studio 2005

4.0 References (Function Lists)

This chapter provides references of the modules included in M3 ORANGE. APIs are described using C / C++ language. Applications are created using Visual Studio 2005 and they are compatible with higher version of Visual Studio.

For accessing C / C++ style API in Visual Studio 2005, we supply M3 ORANGE SDK 1.0.0 Type Library for Visual Studio 2005. Users can add this type library to their projects using browse button in the References dialog (drop down the Project menu and select the References item).

4.1 Bluetooth

This section provides description of the functions and DLLs which are used to manage the Blue module.

Required Files

For C++

Required header:

CustomerDLL.h

Required lib:

N/A

Required DLL:

Does not require DLL in Bluetooth (Already included in Windows folder)

For C#

Required DLL:

BTEAPIdll.dll
BTEAPIdllNet.dll

Supported Product

M3 ORANGE with Stonestreet One BT Stack

4.1.1 Reference and Function List for C++

Structure

BTP_Connection_Info_t – Holds the information about a connection to be made or searched. This currently supports SPP connections.

```
typedef struct
{
    BTP_Connection_ID    ConnectionID;
    BD_ADDR_t            BD_ADDR;
    unsigned int          RFCOMMPort;
    int                   LocalCOMPort;
    unsigned char          MajorVersion;
    unsigned char          MinorVersion;
    Word_t                ConnectionAttributes;
    BTP_Profile_Type      ProfileType;
} BTP_Connection_Info_t;
```

BTP_Connection_Info_t – Field Description

ConnectionID	Unique identifier for the connection supplied by a call to BTPCreateConnection.
BD_ADDR	Address of the remote device hosting the connection.
RFCOMMPort	RFCOMM server port used for the connection.
LocalCOMPort	Local COM port for the connection. If no port is specified (i.e. assign a value of -1), the connection will use the first available port.
MajorVersion	Major version number of the connection's profile.
MinorVersion	Minor version number of the connection's profile.
ConnectionAttributes	Bit-mask defining the connection's attributes. Valid attributes include: BTP_CONNECTION_NONE BTP_CONNECTION_REMEMBERED BTP_CONNECTION_ACTIVE BTP_CONNECTION_ALL
ProfileType	Defines the type of profile used by the connection. Valid values include: BTP_PROFILE_SPP BTP_PROFILE_UNKNOWN

BTP_Connection_Info_Ex_t – Holds the information about a connection to be made or searched. This currently supports SPP or HID connections.

```
typedef struct _tagBTP_Connection_Info_Ex_t
{
    unsigned int          Size;
    unsigned int          Version;
    BTP_Connection_ID    ConnectionID;
    BD_ADDR_t            BD_ADDR;
    int                   LocalCOMPort;
    unsigned char          MajorVersion;
    unsigned char          MinorVersion;
    Word_t                ConnectionAttributes;
    BTP_Profile_Type      ProfileType;
    HLOCAL                ListHandle;
    unsigned int          ListIndex;

    union
    {
        BTPSerialPortProfileInfo_t    RemoteSerialPortProfileInfo;
        BTPHIDProfileInfo_t           RemoteHIDProfileInfo;
    }
}
```

```

    }ProfileInformation;
} BTP_Connection_Info_Ext;

```

BTP_Connection_Info_Ext – Field Descriptions

Size	Fill with the size of the BTP_Connection_Info_Ext structure.
Version	Not used by the application code in this version.
ConnectionID	Unique identifier for the connection supplied by a call to BTPCreateConnection or BTPCreateConnectionEx.
BD_ADDR	Address of the remote device hosting the connection.
LocalCOMPort	Local COM port for the connection. If no port is specified (i.e. assign a value of -1), the connection will use the first available port.
MajorVersion	Major version number of the connection's profile.
MinorVersion	Minor version number of the connection's profile.
ConnectionAttributes	Bit-mask defining the connection's attributes. Valid attributes include: BTP_CONNECTION_NONE BTP_CONNECTION_REMEMBERED BTP_CONNECTION_ACTIVE BTP_CONNECTION_ALL
ProfileType	Defines the type of profile used by the connection. Valid values include: BTP_PROFILE_SPP BTP_PROFILE_HID_DEVICE BTP_PROFILE_UNKNOWN
ListHandle	Not used Currently. For future enhancements.
ListIndex	Not used Currently. For future enhancements.
RemoteSerialPortProfileInfo	Described Below. To be used to store information about an SPP profile. The elements are typically filled up by the application code using the information received from a corresponding service. See BTP_Service_Info_Ext.
RemoteHIDProfileInfo	Described Below. To be used to store information about an HID profile. The elements are typically filled up by the application code using the information received from a corresponding service. See BTP_Service_Info_Ext. <pre> typedef struct _tagBTPSerialPortProfileInfo_t { unsigned int RFCOMMServerPort; bool UseActiveSync; } BTPSerialPortProfileInfo_t; typedef struct _tagBTPHIDProfileInfo_t { unsigned int L2CAPControlChannel; unsigned int L2CAPInterruptChannel; Byte_t DeviceSubclass; bool VirtualCableSupported; bool DeviceAutomaticReconnect; bool DeviceNormallyConnectable; } BTPHIDProfileInfo_t; </pre>

BTP_Connection_Query_t – This structure specifies search criteria for connection searches.

```

typedef struct
{
    Word_t ConnectionAttributes;
} BTP_Connection_Query_t;

```

BTP_Connection_Query_t – Field Description

ConnectionAttributes	Bit-mask that filters the enumeration of connections by attribute. Valid attributes include: BTP_CONNECTION_NONE BTP_CONNECTION_REMEMEBER BTP_CONNECTION_ACTIVE BTP_CONNECTION_ALL
-----------------------------	--

BTP_Device_Info_t – This structure defines a remote Bluetooth device.

```
typedef struct
{
    BD_ADDR_t          BD_ADDR;
    Class_of_Device_t  ClassOfDevice;
    Word_t             DeviceAttributes;
    char               Name[BLUETOOTH_MAX_NAME_SIZE];
} BTP_Device_Info_t;
```

BTP_Device_Info_t – Field Description

BD_ADDR	Address of the remote Bluetooth device.
ClassOfDevice	Bit-mask list of features that determine the class of device for the remote Bluetooth device.
DeviceAttributes	Bit-mask defining the remote device's attributes. Valid attributes include: BTP_DEVICE_NONE BTP_DEVICE_AUTHENTICATED BTP_DEVICE_REMEMBERED BTP_DEVICE_CONNECTED BTP_DEVICE_ALL
Name	Name of the remote Bluetooth device.

BTP_Device_Query_t - This structure provides parameters for device searches.

```
typedef struct
{
    BD_ADDR_t          BD_ADDR;
    Class_of_Device_t  ClassOfDevice;
    Word_t             DeviceAttributes;
    char               Name[BLUETOOTH_MAX_NAME_SIZE];
} BTP_Device_Info_t;
```

BTP_Device_Query_t – Field Description

DeviceAttributes	Bit-mask defining the criteria for devices to be enumerated in the call to BTPFindFirstDevice. Valid values include: BTP_DEVICE_NONE BTP_DEVICE_AUTHENTICATED BTP_DEVICE_REMEMBERED BTP_DEVICE_CONNECTED BTP_DEVICE_ALL
InquiryTimeout	Timeout period in seconds for the GAP device discovery. A value of zero (0) indicates that a GAP inquiry is not requested.

BTP_Device_Query_Ex_t – This structure provides parameters for device searches and supports searching for HID devices only.

```
typedef struct _tagBTP_Device_Query_Ex_t
{
    unsigned int      Size;
    Byte_t            Version;
    Word_t            DeviceAttributes;
    Byte_t            InquiryTimeout;
    BTP_DeviceType_t  DeviceType;
} BTP_Device_Query_Ex_t;
```

BTP_Device_Query_Ex_t – Field Description

Size	Fill with the size of the BTP_Device_Query_Ex_t structure.
Version	Not used by the application code at this time.
DeviceAttributes	Bit-mask defining the criteria for devices to be enumerated in the call to BTPFindFirstDevice. Valid values include: BTP_DEVICE_NONE BTP_DEVICE_AUTHENTICATED BTP_DEVICE_REMEMBERED BTP_DEVICE_CONNECTED BTP_DEVICE_ALL
InquiryTimeout	Timeout period in seconds for the GAP device discovery. A value of zero (0) indicates that a GAP inquiry is not requested.
BTP_DeviceType_t	Enum data type used to specify the type of the devices to search for. <pre>typedef enum { bdAll, //returns all devices bdHID //returns only HID devices } BTP_DeviceType_t;</pre>

BTP_Service_Info_t – This structure defines a service offered by a remote Bluetooth device.

```
typedef struct
{
    BTP_Profile_Type  ProfileType;
    unsigned char     MajorVersion;
    unsigned char     MinorVersion;
    char              ServiceName[BLUETOOTH_MAX_NAME_SIZE];
    unsigned int       RFCOMMPort;
} BTP_Service_Info_t;
```

BTP_Service_Info_t – Field Description

ProfileType	Defines the type of profile for the remote service. Valid values include: BTP_PROFILE_SPP BTP_PROFILE_UNKNOWN
MajorVersion	Major version number of the service's profile.
MinorVersion	Minor version number of the service's profile.
ServiceName	Name of the remote service.
RFCOMMPort	RFCOMM server port used for the remote service.

BTP_Service_Info_Ex_t – This structure defines a service offered by a remote Bluetooth device. This currently supports SPP and HID service information.

```

typedef struct _tagBTP_Service_Info_Ex_t
{
    unsigned int          Size;
    unsigned int          Version;
    BTP_Profile_Type      ProfileType;
    unsigned char         MajorVersion;
    unsigned char         MinorVersion;
    char                  ServiceName[BLUETOOTH_MAX_NAME_SIZE];
    unsigned int          ListIndex;

    union
    {
        BTPSerialPortProfileInfo_t    RemoteSerialPortProfileInfo;
        BTPHIDProfileInfo_t           RemoteHIDProfileInfo;
    } ProfileInformation;
} BTP_Service_Info_Ex_t;

```

BTP_Service_Info_Ex_t – Field Description

Size	Fill with the size of the BTP_Service_Info_Ex_t structure.
Version	Not used by the application code in this version.
ProfileType	Defines the type of profile for the remote service. Valid values include: BTP_PROFILE_SPP BTP_PROFILE_HID_DEVICE BTP_PROFILE_UNKNOWN
MajorVersion	Major version number of the service's profile.
MinorVersion	Minor version number of the service's profile.
ServiceName	Name of the remote service
ListIndex	Not used currently. For future enhancements.
RemoteSerialPortProfileInfo	Described Below. To be used to grab information about an SPP profile. The elements need to be saved in the new Connection_Info_Ex_t structure to create or search a connection.
RemoteHIDProfileInfo	<p>Described Below. To be used to grab information about an HID profile. The elements need to be saved in the new Connection_Info_Ex_t structure to create or search a connection.</p> <pre> typedef struct _tagBTPSerialPortProfileInfo_t { unsigned int RFCOMMServerPort; bool UseActiveSync; } BTPSerialPortProfileInfo_t; typedef struct _tagBTPHIDProfileInfo_t { unsigned int L2CAPControlChannel; unsigned int L2CAPInterruptChannel; Byte_t DeviceSubclass; bool VirtualCableSupported; bool DeviceAutomaticReconnect; bool DeviceNormallyConnectable; } BTPHIDProfileInfo_t; </pre>

BTP_Service_Query_t – This structure defines the parameters for an SDP service query.

```

typedef struct
{
    BD_ADDR_t      BD_ADDR;
    Word_t         NumberServiceUUID;
}

```

```

    SDP_UUID_Entry_t *Service;
} BTP_Service_Query_t;

```

BTP_Service_Query_t – Field Description

BD_ADDR	Address of the remote Bluetooth device whose services are being queried.
NumberServiceUUID	Specifies the number of service UUIDs in the array pointed to by the Service field in this structure. A value of zero (0) indicates that the SDP query will search for all available services.
Service	Pointer to an array of SDP UUID entries. Set to NULL if a general SDP query is requested (see NumberServiceUUID description above). See sample code for help on searching for SPP or HID services using this parameter.

BTP_PIN_t - Used in setting or clearing the incoming or outgoing PIN.

```

typedef struct _tagBTP_PIN_t
{
    unsigned int    PINLength;
    BTP_PIN_Code_t  PINCode;
} BTP_PIN_t;

```

BTP_PIN_t – Field Description

PINLength	The length of the PIN code. This can be either the number of characters in PINCode to be used as PIN or 0 if it is used to clear incoming or outgoing PIN.
PINCode	The PIN Code to be used as incoming or outgoing PIN. This need not be filled if you are trying to clear the PIN (PINLength is 0).

BTP_PIN_Code_t - Structure that holds the PIN code.

```

typedef struct _tagBTP_PIN_Code_t
{
    Byte_t PIN_Code0;
    Byte_t PIN_Code1;
    Byte_t PIN_Code2;
    Byte_t PIN_Code3;
    Byte_t PIN_Code4;
    Byte_t PIN_Code5;
    Byte_t PIN_Code6;
    Byte_t PIN_Code7;
    Byte_t PIN_Code8;
    Byte_t PIN_Code9;
    Byte_t PIN_Code10;
    Byte_t PIN_Code11;
    Byte_t PIN_Code12;
    Byte_t PIN_Code13;
    Byte_t PIN_Code14;
    Byte_t PIN_Code15;
} BTP_PIN_Code_t;

```

BTP_PIN_Code_t – Field Description

PIN_Code	The 'n'th byte of the PIN code. See sample code or SetIncomingPIN or SetOutgoingPIN documentation to see a macro that can set this easily.
-----------------	--

BD_ADDR_t - This structure represents a Bluetooth Board Address.

```

typedef struct

```

```

{
    Byte_t BD_ADDR0;
    Byte_t BD_ADDR1;
    Byte_t BD_ADDR2;
    Byte_t BD_ADDR3;
    Byte_t BD_ADDR4;
    Byte_t BD_ADDR5;
} BD_ADDR_t;

```

Note : BD_ADDR5 is the highest order byte and BD_ADDR0 is the lowest order byte.

SDP_UUID_Entry_t - This structure defines the parameters for an SDP query.

```

typedef struct
{
    SDP_Data_Element_Type_t Data_Element_Type;
    union
    {
        UUID_16_t      UUID_16;
        UUID_32_t      UUID_32;
        UUID_128_t     UUID_128;
    } UUID_Value;
} SDP_UUID_Entry_t;

```

SDP_UUID_Entry_t – Field Description

Data_Element_Type	Specifies which field from the UUID_Value union is valid. Valid values include: deUUID_16 deUUID_32 deUUID_128
UUID_Value	This union contains a 16-, 32-, or 128-bit UUID structure, each of which is defined below.

Additional Structures

```

typedef struct
{
    Byte_t UUID_Byte0;
    Byte_t UUID_Byte1;
} UUID_16_t;

```

```

typedef struct
{
    Byte_t UUID_Byte0;
    Byte_t UUID_Byte1;
    Byte_t UUID_Byte2;
    Byte_t UUID_Byte3;
} UUID_32_t;

```

```

typedef struct
{
    Byte_t UUID_Byte0;
    Byte_t UUID_Byte1;
    Byte_t UUID_Byte2;
    Byte_t UUID_Byte3;
    Byte_t UUID_Byte4;
    Byte_t UUID_Byte5;
    Byte_t UUID_Byte6;
    Byte_t UUID_Byte7;
    Byte_t UUID_Byte8;
    Byte_t UUID_Byte9;
    Byte_t UUID_Byte10;
    Byte_t UUID_Byte11;
}

```

```
    Byte_t UUID_Byte12;  
    Byte_t UUID_Byte13;  
    Byte_t UUID_Byte14;  
    Byte_t UUID_Byte15;  
} UUID_128_t;
```

4.1.1.1 BTPInitAPI

This function initializes the BTE Explorer API module, readying it for use by the application.

```
bool BTPInitAPI (void)
```

Parameters

None

Return Values

Returns true if successful.

C++

```
#include "CustomerDLL.h"

if(BTPInitAPI())
{
    // Init success
}
else
{
    // Init Fail
}
```

4.1.1.2 BTPCreateConnection

This function defines a temporary or persistent (Favorite) connection, which may later be connected by a call to **BTP_Connect** (Supports SPP only).

```
HRESULT BTPCreateConnection (BTP_Connection_Info_t *ConnectionInfo)
```

Parameters

ConnectionInfo

Information defining the new connection.

Also, the new connection ID is returned in this structure. This version supports SPP connections only.

Return Values

BTP_ERROR_SUCCESS if successful.

Error codes include the following values :

BTP_ERROR

BTP_ERROR_INVALID_PARAMETER

C++

```
#include "CustomerDLL.h"

// Find Connection

BTP_Connection_Info_Ex_t connectionInfo;
HRESULT result;
/* Create a favorite association */
connectionInfo.ConnectionAttributes = BTP_CONNECTION_REMEMBERED |
BTP_CONNECTION_ACTIVE;
connectionInfo.LocalCOMPort = 9;
connectionInfo.ProfileType = BTP_PROFILE_SPP;
result = BTPCreateConnectionEx(&connectionInfo);

if (BTP_ERROR_SUCCESS == result)
    SetWindowText(L"create connection");
else
    AfxMessageBox(L"Create Connection_error");
```

4.1.1.3 BTPCreateConnectionEx

This function defines a temporary or persistent (Favorite) connection, which may later be connected by a call to **BTP_Connect** (Supports SPP and HID).

```
HRESULT BTPCreateConnection (BTP_Connection_Info_Ext *ConnectionInfoEx)
```

Parameters

ConnectionInfoEx

Information defining the new connection.

Also, the new connection ID is returned in this structure. Currently supports connecting to HID and SPP.

Return Values

BTP_ERROR_SUCCESS if successful.

Error codes include the following values :

BTP_ERROR

BTP_ERROR_INVALID_PARAMETER

C++

```
#include "CustomerDLL.h"

// Create Connection

BTP_Connection_Info_Ext connectionInfo;
HRESULT result;
    /* Create a favorite association */
connectionInfo.ConnectionAttributes = BTP_CONNECTION_REMEMBERED |
BTP_CONNECTION_ACTIVE;
connectionInfo.LocalCOMPort = 9;
connectionInfo.ProfileType = BTP_PROFILE_SPP;
result = BTPCreateConnectionEx(&connectionInfo);

if (BTP_ERROR_SUCCESS == result)
    SetWindowText(L"create connection");
else
    AfxMessageBox(L"Create Connection error");
```

4.1.1.4 BTPConnect

This function connects to a connection previously defined by a call to **BTPCreateConnection**.

```
HRESULT BTPConnect (BTP_Connection_ID ConnectionID)
```

Parameters

ConnectionID

Unique identifier for a connection which was previously defined by a call to BTPCreateConnnection and BTPCreateConnectionEx.

Return Values

BTP_ERROR_SUCCESS if successful.

Error codes include the following values :

BTP_ERROR

BTP_ERROR_INVALID_PARAMETER

C++

```
#include "CustomerDLL.h"

//Delete Connection
```

```

BTP_Connection_Info_Ext connectionInfo;
HRESULT result;

result=BTPConnect(connectionInfo.ConnectionID);

if(result==BTP_ERROR_SUCCESS)
    SetWindowText(L"Connect");
else if(result ==BTP_ERROR_INVALID_PARAMETER)
    AfxMessageBox(L"ERROR_INVALID_PARAMETER");
else if(result ==BTP_ERROR)
    AfxMessageBox(L"error");

```

4.1.1.5 BTPFindFirstConnection

This function creates a list of active connections and Favorites to traverse, returning the first connection from the list.

```

HRESULT BTPFindFirstConnection (BTP_Connection_Find *ConnectionFind,
    BTP_Connection_Info_t *ConnectionInfo, const BTP_Connection_Query_t *ConnectionQuery)

```

Parameters

ConnectionFind

Handle to the list of connections, needed for subsequent calls to BTPFindNextConnection and BTPFindConnectionClose.

ConnectionInfo

Information defining the first connection from the list.

ConnectionQuery

Provides filtering for the types of connection to be retrieved.

Return Values

BTP_ERROR_SUCCESS if successful.

Error codes include the following values :

BTP_ERROR
 BTP_ERROR_NO_MORE
 BTP_ERROR_INVALID_PARAMETER

C++

```

#include "CustomerDLL.h"

//Find Connection
m_ctrlList.DeleteAllItems();
m_ConnectItem.RemoveAll();

m_listtype = 3;           //list type
HRESULT result;
CString stradd;
BTP_Connection_Find      connectFind;
BTP_Connection_Info_t    connectioinfo;
BTP_Connection_Query_t   connectQuery;

memset(&connectQuery, 0, sizeof(connectQuery));

connectQuery.ConnectionAttributes = BTP_CONNECTION_REMEMBERED;

BeginWaitCursor();

result = BTPFindFirstConnection(&connectFind, &connectioinfo, &connectQuery);
if (BTP_ERROR_SUCCESS == result)
{

```



```

EndWaitCursor();
do
{
BTP_Connection_Info_t connectInfo_temp;
memcpy(&connectInfo_temp,&connectioinfo,sizeof(BTP_Connection_Info_t));
m_Connectitem.AddTail(connectInfo_temp);

result = BTPFindNextConnection(connectFind, &connectioinfo);

} while (BTP_ERROR_SUCCESS == result);

}
BTPFindConnectionClose(connectFind);

int lm_count = m_Connectitem.GetCount();
int i = 0;

for(i = lm_count-1;i>=0;i--)
{
    connectioinfo = m_Connectitem.GetAt(m_Connectitem.FindIndex(i));

    stradd.Format(L "[%02X:%02X:%02X:%02X:%02X:%02X] id:%d",
    connectioinfo.BD_ADDR.BD_ADDR5,
    connectioinfo.BD_ADDR.BD_ADDR4,
    connectioinfo.BD_ADDR.BD_ADDR3,
    connectioinfo.BD_ADDR.BD_ADDR2,
    connectioinfo.BD_ADDR.BD_ADDR1,
    connectioinfo.BD_ADDR.BD_ADDR0,
    connectioinfo.ConnectionID);

m_ctrllist.InsertItem(0, stradd,0);
}

```

4.1.1.6 BTPFindFirstConnectionEx

This function creates a list of active connections and Favorites to traverse, returning the first connection from the list.

```

HRESULT BTPFindFirstConnectionEx (BTP_Connection_Find *ConnectionFind,
    BTP_Connection_Info_Ex_t *ConnectionInfoEx,
    const BTP_Connection_Query_t ConnectionQuery)

```

Parameters

ConnectionFind

Handle to the list of connections, needed for subsequent calls to BTPFindNextConnectionEx and BTPFindConnectionClose.

ConnectionInfoEx

Information defining the first connection from the list. Supports SPP and HID connections.

ConnectionQuery

Provides filtering for the types of connection to be retrieved.

Return Values

BTP_ERROR_SUCCESS if successful.

Error codes include the following values :

BTP_ERROR
 BTP_ERROR_NO_MORE
 BTP_ERROR_INVALID_PARAMETER

C++

```
#include "CustomerDLL.h"

//Find Connection
m_ctrllist.DeleteAllItems();
m_Connectitem.RemoveAll();

m_listtype = 3;           //list type
HRESULT result;
CString stradd;
BTP_Connection_Find      connectFind;
BTP_Connection_Info_t    connectioinfo;
BTP_Connection_Query_t   connectQuery;

memset(&connectQuery, 0, sizeof(connectQuery));

connectQuery.ConnectionAttributes = BTP_CONNECTION_REMEMBERED;

BeginWaitCursor();

result = BTPFindFirstConnection(&connectFind, &connectioinfo, &connectQuery);
if (BTP_ERROR_SUCCESS == result)
{
    EndWaitCursor();
    do
    {
        BTP_Connection_Info_t connectInfo_temp;
        memcpy(&connectInfo_temp, &connectioinfo, sizeof(BTP_Connection_Info_t));
        m_Connectitem.AddTail(connectInfo_temp);

        result = BTPFindNextConnection(connectFind, &connectioinfo);
    } while (BTP_ERROR_SUCCESS == result);
}

BTPFindConnectionClose(connectFind);

int lm_count = m_Connectitem.GetCount();
int i = 0;

for(i = lm_count-1; i>=0; i--)
{
    connectioinfo = m_Connectitem.GetAt(m_Connectitem.FindIndex(i));

    stradd.Format(L "[%02X:%02X:%02X:%02X:%02X:%02X] id:%d",
        connectioinfo.BD_ADDR.BD_ADDR5,
        connectioinfo.BD_ADDR.BD_ADDR4,
        connectioinfo.BD_ADDR.BD_ADDR3,
        connectioinfo.BD_ADDR.BD_ADDR2,
        connectioinfo.BD_ADDR.BD_ADDR1,
        connectioinfo.BD_ADDR.BD_ADDR0,
        connectioinfo.ConnectionID);

    m_ctrllist.InsertItem(0, stradd, 0);
}
```

4.1.1.7 BTPFindNextConnection

This function retrieves the next connection from the list originally created by a call to **BTPFindFirstConnection**.

```
HRESULT BTPFindNextConnection (BTP_Connection_Find ConnectionFind,
    BTP_Connection_Info_t *ConnectionInfo)
```

Parameters

ConnectionFind

Handle to the list connections, originally returned by a call to BTPFindFirstConnection.

ConnectionInfo

Information defining a connection from the list.

Return Values

BTP_ERROR_SUCCESS if successful.

Error codes include the following values :

BTP_ERROR

BTP_ERROR_NO_MORE

BTP_ERROR_INVALID_HANDLE

C++

```
#include "CustomerDLL.h"

// Find Connection
m_ctrllist.DeleteAllItems();
m_Connectitem.RemoveAll();

m_listtype = 3;           //list type
HRESULT result;
CString stradd;
BTP_Connection_Find       connectFind;
BTP_Connection_Info_t     connectioinfo;
BTP_Connection_Query_t    connectQuery;

memset(&connectQuery, 0, sizeof(connectQuery));

connectQuery.ConnectionAttributes = BTP_CONNECTION_REMEMBERED;

BeginWaitCursor();

result = BTPFindFirstConnection(&connectFind, &connectioinfo, &connectQuery);
if (BTP_ERROR_SUCCESS == result)
{
    EndWaitCursor();
    do
    {
        BTP_Connection_Info_t connectInfo_temp;
        memcpy(&connectInfo_temp, &connectioinfo, sizeof(BTP_Connection_Info_t));
        m_Connectitem.AddTail(connectInfo_temp);

        result = BTPFindNextConnection(connectFind, &connectioinfo);
    } while (BTP_ERROR_SUCCESS == result);
}
BTPFindConnectionClose(connectFind);

int lm_count = m_Connectitem.GetCount();
int i = 0;

for(i = lm_count-1; i >= 0; i--)
{
    connectioinfo = m_Connectitem.GetAt(m_Connectitem.FindIndex(i));

    stradd.Format(L "[%02X:%02X:%02X:%02X:%02X:%02X] id:%d",
        connectioinfo.BD_ADDR.BD_ADDR5,
        connectioinfo.BD_ADDR.BD_ADDR4,
        connectioinfo.BD_ADDR.BD_ADDR3,
        connectioinfo.BD_ADDR.BD_ADDR2,
```

```

        connectioinfo.BD_ADDR.BD_ADDR1,
        connectioinfo.BD_ADDR.BD_ADDR0,
        connectioinfo.ConnectionID);

    m_ctrllist.InsertItem(0, stradd,0);
}

```

4.1.1.8 BTPFindNextConnectionEx

This function retrieves the next connection from the list originally created by a call to **BTPFindFirstConnection**.

```

HRESULT BTPFindNextConnectionEx (BTP_Connection_Find ConnectionFind,
    BTP_Connection_Info_Ext *ConnectionInfoEx)

```

Parameters

ConnectionFind

Handle to the list connections, originally returned by a call to BTPFindFirstConnection.

ConnectionInfoEx

Information defining a connection from the list. Supports SPP and HID.

Return Values

BTP_ERROR_SUCCESS if successful.

Error codes include the following values :

BTP_ERROR

BTP_ERROR_NO_MORE

BTP_ERROR_INVALID_HANDLE

C++

```

#include "CustomerDLL.h"

//Find Connection
m_ctrllist.DeleteAllItems();
m_Connectitem.RemoveAll();

m_listtype = 3;           //list type
HRESULT result;
CString stradd;
BTP_Connection_Find      connectFind;
BTP_Connection_Info_t    connectioinfo;
BTP_Connection_Query_t   connectQuery;

memset(&connectQuery, 0, sizeof(connectQuery));

connectQuery.ConnectionAttributes = BTP_CONNECTION_REMEMBERED;

BeginWaitCursor();

result = BTPFindFirstConnection(&connectFind, &connectioinfo, &connectQuery);
if (BTP_ERROR_SUCCESS == result)
{
    EndWaitCursor();
    do
    {
        BTP_Connection_Info_t connectInfo_temp;
        memcpy(&connectInfo_temp,&connectioinfo,sizeof(BTP_Connection_Info_t));
        m_Connectitem.AddTail(connectInfo_temp);

        result = BTPFindNextConnection(connectFind, &connectioinfo);
    } while (BTP_ERROR_SUCCESS == result);
}

```

```

}
BTPFindConnectionClose(connectFind);

int lm_count = m_Connectitem.GetCount();
int i = 0;

for(i = lm_count-1;i>=0;i--)
{
    connectioinfo = m_Connectitem.GetAt(m_Connectitem.FindIndex(i));

    stradd.Format(L "[%02X:%02X:%02X:%02X:%02X:%02X] id:%d",
        connectioinfo.BD_ADDR.BD_ADDR5,
        connectioinfo.BD_ADDR.BD_ADDR4,
        connectioinfo.BD_ADDR.BD_ADDR3,
        connectioinfo.BD_ADDR.BD_ADDR2,
        connectioinfo.BD_ADDR.BD_ADDR1,
        connectioinfo.BD_ADDR.BD_ADDR0,
        connectioinfo.ConnectionID);

    m_ctrllist.InsertItem(0, stradd,0);
}

```

4.1.1.9 BTPFindFirstDevice

This function initializes a GAP inquiry, creating a list of discovered Bluetooth devices. The first device is returned by this function.

```

HRESULT BTPFindFirstDevice (BTP_Device_Find *DeviceFind,
    BTP_Device_Info_t *DeviceInfo,const BTP_Device_Query_t *DeviceQuery)

```

Parameters

DeviceFind

Handle to the list of discovered devices, needed for subsequent calls to BTPFindNextDevice and BTPFindDeviceClose.

DeviceInfo

Information defining the first device.

DeviceQuery

Provides parameters for the GAP Inquiry and filtering for the devices to retrieve.

Return Values

BTP_ERROR_SUCCESS if successful.

Error codes include the following values :

BTP_ERROR
 BTP_ERROR_NO_MORE
 BTP_ERROR_INVALID_PARAMETER

C++

```

#include "CustomerDLL.h"

// Device Find
m_listtype = 1;           //list type
m_ctrllist.DeleteAllItems();
m_Deviceitem.RemoveAll();
memset(&connectionInfo, 0, sizeof(connectionInfo));
connectionInfo.Size = sizeof(BTP_Connection_Info_Ex_t);
connectionInfo.ProfileType = BTP_PROFILE_SPP;
CString str,stradd,strname;

HRESULT result;

```

```

BTP_Device_Find          deviceFind;
BTP_Device_Info_t        deviceInfo;
BTP_Device_Query_Ex_t    deviceQuery;
memset(&deviceQuery, 0, sizeof(deviceQuery));
deviceQuery.Size = sizeof(BTP_Device_Query_Ex_t);

/* Looking for all devices either discovered or remembered */
deviceQuery.DeviceAttributes = BTP_DEVICE_ALL;
deviceQuery.DeviceType = (connectionInfo.ProfileType == BTP_PROFILE_SPP ?
bdAll : bdHID);

/* Tells the API to perform a ten second GAP inquiry, using zero (0) */
/* here will just return previously discovered devices. */
deviceQuery.InquiryTimeout = 10;

BeginWaitCursor();
result = BTPFindFirstDeviceEx(&deviceFind, &deviceInfo, &deviceQuery);

if (BTP_ERROR_SUCCESS == result)
{
    EndWaitCursor();
    do
    {
        BTP_Device_Info_t deviceInfo_temp;
        memcpy(&deviceInfo_temp, &deviceInfo, sizeof(BTP_Device_Info_t));

        m_Deviceitem.AddTail(deviceInfo_temp);
        result = BTPFindNextDevice(deviceFind, &deviceInfo);
    } while (BTP_ERROR_SUCCESS == result);
}

BTPFindDeviceClose(deviceFind);

int lm_count = m_Deviceitem.GetCount();
int i = 0;

for(i = lm_count-1; i>=0; i--)
{
    deviceInfo = m_Deviceitem.GetAt(m_Deviceitem.FindIndex(i));
    stradd.Format(L"%02X:%02X:%02X:%02X:%02X:%02X]Device: %d %s ",
deviceInfo.BD_ADDR.BD_ADDR5,
deviceInfo.BD_ADDR.BD_ADDR4,
deviceInfo.BD_ADDR.BD_ADDR3,
deviceInfo.BD_ADDR.BD_ADDR2,
deviceInfo.BD_ADDR.BD_ADDR1,
deviceInfo.BD_ADDR.BD_ADDR0,
deviceInfo.DeviceAttributes,
CString(deviceInfo.Name));
}

```

4.1.1.10 BTPFindFirstDeviceEx

This function initializes a GAP inquiry, creating a list of discovered Bluetooth devices of a particular device type or all devices. In this version it supports searching for HID devices. The first device that matches the filter is returned by this function.

```

HRESULT BTPFindFirstDeviceEx (BTP_Device_Find *DeviceFind,
    BTP_Device_Info_t *DeviceInfo, const BTP_Device_Query_Ex_t *DeviceQueryEx)

```

Parameters
DeviceFind

Handle to the list of discovered devices, needed for subsequent calls to BTPFindNextDevice and BTPFindDeviceClose.

DeviceInfo

Information defining the first device.

DeviceQueryEx

Provides parameters for the GAP Inquiry and filtering for the devices to retrieve.

Return Values

BTP_ERROR_SUCCESS if successful.

Error codes include the following values :

BTP_ERROR

BTP_ERROR_NO_MORE

BTP_ERROR_INVALID_PARAMETER

C++

```
#include "CustomerDLL.h"

//Device Find
m_listtype = 1;           //list type
m_ctrllist.DeleteAllItems();
m_Deviceitem.RemoveAll();
memset(&connectionInfo, 0, sizeof(connectionInfo));
connectionInfo.Size = sizeof(BTP_Connection_Info_Ex_t);
connectionInfo.ProfileType = BTP_PROFILE_SPP;
CString str,stradd,strname;

HRESULT result;
BTP_Device_Find           deviceFind;
BTP_Device_Info_t         deviceInfo;
BTP_Device_Query_Ex_t     deviceQuery;
memset(&deviceQuery, 0, sizeof(deviceQuery));
deviceQuery.Size = sizeof(BTP_Device_Query_Ex_t);

/* Looking for all devices either discovered or remembered */
deviceQuery.DeviceAttributes = BTP_DEVICE_ALL;
deviceQuery.DeviceType = (connectionInfo.ProfileType == BTP_PROFILE_SPP ?
bdAll : bdHID);

/* Tells the API to perform a ten second GAP inquiry, using zero (0) */

/* here will just return previously discovered devices. */
deviceQuery.InquiryTimeout = 10;

BeginWaitCursor();

result = BTPFindFirstDeviceEx(&deviceFind, &deviceInfo, &deviceQuery);

if (BTP_ERROR_SUCCESS == result)
{
    EndWaitCursor();
    do
    {
        BTP_Device_Info_t deviceInfo_temp;
        memcpy(&deviceInfo_temp,&deviceInfo,sizeof(BTP_Device_Info_t));

        m_Deviceitem.AddTail(deviceInfo_temp);
        result = BTPFindNextDevice(deviceFind, &deviceInfo);
    } while (BTP_ERROR_SUCCESS == result);
}
```

```

BTPFindDeviceClose(deviceFind);

int lm_count = m_Deviceitem.GetCount();
int i = 0;

for(i = lm_count-1; i>=0; i--)
{
    deviceInfo = m_Deviceitem.GetAt(m_Deviceitem.FindIndex(i));
    stradd.Format(L "[%02X:%02X:%02X:%02X:%02X:%02X]Device: %d %s ",
        deviceInfo.BD_ADDR.BD_ADDR5,
        deviceInfo.BD_ADDR.BD_ADDR4,
        deviceInfo.BD_ADDR.BD_ADDR3,
        deviceInfo.BD_ADDR.BD_ADDR2,
        deviceInfo.BD_ADDR.BD_ADDR1,
        deviceInfo.BD_ADDR.BD_ADDR0,
        deviceInfo.DeviceAttributes,
        CString(deviceInfo.Name));
}

```

4.1.1.11 BTPFindNextDevice

This function returns the next device in the list of discovered devices created by a previous call to **BTPFindFirstDevice**.

```

HRESULT BTPFindNextDevice (BTP_Device_Find DeviceFind,
    BTP_Device_Info t *DeviceInfo)

```

Parameters

DeviceFind

Handle to the list of discovered devices, originally returned by a call to BTPFindFirstDevice.

DeviceInfo

Information defining a remote device.

Return Values

BTP_ERROR_SUCCESS if successful.

Error codes include the following values :

BTP_ERROR
 BTP_ERROR_NO_MORE
 BTP_ERROR_INVALID_PARAMETER
 BBTP_ERROR_INVALID_HANDLE

C++

```

#include "CustomerDLL.h"

//Device Find
m_listtype = 1;           //list type
m_ctrllist.DeleteAllItems();
m_Deviceitem.RemoveAll();
memset(&connectionInfo, 0, sizeof(connectionInfo));
connectionInfo.Size = sizeof(BTP_Connection_Info_Ex_t);
connectionInfo.ProfileType = BTP_PROFILE_SPP;
CString str,stradd,strname;

HRESULT result;
BTP_Device_Find          deviceFind;
BTP_Device_Info_t        deviceInfo;
BTP_Device_Query_Ex_t    deviceQuery;
memset(&deviceQuery, 0, sizeof(deviceQuery));
deviceQuery.Size = sizeof(BTP_Device_Query_Ex_t);

```



```

/* Looking for all devices either discovered or remembered */
deviceQuery.DeviceAttributes = BTP_DEVICE_ALL;
deviceQuery.DeviceType = (connectionInfo.ProfileType == BTP_PROFILE_SPP ?
bdAll : bdHID);

/* Tells the API to perform a ten second GAP inquiry, using zero (0) */

/* here will just return previously discovered devices. */
deviceQuery.InquiryTimeout = 10;

BeginWaitCursor();

result = BTPFindFirstDeviceEx(&deviceFind, &deviceInfo, &deviceQuery);

if (BTP_ERROR_SUCCESS == result)
{
    EndWaitCursor();
    do
    {
        BTP_Device_Info_t deviceInfo_temp;
        memcpy(&deviceInfo_temp, &deviceInfo, sizeof(BTP_Device_Info_t));
        m_Deviceitem.AddTail(deviceInfo_temp);
        result = BTPFindNextDevice(deviceFind, &deviceInfo);
    } while (BTP_ERROR_SUCCESS == result);
}

BTPFindDeviceClose(deviceFind);

int lm_count = m_Deviceitem.GetCount();
int i = 0;

for(i = lm_count-1; i >= 0; i--)
{
    deviceInfo = m_Deviceitem.GetAt(m_Deviceitem.FindIndex(i));
    stradd.Format(L"%02X:%02X:%02X:%02X:%02X:%02X]Device: %d %s ",

    deviceInfo.BD_ADDR.BD_ADDR5,
    deviceInfo.BD_ADDR.BD_ADDR4,
    deviceInfo.BD_ADDR.BD_ADDR3,
    deviceInfo.BD_ADDR.BD_ADDR2,
    deviceInfo.BD_ADDR.BD_ADDR1,
    deviceInfo.BD_ADDR.BD_ADDR0,
    deviceInfo.DeviceAttributes,
    CString(deviceInfo.Name));
}

```

4.1.1.12 BTPFindLocalDevice

This function will return information about the local device. This will include the Bluetooth address, the friendly name, and the class of device.

HRESULT BTPFindLocalDevice(BTP_Device_Info_t *DeviceInfo)

Parameters

DeviceInfo

Will receive information defining the local device.

Return Values

BTP_ERROR_SUCCESS if successful.

Error codes include the following values :

BTP_ERROR_INVALID_PARAMETER
BTP_ERROR_MSG_SEND

C++

```
#include "CustomerDLL.h"
HRESULT result;
DWORD classOfDevice;
CString str;
BTP_Find_Local_Device_From_BTE_t device;
result = BTPFindLocalDevice(&device);
if (BTP_ERROR_SUCCESS == result)
{
    classOfDevice = (((unsigned int)
device.DeviceInfo.ClassOfDevice.Class_of_Device0) << 16) |
(((unsigned int)device.DeviceInfo.ClassOfDevice.Class_of_Device1) << 8)
((unsigned int)device.DeviceInfo.ClassOfDevice.Class_of_Device2);

    str.Format(L"\tFriendly Name: %s\n\tBD_ADDR:
[%02X:%02X:%02X:%02X:%02X:%02X]\n\tClass of Device: %u (0x%02X%02X%02X)\n",
CString(device.DeviceInfo.Name),
device.DeviceInfo.BD_ADDR.BD_ADDR5,
device.DeviceInfo.BD_ADDR.BD_ADDR4,
device.DeviceInfo.BD_ADDR.BD_ADDR3,
device.DeviceInfo.BD_ADDR.BD_ADDR2,
device.DeviceInfo.BD_ADDR.BD_ADDR1,
device.DeviceInfo.BD_ADDR.BD_ADDR0,
classOfDevice,
device.DeviceInfo.ClassOfDevice.Class_of_Device0,
device.DeviceInfo.ClassOfDevice.Class_of_Device1,
device.DeviceInfo.ClassOfDevice.Class_of_Device2);
    AfxMessageBox(str);
}
```

4.1.1.13 BTPFindFirstService

This function performs an SDP service discovery on the specified device, return the first service from the resulting list.

```
HRESULT BTPFindFirstService (BTP_Service_Find *ServiceFind,
    BTP_Service_Info t *ServiceInfo, const BTP_Service_Query t *ServiceQuery)
```

Parameters

ServiceFind

Handle to the list of remote services, needed for subsequent calls to BTPFindNextService and BTPFindServiceClose.

ServiceInfo

Information defining the first remote service in the list.

ServiceQuery

Provides parameters for the SDP query.

Return Values

BTP_ERROR_SUCCESS if successful.

Error codes include the following values :

BTP_ERROR
BTP_ERROR_NO_MORE
BTP_ERROR_INVALID_HANDLE
BTP_ERROR_MSG_TOO_BIG

C++

```

#include "CustomerDLL.h"

// Service Find
m_ctrllist.DeleteAllItems();

unsigned                                serviceUUID= 16;
HRESULT result;
BTP_Service_Find                       serviceFind;
BTP_Service_Info_Ex_t                 serviceInfo;
BTP_Service_Query_t                   serviceQuery;
SDP_UUID_Entry_t                       service[1];
CString                                strname;

serviceInfo.Size = sizeof(BTP_Service_Info_Ex_t);

memset(&serviceQuery, 0, sizeof(serviceQuery));

/* Set the BD ADDR of the device that we want to query */
serviceQuery.BD_ADDR = connectionInfo.BD_ADDR;
serviceQuery.NumberServiceUUID = 1;
serviceQuery.Service = service;

/* Normally, the service UUID will be a 16-bit UUID. There is only one */
/* custom case that we have encountered where a 128-bit UUID has been */
/* used for a custom SPP service. For this reason, we will only check */
/* for the special 128-bit case and assume all others are a 16-bit UUID.*/
service[0].SDP_Data_Element_Type = deUUID_16;
switch(connectionInfo.ProfileType)
{
    case BTP_PROFILE_SPP:
        if (128==serviceUUID)
        {
            service[0].SDP_Data_Element_Type = deUUID_128;
            /* This is the special case of a custom 128-bit UUID SPP. */
            ASSIGN_UUID_128((service[0].UUID_Value.UUID_128),
                0x85, 0x60, 0xCA, 0x18,          // 0x8560CA18
                0x62, 0x3F,                      // 0x623f
                0x4A, 0x77,                      // 0x4a77
                0x9F, 0x5E, 0x3C, 0x52, 0x66, 0x68, 0x05, 0x1A); // 0x9f, 0x5e, 0x3c,
                0x52, 0x66, 0x68, 0x05, 0x1a
        }
        else
        {
            /* We will only request services that support SPP. */
            /* FYI: The 16-bit UUID for SPP is 0x1101. Reference the Bluetooth*/
            /* spec for additional UUIDs at http://www.bluetooth.org */
            ASSIGN_UUID_16((service[0].UUID_Value.UUID_16), 0x11, 0x01)
        }
        break;

    case BTP_PROFILE_HID_HOST:
    case BTP_PROFILE_HID_DEVICE:
        /* We will only request services that support HID. */
        /* FYI: The 16-bit UUID for HID is 0x1124. Reference the Bluetooth */
        /* spec for additional UUIDs at http://www.bluetooth.org */
        ASSIGN_UUID_16((service[0].UUID_Value.UUID_16), 0x11, 0x24)
        break;
}

BeginWaitCursor();
/* Search for the service */
result = BTPFindFirstServiceEx(&serviceFind, &serviceInfo, &serviceQuery);
if (BTP_ERROR_SUCCESS == result)
{

```

```

EndWaitCursor();
do
{
    /* Is this the service to which we want to connect? For this ex- */
    /* ample, we will limit our search to any service that at least */
    /* begins with the string passed in as the service name. */
    /* We found a service that is close to what we are seeking */
    /* Populate the Connection Info structure for this service. */
    connectionInfo.ProfileType = serviceInfo.ProfileType;
    connectionInfo.MajorVersion = serviceInfo.MajorVersion;
    connectionInfo.MinorVersion = serviceInfo.MinorVersion;
    switch(connectionInfo.ProfileType)
    {
        case BTP_PROFILE_UNKNOWN:
            /* Treat unknown profiles as RFCOMM based (SPP) profiles */
            connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.
RFCOMMServerPort =
serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.
RFCOMMServerPort;

            connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.
UseActiveSync =
serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.
UseActiveSync;

            break;

            case BTP_PROFILE_SPP:
                connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.
RFCOMMServerPort =
serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.
RFCOMMServerPort;

                connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.
UseActiveSync =
serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.
UseActiveSync;

                strname.Format(L"%s\n",CString(serviceInfo.ServiceName));

                m_ctrllist.InsertItem(0, strname,0);

                break;

                case BTP_PROFILE_HID_HOST:

                case BTP_PROFILE_HID_DEVICE:

                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
DeviceAutomaticReconnect =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
DeviceAutomaticReconnect;

                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
DeviceNormallyConnectable =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
DeviceNormallyConnectable;

                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
DeviceSubclass = serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
DeviceSubclass;

                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
L2CAPControlChannel =

```

```

serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
    L2CAPControlChannel;

    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
    L2CAPInterruptChannel =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
    L2CAPInterruptChannel;

    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
    VirtualCableSupported =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
    VirtualCableSupported;

    break;

    result = BTPFindNextServiceEx(serviceFind, &serviceInfo);
} while (BTP_ERROR_SUCCESS == result);

if (BTP_ERROR_NO_MORE != result)
    fprintf(stderr, "%s - ERROR (line %d): Failed to find next service.
Result = %d\n", __FUNCTION__, __LINE__, result);
    BTPFindServiceClose(serviceFind);
}

```

4.1.1.14 BTPFindFirstServiceEx

This function performs an SDP service discovery on the specified devices, return the first service from the resulting list.

```

HRESULT BTPFindFirstServiceEx (BTP_Service_Find *ServiceFind,
    BTP_Service_Info_Ex_t *ServiceInfoEx,
    const BTP_Service_Query_t *ServiceQuery)

```

Parameters

ServiceFind

Handle to the list of remote services, needed for subsequent calls to BTPFindNextServiceEx and BTPFindServiceClose.

ServiceInfoEx

Information defining the first remote service in the list. This currently supports SPP and HID services.

ServiceQuery

Provides parameters for the SDP query. This currently supports SPP and HID searching by specifying their UUIDs.

Return Values

BTP_ERROR_SUCCESS if successful.

Error codes include the following values :

```

BTP_ERROR
BTP_ERROR_NO_MORE
BTP_ERROR_INVALID_HANDLE
BTP_ERROR_MSG_TOO_BIG

```

C++

```

#include "CustomerDLL.h"

// Service Find
m_ctrllist.DeleteAllItems();

unsigned                serviceUUID= 16;
HRESULT result;

```

```

BTP_Service_Find          serviceFind;
BTP_Service_Info_Ex_t     serviceInfo;
BTP_Service_Query_t       serviceQuery;
SDP_UUID_Entry_t          service[1];
CString                   strname;

serviceInfo.Size = sizeof(BTP_Service_Info_Ex_t);

memset(&serviceQuery, 0, sizeof(serviceQuery));

/* Set the BD ADDR of the device that we want to query */
serviceQuery.BD_ADDR = connectionInfo.BD_ADDR;
serviceQuery.NumberServiceUUID = 1;
serviceQuery.Service = service;

/* Normally, the service UUID will be a 16-bit UUID. There is only one */
/* custom case that we have encountered where a 128-bit UUID has been */
/* used for a custom SPP service. For this reason, we will only check */
/* for the special 128-bit case and assume all others are a 16-bit UUID.*/
service[0].SDP_Data_Element_Type = deUUID_16;
switch(connectionInfo.ProfileType)
{
    case BTP_PROFILE_SPP:
        if (128==serviceUUID)
        {
            service[0].SDP_Data_Element_Type = deUUID_128;
            /* This is the special case of a custom 128-bit UUID SPP. */
            ASSIGN_UUID_128((service[0].UUID_Value.UUID_128),
                0x85, 0x60, 0xCA, 0x18,          // 0x8560CA18
                0x62, 0x3F,                      // 0x623f
                0x4A, 0x77,                      // 0x4a77
                0x9F, 0x5E, 0x3C, 0x52, 0x66, 0x68, 0x05, 0x1A); // 0x9f, 0x5e, 0x3c,
                0x52, 0x66, 0x68, 0x05, 0x1a
        }
        else
        {
            /* We will only request services that support SPP. */
            /* FYI: The 16-bit UUID for SPP is 0x1101. Reference the Bluetooth */
            /* spec for additional UUIDs at http://www.bluetooth.org */
            ASSIGN_UUID_16((service[0].UUID_Value.UUID_16), 0x11, 0x01)
        }
        break;
    case BTP_PROFILE_HID_HOST:
    case BTP_PROFILE_HID_DEVICE:
        /* We will only request services that support HID. */
        /* FYI: The 16-bit UUID for HID is 0x1124. Reference the Bluetooth */
        /* spec for additional UUIDs at http://www.bluetooth.org */
        ASSIGN_UUID_16((service[0].UUID_Value.UUID_16), 0x11, 0x24)
        break;
}

BeginWaitCursor();
/* Search for the service */
result = BTPFindFirstServiceEx(&serviceFind, &serviceInfo, &serviceQuery);
if (BTP_ERROR_SUCCESS == result)
{
    EndWaitCursor();
    do
    {
        /* Is this the service to which we want to connect? For this ex- */
        /* ample, we will limit our search to any service that at least */
        /* begins with the string passed in as the service name. */

        /* We found a service that is close to what we are seeking */

```

```

/* Populate the Connection Info structure for this service. */
connectionInfo.ProfileType = serviceInfo.ProfileType;
connectionInfo.MajorVersion = serviceInfo.MajorVersion;
connectionInfo.MinorVersion = serviceInfo.MinorVersion;
switch (connectionInfo.ProfileType)
{
    case BTP_PROFILE_UNKNOWN:
        /* Treat unknown profiles as RFCOMM based (SPP) profiles */
        connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.
RFCOMMServerPort =
serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.
RFCOMMServerPort;

        connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.
UseActiveSync =
serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.
UseActiveSync;

        break;

        case BTP_PROFILE_SPP:
            connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.
RFCOMMServerPort =
serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.
RFCOMMServerPort;

            connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.
UseActiveSync =
serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.
UseActiveSync;

            strname.Format(L"%s\n", CString(serviceInfo.ServiceName));

            m_ctrllist.InsertItem(0, strname, 0);
            break;

            case BTP_PROFILE_HID_HOST:

                case BTP_PROFILE_HID_DEVICE:
                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
DeviceAutomaticReconnect =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
DeviceAutomaticReconnect;

                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
DeviceNormallyConnectable =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
DeviceNormallyConnectable

                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
DeviceSubclass = serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
DeviceSubclass;

                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
L2CAPControlChannel =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
L2CAPControlChannel;

                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
L2CAPInterruptChannel =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
L2CAPInterruptChannel;

                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.

```

```

        VirtualCableSupported =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
        VirtualCableSupported;

        break;
    }
    result = BTPFindNextServiceEx(serviceFind, &serviceInfo);
} while (BTP_ERROR_SUCCESS == result);

if (BTP_ERROR_NO_MORE != result)
    fprintf(stderr, "%s - ERROR (line %d): Failed to find next service.
Result = %d\n", __FUNCTION__, __LINE__, result);

BTPFindServiceClose(serviceFind);
}

```

4.1.1.15 BTPFindNextService

This function returns the next service in the list of services created by a previous call to **BTPFindFirstService**.

```

HRESULT BTPFindNextService (BTP_Service_Find ServiceFind,
        BTP_Service_Info_t *ServiceInfo)

```

Parameters

ServiceFind

Handle to the list of remote services, originally returned by a call to BTPFindFirstService.

ServiceInfo

Information defining a remote service from the list.

Return Values

BTP_ERROR_SUCCESS if successful.

Error codes include the following values.

BTP_ERROR

BTP_ERROR_NO_MORE

BTP_ERROR_INVALID_HANDLE

C++

```

#include "CustomerDLL.h"

// Service Find
m ctrllist.DeleteAllItems();

unsigned                serviceUUID= 16;
HRESULT result;
BTP_Service_Find        serviceFind;
BTP_Service_Info_Ex_t   serviceInfo;
BTP_Service_Query_t     serviceQuery;
SDP_UUID_Entry_t        service[1];
CString                 strname;

serviceInfo.Size = sizeof(BTP_Service_Info_Ex_t);

memset(&serviceQuery, 0, sizeof(serviceQuery));

/* Set the BD ADDR of the device that we want to query */
serviceQuery.BD_ADDR = connectionInfo.BD_ADDR;
serviceQuery.NumberServiceUUID = 1;
serviceQuery.Service = service;

```



```

/* Normally, the service UUID will be a 16-bit UUID. There is only one */
/* custom case that we have encountered where a 128-bit UUID has been */
/* used for a custom SPP service. For this reason, we will only check */
/* for the special 128-bit case and assume all others are a 16-bit UUID.*/
service[0].SDP_Data_Element_Type = deUUID_16;
switch(connectionInfo.ProfileType)
{
    case BTP_PROFILE_SPP:
        if (128==serviceUUID)
        {
            service[0].SDP_Data_Element_Type = deUUID_128;
            /* This is the special case of a custom 128-bit UUID SPP. */
            ASSIGN_UUID_128((service[0].UUID_Value.UUID_128),
                0x85, 0x60, 0xCA, 0x18,    // 0x8560CA18
                0x62, 0x3F,              // 0x623f
                0x4A, 0x77,              // 0x4a77
                0x9F, 0x5E, 0x3C, 0x52, 0x66, 0x68, 0x05, 0x1A); // 0x9f, 0x5e, 0x3c,
                0x52, 0x66, 0x68, 0x05, 0x1a
        }
        else
        {
            /* We will only request services that support SPP. */
            /* FYI: The 16-bit UUID for SPP is 0x1101. Reference the Bluetooth*/
            /* spec for additional UUIDs at http://www.bluetooth.org */
            ASSIGN_UUID_16((service[0].UUID_Value.UUID_16), 0x11, 0x01)
        }
        break;

    case BTP_PROFILE_HID_HOST:

    case BTP_PROFILE_HID_DEVICE:
        /* We will only request services that support HID. */
        /* FYI: The 16-bit UUID for HID is 0x1124. Reference the Bluetooth */
        /* spec for additional UUIDs at http://www.bluetooth.org */
        ASSIGN_UUID_16((service[0].UUID_Value.UUID_16), 0x11, 0x24)
        break;
}

BeginWaitCursor();
/* Search for the service */
result = BTPFindFirstServiceEx(&serviceFind, &serviceInfo, &serviceQuery);
if (BTP_ERROR_SUCCESS == result)
{
    EndWaitCursor();
    do
    {
        /* Is this the service to which we want to connect? For this ex- */
        /* ample, we will limit our search to any service that at least */
        /* begins with the string passed in as the service name. */

        /* We found a service that is close to what we are seeking */
        /* Populate the Connection Info structure for this service. */
        connectionInfo.ProfileType = serviceInfo.ProfileType;
        connectionInfo.MajorVersion = serviceInfo.MajorVersion;
        connectionInfo.MinorVersion = serviceInfo.MinorVersion;
        switch(connectionInfo.ProfileType)
        {
            case BTP_PROFILE_UNKNOWN:
                /* Treat unknown profiles as RFCOMM based (SPP) profiles */
                connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.
                RFCOMMServerPort =
                    serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.
                    RFCOMMServerPort;

```

```

        connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.
        UseActiveSync =
serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.
        UseActiveSync;

        break;

        case BTP_PROFILE_SPP:
            connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.
            RFCOMMServerPort =
serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.
            RFCOMMServerPort;

            connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.
            UseActiveSync =
serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.
            UseActiveSync;

            strname.Format(L"%s\n",CString(serviceInfo.ServiceName));

            m_ctrllist.InsertItem(0, strname,0);
            break;

            case BTP_PROFILE_HID_HOST:

            case BTP_PROFILE_HID_DEVICE:

                connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
                DeviceAutomaticReconnect =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
                DeviceAutomaticReconnect;

                connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
                DeviceNormallyConnectable=
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
                DeviceNormallyConnectable;

                connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
                DeviceSubclass = serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
                DeviceSubclass;

                connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
                L2CAPControlChannel =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
                L2CAPControlChannel;

                connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
                L2CAPInterruptChannel =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
                L2CAPInterruptChannel;

                connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
                VirtualCableSupported =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
                VirtualCableSupported;

            break;

            result = BTPFindNextServiceEx(serviceFind, &serviceInfo);
        } while (BTP_ERROR_SUCCESS == result);

        if (BTP_ERROR_NO_MORE != result)
            fprintf(stderr, "%s - ERROR (line %d): Failed to find next service.
Result = %d\n",    FUNCTION    ,    LINE    , result);

```

```

    BTPFindServiceClose(serviceFind);
}

```

4.1.1.16 BTPFindNextServiceEx

This function returns the next service in the list of services created by a previous call to **BTPFindFirstService**.

```

HRESULT BTPFindNextServiceEx (BTP_Service_Find ServiceFind,
    BTP_Service_Info_Ext *ServiceInfoEx)

```

Parameters

ServiceFind

Handle to the list of remote services, originally returned by a call to BTPFindFirstService.

ServiceInfoEx

Information defining the first remote service in the list. This currently supports SPP and HID services.

Return Values

BTP_ERROR_SUCCESS if successful.

Error codes include the following values :

BTP_ERROR

BTP_ERROR_NO_MORE

BTP_ERROR_INVALID_HANDLE

C++

```

#include "CustomerDLL.h"

// Service Find
m_ctrllist.DeleteAllItems();

unsigned                serviceUUID= 16;
HRESULT result;
BTP_Service_Find        serviceFind;
BTP_Service_Info_Ext_t  serviceInfo;
BTP_Service_Query_t     serviceQuery;
SDP_UUID_Entry_t        service[1];
CString                 strname;

serviceInfo.Size = sizeof(BTP_Service_Info_Ext_t);

memset(&serviceQuery, 0, sizeof(serviceQuery));

/* Set the BD_ADDR of the device that we want to query */
serviceQuery.BD_ADDR = connectionInfo.BD_ADDR;
serviceQuery.NumberServiceUUID = 1;
serviceQuery.Service = service;

/* Normally, the service UUID will be a 16-bit UUID. There is only one */
/* custom case that we have encountered where a 128-bit UUID has been */
/* used for a custom SPP service. For this reason, we will only check */
/* for the special 128-bit case and assume all others are a 16-bit UUID.*/
service[0].SDP_Data_Element_Type = deUUID_16;
switch(connectionInfo.ProfileType)
{
    case BTP_PROFILE_SPP:
        if (128==serviceUUID)
        {
            service[0].SDP_Data_Element_Type = deUUID_128;
            /* This is the special case of a custom 128-bit UUID SPP. */

```

```

        ASSIGN_UUID_128((service[0].UUID_Value.UUID_128),
        0x85, 0x60, 0xCA, 0x18,          // 0x8560CA18
        0x62, 0x3F,                      // 0x623f
        0x4A, 0x77,                      // 0x4a77
        0x9F, 0x5E, 0x3C, 0x52, 0x66, 0x68, 0x05, 0x1A); // 0x9f, 0x5e, 0x3c,
        0x52, 0x66, 0x68, 0x05, 0x1a
    }
    else
    {
        /* We will only request services that support SPP. */
        /* FYI: The 16-bit UUID for SPP is 0x1101. Reference the Bluetooth*/
        /* spec for additional UUIDs at http://www.bluetooth.org */
        ASSIGN_UUID_16((service[0].UUID_Value.UUID_16), 0x11, 0x01)
    }
    break;

case BTP_PROFILE_HID_HOST:

case BTP_PROFILE_HID_DEVICE:
    /* We will only request services that support HID. */
    /* FYI: The 16-bit UUID for HID is 0x1124. Reference the Bluetooth */
    /* spec for additional UUIDs at http://www.bluetooth.org */
    ASSIGN_UUID_16((service[0].UUID_Value.UUID_16), 0x11, 0x24)
    break;
}

BeginWaitCursor();
/* Search for the service */
result = BTPFindFirstServiceEx(&serviceFind, &serviceInfo, &serviceQuery);
if (BTP_ERROR_SUCCESS == result)
{
    EndWaitCursor();
    do
    {
        /* Is this the service to which we want to connect? For this ex- */
        /* ample, we will limit our search to any service that at least */
        /* begins with the string passed in as the service name. */

        /* We found a service that is close to what we are seeking */
        /* Populate the Connection Info structure for this service. */
        connectionInfo.ProfileType = serviceInfo.ProfileType;
        connectionInfo.MajorVersion = serviceInfo.MajorVersion;
        connectionInfo.MinorVersion = serviceInfo.MinorVersion;
        switch(connectionInfo.ProfileType)
        {
            case BTP_PROFILE_UNKNOWN:
                /* Treat unknown profiles as RFCOMM based (SPP) profiles */
                connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.
                RFCOMMServerPort =
serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.
                RFCOMMServerPort;

                connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.
                UseActiveSync =
serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.
                UseActiveSync;

                break;

            case BTP_PROFILE_SPP:
                connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.
                RFCOMMServerPort =
serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.
                RFCOMMServerPort;

```

```

        connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.
        UseActiveSync =
serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.
        UseActiveSync;

        strname.Format(L"%s\n",CString(serviceInfo.ServiceName));

        m_ctrllist.InsertItem(0, strname,0);
        break;

        case BTP_PROFILE_HID_HOST:

            case BTP_PROFILE_HID_DEVICE:
                connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
                DeviceAutomaticReconnect =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
                DeviceAutomaticReconnect;

                connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
                DeviceNormallyConnectable =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
                DeviceNormallyConnectable;

                connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
                DeviceSubclass = serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
                DeviceSubclass;

                connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
                L2CAPControlChannel =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
                L2CAPControlChannel;

                connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
                L2CAPInterruptChannel =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
                L2CAPInterruptChannel;

                connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
                VirtualCableSupported =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
                VirtualCableSupported;

            break;
        }
        result = BTPFindNextServiceEx(serviceFind, &serviceInfo);
    } while (BTP_ERROR_SUCCESS == result);

    if (BTP_ERROR_NO_MORE != result)
        fprintf(stderr, "%s - ERROR (line %d): Failed to find next service.
Result = %d\n", __FUNCTION__, __LINE__, result);

    BTPFindServiceClose(serviceFind);
}

```

4.1.1.17 BTPDeleteConnection

This function discards a previously defined connection. After deleting a connection, its connection ID is no longer valid.

HRESULT BTPDeleteConnection (BTP Connection ID ConnectionID)

Parameters*ConnectionID*

Unique identifier for a connection which was previously defined by a call to `BTPCreateConnection` and `BTPCreateConnectionEx`.

Return Values

`BTP_ERROR_SUCCESS` if successful.

Error codes include the following values :

`BTP_ERROR`

`BTP_ERROR_INVALID_PARAMETER`

C++

```
#include "CustomerDLL.h"

// Delete Connection

BTP_Connection_Info_Ex_t connectionInfo;
HRESULT result;
/* Delete the favorite*/
result = BTPDeleteConnection(connectionInfo.ConnectionID);
if (BTP_ERROR_SUCCESS != result)
    SetWindowText(L"delete connection");
else
    AfxMessageBox(L"delete Connection_error");
```

4.1.1.18 BTPDisconnect

This function disconnects from an active connection. If the connection is not persistent, the connection is also deleted, invalidating its connection ID.

`HRESULT BTPDisconnect (BTP_Connection_ID ConnectionID)`

Parameters*ConnectionID*

Unique identifier for a connection which was previously defined by a call to `BTPCreateConnnection` and `BTPCreateConnectionEx`.

Return Values

`BTP_ERROR_SUCCESS` if successful.

Error codes include the following values :

`BTP_ERROR`

`BTP_ERROR_INVALID_PARAMETER`

C++

```
#include "CustomerDLL.h"

// Delete Connection

BTP_Connection_Info_Ex_t connectionInfo;
HRESULT result;

result = BTPDisconnect(connectionInfo.ConnectionID);

if(result == BTP_ERROR_SUCCESS)
    SetWindowText(L"Disconnect");
else if(result == BTP_ERROR_INVALID_PARAMETER)
    AfxMessageBox(L"ERROR_INVALID_PARAMETER");
else if(result == BTP_ERROR)
    AfxMessageBox(L"error");
```

4.1.1.19 BTPFindConnectionClose

This function deletes the remote list of connections and performs any additional cleanup.

HRESULT BTPFindConnectionClose (BTP_Connection_Find ConnectionFind)

Parameters

ConnectionFind

Handle to the list connections, originally returned by a call to BTPFindFirstConnection or BTPFindFirstConnectionEx.

Return Values

BTP_ERROR_SUCCESS if successful.

Error codes include the following values :

BTP_ERROR

BTP_ERROR_INVALID_HANDLE

C++

```
#include "CustomerDLL.h"

// Find Connection
m_ctrllist.DeleteAllItems();
m_Connectitem.RemoveAll();

m_listtype = 3;           //list type
HRESULT result;
CString stradd;
BTP_Connection_Find      connectFind;
BTP_Connection_Info_t    connectioinfo;
BTP_Connection_Query_t   connectQuery;

memset(&connectQuery, 0, sizeof(connectQuery));

connectQuery.ConnectionAttributes = BTP_CONNECTION_REMEMBERED;

BeginWaitCursor();

result = BTPFindFirstConnection(&connectFind, &connectioinfo, &connectQuery);
if (BTP_ERROR_SUCCESS == result)
{
    EndWaitCursor();
    do
    {
        BTP_Connection_Info_t  connectInfo_temp;
        memcpy(&connectInfo_temp, &connectioinfo, sizeof(BTP_Connection_Info_t));

        m_Connectitem.AddTail(connectInfo_temp);

        result = BTPFindNextConnection(connectFind, &connectioinfo);

    } while (BTP_ERROR_SUCCESS == result);
}
BTPFindConnectionClose(connectFind);

int lm_count = m_Connectitem.GetCount();
int i = 0;

for(i = lm_count-1; i>=0; i--)
{
```

```

        connectioinfo = m_Connectitem.GetAt(m_Connectitem.FindIndex(i));

        stradd.Format(L "[%02X:%02X:%02X:%02X:%02X:%02X] id:%d",
        connectioinfo.BD_ADDR.BD_ADDR5,
        connectioinfo.BD_ADDR.BD_ADDR4,
        connectioinfo.BD_ADDR.BD_ADDR3,
        connectioinfo.BD_ADDR.BD_ADDR2,
        connectioinfo.BD_ADDR.BD_ADDR1,
        connectioinfo.BD_ADDR.BD_ADDR0,
        connectioinfo.ConnectionID);

        m_ctrllist.InsertItem(0, stradd,0);
    }

```

4.1.1.20 BTPFindDeviceClose

This function deletes the remote list of devices and performs any additional cleanup.

HRESULT BTPFindDeviceClose (BTP_Device_Find DeviceFind)

Parameters

DeviceFind

Handle to the list of discovered devices, originally returned by a call to BTPFindFirstDevice.

Return Values

BTP_ERROR_SUCCESS if successful.

Error codes include the following values.

BTP_ERROR

BTP_ERROR_INVALID_HANDLE

C++

```

#include "CustomerDLL.h"

// Device Find
m_listtype = 1;                                //list type
m_ctrllist.DeleteAllItems();
m_Deviceitem.RemoveAll();
memset(&connectionInfo, 0, sizeof(connectionInfo));
connectionInfo.Size = sizeof(BTP_Connection_Info_Ex_t);
connectionInfo.ProfileType = BTP_PROFILE_SPP;
CString str,stradd,strname;

HRESULT result;
BTP_Device_Find          deviceFind;
BTP_Device_Info_t        deviceInfo;
BTP_Device_Query_Ex_t    deviceQuery;
memset(&deviceQuery, 0, sizeof(deviceQuery));
deviceQuery.Size = sizeof(BTP_Device_Query_Ex_t);

/* Looking for all devices either discovered or remembered */
deviceQuery.DeviceAttributes = BTP_DEVICE_ALL;
deviceQuery.DeviceType = (connectionInfo.ProfileType == BTP_PROFILE_SPP ?
bdAll : bdHID);

/* Tells the API to perform a ten second GAP inquiry, using zero (0) */

/* here will just return previously discovered devices. */
deviceQuery.InquiryTimeout = 10;

BeginWaitCursor();

```



```

result = BTPFindFirstDeviceEx(&deviceFind, &deviceInfo, &deviceQuery);

if (BTP_ERROR_SUCCESS == result)
{
    EndWaitCursor();
    do
    {
        BTP_Device_Info_t deviceInfo_temp;
        memcpy(&deviceInfo_temp, &deviceInfo, sizeof(BTP_Device_Info_t));

        m_Deviceitem.AddTail(deviceInfo_temp);
        result = BTPFindNextDevice(deviceFind, &deviceInfo);
    } while (BTP_ERROR_SUCCESS == result);
}

BTPFindDeviceClose(deviceFind);

int lm_count = m_Deviceitem.GetCount();
int i = 0;

for(i = lm_count-1; i >= 0; i--)
{
    deviceInfo = m_Deviceitem.GetAt(m_Deviceitem.FindIndex(i));
    stradd.Format(L "[%02X:%02X:%02X:%02X:%02X:%02X]Device: %d %s ",

    deviceInfo.BD_ADDR.BD_ADDR5,
    deviceInfo.BD_ADDR.BD_ADDR4,
    deviceInfo.BD_ADDR.BD_ADDR3,
    deviceInfo.BD_ADDR.BD_ADDR2,
    deviceInfo.BD_ADDR.BD_ADDR1,
    deviceInfo.BD_ADDR.BD_ADDR0,
    deviceInfo.DeviceAttributes,
    CString(deviceInfo.Name));
}

```

4.1.1.21 BTPFindServiceClose

This function deletes the remote list of services and performs any additional cleanup.

HRESULT BTPFindServiceClose (BTP Service Find ServiceFind)

Parameters

ServiceFind

Handle to the list of remote services, originally returned by a call to BTPFindFirstService or BTPFindFirstServiceEx.

Return Values

BTP_ERROR_SUCCESS if successful.

Error codes include the following values :

BTP_ERROR

BTP_ERROR_INVALID_HANDLE

C++

```

#include "CustomerDLL.h"

// Service Find
m_ctrllist.DeleteAllItems();

unsigned                serviceUUID= 16;
HRESULT result;
BTP_Service_Find        serviceFind;

```

```

BTP_Service_Info_Ex_t  serviceInfo;
BTP_Service_Query_t    serviceQuery;
SDP_UUID_Entry_t      service[1];
CString                strname;

serviceInfo.Size = sizeof(BTP_Service_Info_Ex_t);

memset(&serviceQuery, 0, sizeof(serviceQuery));

/* Set the BD ADDR of the device that we want to query */
serviceQuery.BD_ADDR = connectionInfo.BD_ADDR;
serviceQuery.NumberServiceUUID = 1;
serviceQuery.Service = service;

/* Normally, the service UUID will be a 16-bit UUID. There is only one */
/* custom case that we have encountered where a 128-bit UUID has been */
/* used for a custom SPP service. For this reason, we will only check */
/* for the special 128-bit case and assume all others are a 16-bit UUID.*/
service[0].SDP_Data_Element_Type = deUUID_16;
switch(connectionInfo.ProfileType)
{
    case BTP_PROFILE_SPP:
        if (128==serviceUUID)
        {
            service[0].SDP_Data_Element_Type = deUUID_128;
            /* This is the special case of a custom 128-bit UUID SPP. */
            ASSIGN_UUID_128((service[0].UUID_Value.UUID_128),
                0x85, 0x60, 0xCA, 0x18,          // 0x8560CA18
                0x62, 0x3F,                      // 0x623f
                0x4A, 0x77,                      // 0x4a77
                0x9F, 0x5E, 0x3C, 0x52, 0x66, 0x68, 0x05, 0x1A); // 0x9f, 0x5e, 0x3c,
                0x52, 0x66, 0x68, 0x05, 0x1a
        }
        else
        {
            /* We will only request services that support SPP. */
            /* FYI: The 16-bit UUID for SPP is 0x1101. Reference the Bluetooth */
            /* spec for additional UUIDs at http://www.bluetooth.org */
            ASSIGN_UUID_16((service[0].UUID_Value.UUID_16), 0x11, 0x01)
        }
        break;
    case BTP_PROFILE_HID_HOST:
    case BTP_PROFILE_HID_DEVICE:
        /* We will only request services that support HID. */
        /* FYI: The 16-bit UUID for HID is 0x1124. Reference the Bluetooth */
        /* spec for additional UUIDs at http://www.bluetooth.org */
        ASSIGN_UUID_16((service[0].UUID_Value.UUID_16), 0x11, 0x24)
        break;
}

BeginWaitCursor();
/* Search for the service */
result = BTPFindFirstServiceEx(&serviceFind, &serviceInfo, &serviceQuery);
if (BTP_ERROR_SUCCESS == result)
{
    EndWaitCursor();
    do
    {
        /* Is this the service to which we want to connect? For this ex- */
        /* ample, we will limit our search to any service that at least */
        /* begins with the string passed in as the service name. */

        /* We found a service that is close to what we are seeking */
        /* Populate the Connection Info structure for this service. */

```

```

connectionInfo.ProfileType = serviceInfo.ProfileType;
connectionInfo.MajorVersion = serviceInfo.MajorVersion;
connectionInfo.MinorVersion = serviceInfo.MinorVersion;
switch (connectionInfo.ProfileType)
{
    case BTP_PROFILE_UNKNOWN:
        /* Treat unknown profiles as RFCOMM based (SPP) profiles */
        connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.
RFCOMMServerPort =
serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.
RFCOMMServerPort;

        connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.
UseActiveSync =
serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.
UseActiveSync;

        break;

        case BTP_PROFILE_SPP:
            connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.
RFCOMMServerPort =
serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.
RFCOMMServerPort;

            connectionInfo.ProfileInformation.RemoteSerialPortProfileInfo.
UseActiveSync =
serviceInfo.ProfileInformation.RemoteSerialPortProfileInfo.
UseActiveSync;

            strname.Format(L"%s\n", CString(serviceInfo.ServiceName));

            m_ctrllist.InsertItem(0, strname, 0);
            break;

            case BTP_PROFILE_HID_HOST:

                case BTP_PROFILE_HID_DEVICE:
                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
DeviceAutomaticReconnect =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
DeviceAutomaticReconnect;

                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
DeviceNormallyConnectable =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
DeviceNormallyConnectable;

                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
DeviceSubclass = serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
DeviceSubclass;

                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
L2CAPControlChannel =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
L2CAPControlChannel;

                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
L2CAPInterruptChannel =
serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
L2CAPInterruptChannel;

                    connectionInfo.ProfileInformation.RemoteHIDProfileInfo.
VirtualCableSupported =

```

```

serviceInfo.ProfileInformation.RemoteHIDProfileInfo.
    VirtualCableSupported;

    break;
}
result = BTPFindNextServiceEx(serviceFind, &serviceInfo);
} while (BTP_ERROR_SUCCESS == result);

if (BTP_ERROR_NO_MORE != result)
    fprintf(stderr, "%s - ERROR (line %d): Failed to find next service.
Result = %d\n", __FUNCTION__, __LINE__, result);

BTPFindServiceClose(serviceFind);
}

```

4.1.1.22 BTPCloseAPI

This function is responsible for cleaning up the module after it is no longer needed by the application.

```
void BTPCloseAPI (void)
```

Parameters

None

Return Values

None

C++

```
#include "CustomerDLL.h"
```

```
BTPCloseAPI();
```

4.1.2 Reference and Function List for C#

Structure

_CONNECTINFO

```
public struct _CONNECTINFO
{
    public M3BTEDllNet._TAG_ADD BD_ADDR;
    public ushort ConnectionAttributes;
    public uint ConnectionID;
    public int LocalCOMPort;
    public byte MajorVersion;
    public byte MinorVersion;
    public uint ProfileType;
    public uint RFCOMMServerPort;
    public bool UseActiveSync;
}
```

_LOCALINFO

```
public struct _LOCALINFO
{
    public M3BTEDllNet._TAG_ADD BD_ADDR;
    public M3BTEDllNet._TAG_CLASS_DEVICE ClassOfDevice;
    public ushort DeviceAttributes;
}
```

_TAG_ADD

```
public struct _TAG_ADD
{
    public byte add0;
    public byte add1;
    public byte add2;
    public byte add3;
    public byte add4;
    public byte add5;
}
```

_TAG_CLASS_DEVICE

```
public struct _TAG_CLASS_DEVICE
{
    public byte class0;
    public byte class1;
    public byte class2;
}
```

4.1.2.1 BTPInitAPI

This function initializes the BTE Explorer API module, readying it for use by the application.

```
public bool M3BTPInit();
```

Parameters

None

Return Values

Returns true if successful.

C#

```
using BTEAPIdllNet;

M3BTEDllNet BteDll;

BteDll = new M3BTEDllNet();

if(BteDll.M3BTPInit())
{
    // Init success
}
else
{
    // Init Fail
}
```

4.1.2.2 BTPCreatConnection

This function defines a temporary or persistent (Favorite) connection, which may later be connected by a call to **M3BTPConnect** (Supports SPP and HID).

```
public bool M3BTPCreateConnection(out M3BTEDllNet. CONNECTINFO connectinfo);
```

Parameters

connectioninfo

Information defining the new connection.

Also, the new connection ID is returned in this structure. Currently supports connecting to HID and SPP.

Return Values

Returns true on success, false otherwise.

C#

```
using BTEAPIdllNet;

M3BTEDllNet BteDll;

BteDll = new M3BTEDllNet();

uint nConnectionNum = 0; // First Connection

M3BTEDllNet._CONNECTINFO connectinfo;
byte[] Name_s = new byte[1026];
connectinfo.BD_ADDR = m_localinfo.BD_ADDR;

BteDll.M3BTPGetFindService(nConnectionNum, out connectinfo, Name_s);

SString szstr; szstr = String.Format("FIND service {0:x2} {1:x2} {2:x2} {3:x2} {4:x2} {5:x2} {6:G}", connectinfo.BD_ADDR.add5, connectinfo.BD_ADDR.add4, connectinfo.BD_ADDR.add3, connectinfo.BD_ADDR.add2, connectinfo.BD_ADDR.add1, connectinfo.BD_ADDR.add0, Encoding.Default.GetString(Name_s, 0, 1026));
```

```

MessageBox.Show(szstr);

if (BteDll.M3BTPCreateConnection(out connectinfo))
    MessageBox.Show("create success");
else
    MessageBox.Show("create fail");

```

4.1.2.3 BTPConnect

This function connects to a connection previously defined by a call to **BTPCreateConnection**.

```
public bool M3BTPConnect(uint ConnectionID);
```

Parameters

ConnectionID

Unique identifier for a connection which was previously defined by a call to ExM3BTPGetFindConnection.

Return Values

Returns true on success, false otherwise.

C#

```

using BTEAPIdllNet;

M3BTEDllNet BteDll;

BteDll = new M3BTEDllNet();

uint nConnectionNum = 0;

M3BTEDllNet. CONNECTINFO connectinfo;
BBteDll.M3BTPGetFindConnection(nConnectionNum , out connectinfo);
BteDll.M3BTPConnect(connectinfo.ConnectionID);

```

4.1.2.4 BTPFindConnection

The **BTPFindConnection** finds connections saved in Bluetooth and retrieves the list. This list will be saved in memory and can be called through the **M3BTPGetFindConnection** function.

```
public uint M3BTPFindConnection();
```

Parameters

None

Return Values

Returns the number of Connections currently saved in unit form.

C#

```

using BTEAPIdllNet;

M3BTEDllNet BteDll;

BteDll = new M3BTEDllNet();

int connectioncount = BteDll.M3BTPFindConnection();

for (uint i = 1; i <= connectioncount ; i++)
{
    M3BTEDllNet._CONNECTINFO connectinfo;
    String szstr;

```

```

        BteDll.M3BTPGetFindConnection(i, out connectinfo);
        szstr = String.Format("{0:x2} {1:x2} {2:x2} {3:x2} {4:x2} {5:x2}",
            connectinfo.BD_ADDR.add5,
            connectinfo.BD_ADDR.add4,
            connectinfo.BD_ADDR.add3,
            connectinfo.BD_ADDR.add2,
            connectinfo.BD_ADDR.add1,
            connectinfo.BD_ADDR.add0);

        MessageBox.Show("szstr");
    }

```

4.1.2.5 BTPFindDevice

When the **BTPFindDevice** function is called, the Bluetooth module searches devices within antenna range and the result is saved in memory. The device list can be called through the **M3BTPGetFindDevice** function.

```
public uint M3BTPFindDevice();
```

Parameters

None

Return Values

Returns the number of devices currently saved in unit form.

```

C#
using BTEAPIdllNet;

M3BTEDllNet BteDll;

BteDll = new M3BTEDllNet();

int devicecount = BteDll.M3BTPFindDevice();

String szstr;

ffor (uint i = 1; i <= devicecount ; i++)
{
    M3BTEDllNet._LOCALINFO localinfo;
    byte[] Name = new byte[1026];

    BteDll.M3BTPGetFindDevice(i, out localinfo, Name);
    szstr = String.Format("{0:x2} {1:x2} {2:x2} {3:x2} {4:x2} {5:x2} {6:G}",
        localinfo.BD_ADDR.add5,
        localinfo.BD_ADDR.add4,
        localinfo.BD_ADDR.add3,
        localinfo.BD_ADDR.add2,
        localinfo.BD_ADDR.add1,
        localinfo.BD_ADDR.add0,
        Encoding.Default.GetString(Name, 0, 1026));
    listBox1.Items.Add(szstr);
}

```

4.1.2.6 BTPFindLocalDevice

This function will return information about the local device. This will include the Bluetooth address, the friendly name, and the class of device.

```
public void M3BTPFindLocalDevice(out M3BTEDllNet._LOCALINFO localinfo,
    byte[] name);
```

Parameters

localinfo

Will receive information defining the local device.

name

Retrieve the name of device.

Return Values

None

C#

```
#include "CustomerDLL.h"

M3BTEDllNet._LOCALINFO localinfo;
byte [] Name = new byte[1026];

String szstr;
BteDll.M3BTPFindLocalDevice(out localinfo, Name);
szstr = String.Format("{0:x2} {1:x2} {2:x2} {3:x2} {4:x2} {5:x2} {6:G}",
    localinfo.BD_ADDR.add5,
    localinfo.BD_ADDR.add4,
    localinfo.BD_ADDR.add3,
    localinfo.BD_ADDR.add2,
    localinfo.BD_ADDR.add1,
    localinfo.BD_ADDR.add0,
    Encoding.Default.GetString(Name, 0, 1026));

MessageBox.Show(szstr);
```

4.1.2.7 BTPFindService

The **BTPFindService** searches the service supported by device which is designated as Localinfo. Searched service is saved in memory and the service list can be called through the **M3BTPFindService** function.

```
public uint M3BTPFindService(out M3BTEDllNet._LOCALINFO localinfo);
```

Parameters

localinfo

Structure that contains information of the device to search service list.

Return Values

Returns the number of services currently saved in unit form.

C#

```
using BTEAPIdllNet;

M3BTEDllNet BteDll;

BteDll = new M3BTEDllNet();

String szstr;
byte[] Name_d = new byte[1026];
uint nDeviceNum = 1;

BteDll.M3BTPGetFindDevice(nDeviceNum , out m_localinfo, Name_d);

szstr = String.Format("FIND service {0:x2} {1:x2} {2:x2} {3:x2} {4:x2} {5:x2} {6:G}",
    m_localinfo.BD_ADDR.add5,
    m_localinfo.BD_ADDR.add4,
    m_localinfo.BD_ADDR.add3,
    m_localinfo.BD_ADDR.add2,
```

```

        m_localinfo.BD_ADDR.add1,
        m_localinfo.BD_ADDR.add0,
        Encoding.Default.GetString(Name_d, 0, 1026));

uint nservicecount = BteDll.M3BTPFindService(out m_localinfo);

for (uint i = 1; i <= nservicecount ; i++)
{
    M3BTEDllNet._CONNECTINFO connectinfo;
    byte[] Name_s = new byte[1026];

    BteDll.M3BTPGetFindService(i, out connectinfo, Name_s);
    szstr = String.Format("{0:G}", Encoding.Default.GetString(Name_s, 0, 1026));

    MessageBox.Shwo(szstr);
}

```

4.1.2.8 BTPDeleteConnection

This function discards a previously defined connection. After deleting a connection, its connection ID is no longer valid.

```
public bool M3BTPDeleteConnection(uint ConnectionID);
```

Parameters

ConnectionID

Unique identifier for a connection which was previously defined by a call to M3BTPCreateConnection.

Return Values

Returns true on success, false otherwise.

C#

```

using BTEAPIdllNet;

M3BTEDllNet BteDll;

BteDll = new M3BTEDllNet();

uint nConnectionNum = 0; // First Connection

M3BTEDllNet._CONNECTINFO connectinfo;
BteDll.M3BTPGetFindConnection(nConnectionNum , out connectinfo);

if (BteDll.M3BTPDeleteConnection(connectinfo.ConnectionID))
    MessageBox.Show("delete success");
else
    MessageBox.Show("delete fail");

```

4.1.2.9 BTPDisconnect

This function disconnects from an active connection. If the connection is not persistent, the connection is also deleted, invalidating its connection ID.

```
public bool M3BTPDisconnect(uint DisConnectionID);
```

Parameters

ConnectionID

Unique identifier for a connection which was previously defined by a call to M3BTPCreateConnection.

Return Values

Returns true on success, false otherwise.

C#

```
using BTEAPIdllNet;

M3BTEDllNet BteDll;

BteDll = new M3BTEDllNet();

uint nConnectionNum = 0;

M3BTEDllNet._CONNECTINFO connectinfo;
BBteDll.M3BTPGetFindConnection(nConnectionNum , out connectinfo);

BteDll.M3BTPDisconnect(connectinfo.ConnectionID);
```

4.1.2.10 BTPGetFindConnection

The **BTPGetFindConnection** function gets one of searched connections by the **M3BTPFindConnection** function.

```
public void M3BTPGetFindConnection(uint nSelectNum,
    out M3BTEDllNet._CONNECTINFO connectinfo);
```

Parameters

nSelectNum

Select the number of connections. It should be less than or equal to the total number of connections that M3BTPFindConnection returned.

connectinfo

Get information of selected Connection.

Return Values

None

C#

```
using BTEAPIdllNet;

M3BTEDllNet BteDll;

BteDll = new M3BTEDllNet();

int connectioncount = BteDll.M3BTPFindConnection();

for (uint i = 1; i <= connectioncount ; i++)
{
    M3BTEDllNet._CONNECTINFO connectinfo;
    String szstr;

    BteDll.M3BTPGetFindConnection(i, out connectinfo);
    szstr = String.Format("{0:x2} {1:x2} {2:x2} {3:x2} {4:x2} {5:x2}",
        connectinfo.BD_ADDR.add5,
        connectinfo.BD_ADDR.add4,
        connectinfo.BD_ADDR.add3,
        connectinfo.BD_ADDR.add2,
        connectinfo.BD_ADDR.add1,
        connectinfo.BD_ADDR.add0);

    MessageBox.Show("szstr");
}
```

4.1.2.11 BTPGetFindDevice

BTPGetFindDevice function gets one of the searched devices by the **M3BTPFindDevice** function.

```
public void M3BTPGetFindDevice(uint nSelectNum,  
    out M3BTEDllNet. LOCALINFO localinfo, byte[] name);
```

Parameters

nSelectNum

Select the number of connections. It should be less than or equal to the total number of connections that M3BTPFindConnection returned.

localinfo

Get information of selected Device.

name

Get name of selected Device.

Return Values

None

C#

```
using BTEAPIdllNet;  
  
M3BTEDllNet BteDll;  
  
BteDll = new M3BTEDllNet();  
  
int devicecount = BteDll.M3BTPFindDevice();  
  
String szstr;  
  
for (uint i = 1; i <= devicecount ; i++)  
{  
    M3BTEDllNet. LOCALINFO localinfo;  
    byte[] Name = new byte[1026];  
  
    BteDll.M3BTPGetFindDevice(i, out localinfo, Name);  
    szstr = String.Format("{0:x2} {1:x2} {2:x2} {3:x2} {4:x2} {5:x2} {6:G}",  
        localinfo.BD_ADDR.add5,  
        localinfo.BD_ADDR.add4,  
        localinfo.BD_ADDR.add3,  
        localinfo.BD_ADDR.add2,  
        localinfo.BD_ADDR.add1,  
        localinfo.BD_ADDR.add0,  
        Encoding.Default.GetString(Name, 0, 1026));  
    listBox1.Items.Add(szstr);  
}
```

4.1.2.12 BTPGetFindService

BTPGetFindService function gets the service which is searched by the **M3BTPFindService** function.

```
public void M3BTPGetFindService(uint nSelectNum,  
    out M3BTEDllNet. CONNECTINFO connectinfo, byte[] name);
```

Parameters

nSelectNum

Select the number of connections. It should be less than or equal to the total number of connections that M3BTPFindConnection returned.

connectinfo

Get information of selected Service.

name

Get name of selected Service.

Return Values

None

C#

```
using BTEAPIdllNet;

M3BTEDllNet BteDll;

BteDll = new M3BTEDllNet();

String szstr;
byte[] Name_d = new byte[1026];
uint nDeviceNum = 1;

BteDll.M3BTPGetFindDevice(nDeviceNum , out m_localinfo, Name_d);

szstr = String.Format("FIND service {0:x2} {1:x2} {2:x2} {3:x2} {4:x2} {5:x2} {6:G}",
    m_localinfo.BD_ADDR.add5,
    m_localinfo.BD_ADDR.add4,
    m_localinfo.BD_ADDR.add3,
    m_localinfo.BD_ADDR.add2,
    m_localinfo.BD_ADDR.add1,
    m_localinfo.BD_ADDR.add0,
    Encoding.Default.GetString(Name_d, 0, 1026));

uint nservicecount = BteDll.M3BTPFindService(out m_localinfo);

for (uint i = 1; i <= nservicecount ; i++)
{
    M3BTEDllNet._CONNECTINFO connectinfo;
    byte[] Name_s = new byte[1026];

    BteDll.M3BTPGetFindService(i, out connectinfo, Name_s);
    szstr = String.Format("{0:G}", Encoding.Default.GetString(Name_s, 0, 1026));

    MessageBox.Shwo(szstr);
}
```

4.1.2.13 BTPCloseAPI

This function is responsible for cleaning up the module after it is no longer needed by the application.

bool M3BTPClose()

Parameters

None

Return Values

Returns true on success, false otherwise.

C#

```
using BTEAPIdllNet;

M3BTEDllNet BteDll;

BteDll = new M3BTEDllNet();
```

```
if (BteDll.M3BTPClose())  
    MessageBox.Show("close success");  
else  
    MessageBox.Show("fail");
```

4.1.3 BTE Explorer API Error Codes

The following table lists all possible error codes and a brief description of their meaning.

Error Code	Description
BTP_ERROR_SUCCESS	Indicates the command executed successfully.
BTP_ERROR	Indicates that the command was unsuccessful; used when no other more specific error code is applicable.
BTP_ERROR_INVALID_PARAMETER	One of the parameters is not valid.
BTP_ERROR_INVALID_HANDLE	A handle is not valid.
BTP_ERROR_NO_MORE	Indicates there are no more elements in the list.
BTP_ERROR_MSG_SEND	Indicates sending of the command to BTE Explorer failed.
BTP_ERROR_MSG_RECEIVE	Indicates BTE Explorer failed to respond to the command.
BTP_ERROR_NO_COM_PORT	Indicates there are no available COM ports with which to connect.
BTP_ERROR_BTEXP_NOT_RUNNING	Indicates BTE Explorer could not be started.
BTP_ERROR_NO_KEY_AVAILABLE	Returned by authentication callbacks, indicating the callback cannot provide the pass key for the specified device.

4.2 Camera

This section provides description of the functions and DLLs which are used to manage the camera module.

Required Files

For C++

Required header:

M3SkyCamera.h

Required lib:

M3SkyCamera.lib

Required DLL:

M3SkyCamera.dll

For C#

Required DLL:

M3SkyCamera.dll

Supported Product

M3 ORANGE with camera option.

4.2.1 Reference and Function List for C++

Enum

CAMERA_MODE

```
typedef enum {  
    STILL_MODE = 0,  
    VIDEO_MODE  
} CAMERA_MODE;
```

VIDEO_TYPE

```
typedef enum {  
    VIDEO_ASF = 0,  
    VIDEO_WMV  
} VIDEO_TYPE;
```

4.2.1.1 OpenCamera

This function performs initialization. And switches preview into a running state.

```
BOOL OpenCamera (HWND hWnd, CAMERA_MODE cameramode,
                 VIDEO_TYPE videotype= VIDEO_WMV);
```

Parameters

hWnd

Handle of owner window.

cameramode

If this value is STILL_MODE, setting camera to make still image.

If this value is VIDEO_MODE, setting camera to make video.

videotype

If this value is VIDEO_WMV, videos file is saved *.wmv.

If this value is VIDEO_ASF, videos file is saved *.asf.

Return Values

TRUE if camera is successfully initialized. Otherwise, it is FALSE.

C++

```
#include "M3SkyCamera.h"

m_cameramode = STILL_MODE;    // still Mode
m_videotype = VIDEO_ASF;      // video File Type
OpenCamera(this->m_hWnd,m_cameramode,m_videotype);
```

4.2.1.2 CloseCamera

This function performs uninitialization. And switches preview into a stop state.

```
void CloseCamera( );
```

Parameters

None

Return Values

None

C++

```
#include "M3SkyCamera.h"

CloseCamera();
```

4.2.1.3 PreviewStart

This function starts preview.

```
BOOL PreviewStart( );
```

Parameters

None

Return Values

TRUE if preview is successfully a running state. Otherwise, it is FALSE.

Remarks

This function can be performed, after OpenCamera function is called.

4.2.1.4 PreviewStop

This function stops preview.

```
BOOL PreviewStop( );
```

Parameters

None

Return Values

TRUE if preview is successfully a stopped state. Otherwise, it is FALSE.

Remarks

This function can be performed, after PreviewStart function is called.

C++

```
#include "M3SkyCamera.h"

PreviewStop();
```

4.2.1.5 CaptureStill

This function makes still image file. CaptureStill can make jpg, bmp, png type file.

```
WCHAR CaptureStill(WCHAR FileFullName);
```

Parameters

FileFullName

If FileFullName is NULL, automatically makes image file of jpg type(current date.jpg).

If FileFullName is not NULL, makes image file same as filefullname.

Return Values

Return value is image file's full name made.

Remarks

When preview is a running state and camera mode is a still mode, this function can be operated.

C++

```
#include "M3SkyCamera.h"

TCHAR name[1024] = {0x00};

if(bSaveDateMode)
{
    TCHAR *Name;
    TCHAR *reName;
    Name = NULL;

    reName = CaptureStill(Name);
}
else
{
    wcscpy(name, L"1.bmp");
    CaptureStill(name);
}
```

4.2.1.6 VideoStart

This function starts video.

```
BOOL VideoStart(PTCHAR FileFullName);
```

Parameters

FileFullName

Video file name

Return Values

TRUE if video is successfully into a running state. Otherwise, it is FALSE.

Remarks

When preview state is running, this function can be performed, after OpenCamera function is called.

C++

```
#include "M3SkyCamera.h"

TCHAR name[50] = {0x00};

if(name[0] == 0x00)
{
    VideoStart(0);    // file full name : \My Documents\My Pictures\\(DATE).wmv
}
else
{
    VideoStart(name);
}
```

4.2.1.7 VideoStop

This function stops video.

```
BOOL VideoStop( );
```

Parameters

None

Return Values

TRUE if video is successfully into a stopped state. Otherwise, it is FALSE.

Remarks

When Video state is running, this function can be performed, after VideoStart function is called.

C++

```
#include "M3SkyCamera.h"

VideoStop();
```

4.2.1.8 AutoFocus

This function operates autofocus action.

```
BOOL AutoFocus( );
```

Parameters

option

None

Return Values

TRUE if Autofocus is successfully operated. Otherwise, it is FALSE.

Remarks

When preview is a running state, this function can be performed.

```
C++
#include "M3SkyCamera.h"

if(!AutoFocus())
{
    // Fail
}
else
{
    // Success
}
```

4.2.1.9 FlashOn

This function controls flash power.

```
BOOL FlashOn(BOOL on);
```

Parameters

on

If on is TRUE, flash power enables.

If on is FALSE, flash power disables.

Return Values

TRUE if flash power is successfully set. Otherwise, it is FALSE.

Remarks

None

```
C++
#include "M3SkyCamera.h"

if(bWantFlashOn)
{
    FlashOn(TRUE);
}
else
{
    FlashOn(FALSE);
}
```

4.2.1.10 SetWindowPosition

This function sets the position of the video window (not the client rectangle position) in device coordinates.

```
BOOL SetWindowPosition(long left, long top, long width, long height);
```

Parameters

left

Specifies the x-axis origin of the window.

top

Specifies the y-axis origin of the window.

width

Width of the window.

height

Height of the window.

Return Values

TRUE if camera is successfully initialized. Otherwise, it is FALSE.

4.2.1.11 SetQuality

This function specifies the quality of still image of jpeg type.

```
void SetQuality(DWORD value);
```

Parameters

value

The value specifies quality (0 : Low, 1 : Normal, 2 : High).

Return Values

If the method succeeds, it returns TRUE. Otherwise it returns FALSE.

Remarks

When preview is a stopping state, this function can be performed.

C++

```
#include "M3SkyCamera.h"

// 0 is Low quality
// 1 is Normal quality
// 2 is High quality

SetQuality(2);
```

4.2.1.12 GetQuality

This function gets the quality of still image of jpeg type.

```
DWORD GetQuality( );
```

Parameters

None

Return Values

Returns quality value.

Remarks

When preview is a stopping state, this function can be performed.

C++

```
#include "M3SkyCamera.h"

DWORD dwQuality;

// 0 is Low quality
// 1 is Normal quality
// 2 is High quality

dwQuality = GetQuality();
```

4.2.1.13 SetResolution

This function specifies the resolution.

```
BOOL SetResolution(int PinType, int nResolution);
```

Parameters

PinType

The value specifies pin (1 : still pin, 2 : capture pin). Capture pin resolution is supported.

nResolution

The range of value is 0 to 7.

(0 : 176*144, 1 : 320*240, 2 : 352*288, 3 : 640*480, 4 : 800*600, 5 : 1280*960, 6 : 1600*1200, 7 : 2048*1536)

Refer to subkeys in

"HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\PictureResolution\".

Return Values

If the method succeeds, it returns TRUE. Otherwise it returns FALSE.

Remarks

When preview is a stopping state, this function can be performed.

C++

```
#include "M3SkyCamera.h"

int nResolution;

// 0 is 176*144
// 1 is 320*240
// 2 is 352*288
// 3 is 640*480
// 4 is 800*600
// 5 is 1280*960
// 6 is 1600*1200
// 7 is 2048*1536

if(bStillMode)
{
    SetResolution(1, 2);
}
else
{
    SetResolution(2, 2);
}
```

4.2.1.14 GetResolution

This function gets the resolution.

```
long GetResolution(int PinType);
```

Parameters

PinType

The value specifies pin (1 : still pin, 2 : capture pin).

Return Values

Returns resolution value.

(0 : 176*144, 1 : 320*240, 2 : 352*288, 3 : 640*480, 4 : 800*600, 5 : 1280*960, 6 : 1600*1200, 7 : 2048*1536)

Refer to subkeys in

"HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\PictureResolution\".

Remarks

When preview is a stopping state, this function can be performed.

C++

```
#include "M3SkyCamera.h"

int nResolution;

// 0 is 176*144
// 1 is 320*240
// 2 is 352*288
// 3 is 640*480
// 4 is 800*600
// 5 is 1280*960
// 6 is 1600*1200
// 7 is 2048*1536

if(bStillMode)
{
    nResolution = GetResolution(1);
}
else
{
    nResolution = GetResolution(2);
}
```

4.2.1.15 SetBrightness

This function specifies the brightness.

```
BOOL SetBrightness(long value);
```

Parameters

value

The range of value is +3 to -3.

Return Values

If the method succeeds, it returns TRUE. Otherwise it returns FALSE.

Remarks

When preview is a stopping state, this function can be performed.

C++

```
#include "M3SkyCamera.h"

long value;

value = 0;

SetBrightness(value);
```

4.2.1.16 GetWhiteBalance

This function gets the whitebalance.

```
long GetWhiteBalance( );
```

Parameters

None

Return Values

Return whitebalance value (0 : Auto, 1 : Sunny, 2 : Cloudy, 3 : Fluorescent, 4 : Incandescent).

Remarks

When preview is a stopping state, this function can be performed.

C++

```
#include "M3SkyCamera.h"

int nWhiteBalanceMode;

// 0 is Auto
// 1 is Sunny
// 2 is Cloudy
// 3 is Fluorescent
// 4 is Incandescent

nWhiteBalanceMode= GetWhiteBalance();
```

4.2.1.17 GetInfo

This function gets dll information.

```
PTCHAR GetInfo( );
```

Parameters

None

Return Values

Returns dll information string.

Remarks

None

4.2.1.18 SetAfMode

This function sets the Auto Focus mode.

```
int SetAfMode(int nMode);
```

Parameters

Manual AF

0 : Enables AF only when pressing AF button.

Always AF

1 : Enables AF before shooting.

Auto AF

2: Performs AF by automatically recognizing preview image changes.

Return Values

Returns AF mode as currently set.

C++

```
#include "M3SkyCamera.h"

void M3Camera_SetAF(int nMode)
{
    switch(nMode)
    {
        case 0:
            SetAfMode(0);
        case 1:
            SetAfMode(1);
        case 2:
```

```
        SetAfMode(2);  
    }  
}
```

4.2.2 Reference and Function List for C#

Enum

CAMERA_MODE

```
public enum CAMERA_MODE
{
    STILL_MODE = 0,
    VIDEO_MODE
};
```

VIDEO_TYPE

```
public enum VIDEO_TYPE
{
    VIDEO_ASF = 0,
    VIDEO_WMV
};
```

4.2.2.1 Open

This function performs initialization. And switches preview into a running state.

```
bool Open(ntPtr hWnd, CAMERA_MODE cameramode,  
          VIDEO_TYPE videotype= VIDEO WMV );
```

Parameters

hWnd

Handle of owner window.

cameramode

If this value is STILL_MODE, setting camera to make still image.

If this value is VIDEO_MODE, setting camera to make video.

videotype

If this value is VIDEO_WMV, videos file is saved *.wmv.

If this value is VIDEO_ASF, videos file is saved *.asf.

Return Values

TRUE if camera is successfully initialized. Otherwise, it is FALSE.

C#

```
private Camera m_Cam;  
  
private CAMERA_MODE m_cameramode;  
private VIDEO_TYPE m_videotype;  
  
m_Cam = new Camera();  
  
m_cameramode = CAMERA_MODE.STILL_MODE;  
m_videotype = VIDEO_TYPE.VIDEO_ASF;  
  
m_Cam.Open(this.Handle, m_cameramode, m_videotype);
```

4.2.2.2 Close

This function performs uninitialization. And switches preview into a stop state.

```
void Close( );
```

Parameters

None

Return Values

None

C#

```
Camera.Close();
```

4.2.2.3 Preview_Start

This function starts preview.

```
bool Preview_Start( );
```

Parameters

None

Return Values

TRUE if preview is successfully a running state. Otherwise, it is FALSE.

Remarks

This function can be performed, after Open function is called.

C#

```
private Camera m_Cam;

m_Cam = new Camera();

m_Cam.Preview_Start();
```

4.2.2.4 Preview_Stop

This function stops preview.

```
bool Preview_Stop( );
```

Parameters

None

Return Values

TRUE if preview is successfully a stopped state. Otherwise, it is FALSE.

Remarks

This function can be performed, after Preview_Start function is called.

C#

```
private Camera m_Cam;

m_Cam = new Camera();

m_Cam.Preview_Stop();
```

4.2.2.5 Capture

This function makes still image file. CaptureStill can make jpg, bmp, png type file.

```
void Capture(string FileFullName, StringBuilder path);
```

Parameters

FileFullName

If FileFullName is NULL, automatically makes image file of jpg type(current date.jpg).

If FileFullName is not NULL, makes image file same as filefullname.

path

If FileFullName is NULL, path will have FileFullName.

Return Values

None

Remarks

When preview is a running state and camera mode is a still mode, this function can be operated.

C#

```
private Camera m_Cam;

m_Cam = new Camera();

if (m_bStillDateSaveMode)
{
    StringBuilder path = new StringBuilder(100);
```

```

        m_Cam.Capture(null, path);
    }
    else
    {
        filefullname = m_strStillFolder + m_strStillFileName;
        m_Cam.Capture(filefullname);
    }
}

```

4.2.2.6 Video_Start

This function starts video.

```
bool Video_Start(string FileFullName);
```

Parameters

FileFullName

Video file name

Return Values

TRUE if video is successfully into a running state. Otherwise, it is FALSE.

Remarks

When preview state is running, this function can be performed, after Open function is called.

C#

```

private Camera m_Cam;

m_Cam = new Camera();

if (m_bVideoDateSaveMode)
{
    m_Cam.Video_Start(null);    // file full name : \My Documents\My
    Pictures\\(DATE).wmv
}
else
{
    filefullname = m_strVideoFolder + m_strVideoFileName;
    m_Cam.Video_Start(filefullname);
}

```

4.2.2.7 Video_Stop

This function stops video.

```
bool Video_Stop( );
```

Parameters

None

Return Values

TRUE if video is successfully into a stopped state. Otherwise, it is FALSE.

Remarks

When Video state is running, this function can be performed. After Video_Start function is called.

C#

```

private Camera m_Cam;

m_Cam = new Camera();

```

```
m_Cam.Video_Stop();
```

4.2.2.8 SetAutoFocus

This function operates autofocus action.

```
bool SetAutoFocus( );
```

Parameters

None

Return Values

TRUE if preview is successfully a running state. Otherwise, it is FALSE.

Remarks

None

C#

```
private Camera m_Cam;  
  
m_Cam = new Camera();  
  
m_Cam.SetAutoFocus();
```

4.2.2.9 Flash_On

This function controls flash power.

```
bool Flash_On(bool on);
```

Parameters

on

If on is TRUE, flash power enables.

If on is FALSE, flash power disables.

Return Values

TRUE if Flash power is successfully set. Otherwise, it is FALSE.

Remarks

None

C#

```
private Camera m_Cam;  
  
m_Cam = new Camera();  
  
if(bFlashOn)  
    m_Cam.Flash_On(true);  
else  
    m_Cam.Flash_On(false);
```

4.2.2.10 SetPosition

This function sets the position of the video window (not the client rectangle position) in device coordinates.

```
bool SetPosition(int left, int top, int width, int height);
```

Parameters

left

Specifies the x-axis origin of the window.

top

Specifies the y-axis origin of the window.

width

Width of the window.

height

Height of the window.

Return Values

True if camera is successfully initialized. Otherwise, it is FALSE.

4.2.2.11 Set_Quality

This function specifies the quality of still image of jpeg type.

```
void Set_Quality(int value);
```

Parameters

value

The value specifies quality (0 : Low, 1 : Normal, 2 : High).

Return Values

None

Remarks

When preview is a stopping state, this function can be performed.

4.2.2.12 Get_Quality

This function gets the quality of still image of jpeg type.

```
int Get_Quality( );
```

Parameters

None

Return Values

Return quality value.

Remarks

When preview is a stopping state, this function can be performed.

4.2.2.13 Set_Resolution

This function specifies the resolution.

```
bool Set_Resolution(int PinType, int nResolution);
```

Parameters

PinType

The value specifies pin (1 : still pin, 2 : capture pin).

nResolution

The value's range is 0 ~ 6.

(0 : 176*144, 1 : 320*240, 2 : 352*288, 3 : 640*480, 4 : 800*600, 5 : 1280*960, 6 : 1600*1200, 7 : 2048*1536)

Refer to subkeys in

"HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\PictureResolution\".

Return Values

If the method succeeds, it returns TRUE. Otherwise it returns FALSE.

Remarks

When preview is a stopping state, this function can be performed.

C#

```
private Camera m_Cam;

m_Cam = new Camera();

// 0 is 176*144
// 1 is 320*240
// 2 is 352*288
// 3 is 640*480
// 4 is 800*600
// 5 is 1280*960
// 6 is 1600*1200
// 7 is 2048*1536

if(bStillMode == true)
{
    m_Cam.Set_Resolution(1,0);
}
else
{
    m_Cam.Set_Resolution(2,0);
}
```

4.2.2.14 Get_Resolution

This function gets the resolution.

```
int Get_Resolution( int PinType );
```

Parameters

PinType

The value specifies pin (1 : still pin, 2 : capture pin).

Return Values

Return resolution value.

(0 : 176*144, 1 : 320*240, 2 : 352*288, 3 : 640*480, 4 : 800*600, 5 : 1280*960, 6 : 1600*1200, 7 : 2048*1536)

Refer to subkeys in

"HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\PictureResolution\".

Remarks

When preview is a stopping state, this function can be performed.

C#

```
private Camera m_Cam;

m_Cam = new Camera();

int nResolution;
nResolution = m_Cam.Get_Resolution();
```

4.2.2.15 Set_Brightness

This function specifies the brightness.

```
bool Set_Brightness(long value);
```

Parameters

value

Value's range is +3~-3.

Return Values

If the method succeeds, it returns TRUE. Otherwise it returns FALSE.

Remarks

When preview is a stopping state, this function can be performed.

C#

```
private Camera m_Cam;  
  
m_Cam = new Camera();  
  
int nBrightness = 3;  
  
m_Cam.Set_Brightness(nBrightness);
```

4.2.2.16 Get_WhiteBalance

This function gets the whitebalance.

```
int Get_WhiteBalance();
```

Parameters

None

Return Values

Return whitebalance value.(0 : Auto, 1 : Sunny, 2 : Cloudy, 3 : Fluorescent, 4 : Incandescent)

Remarks

When preview is a stopping state, this function can be performed.

C#

```
private Camera m_Cam;  
  
m_Cam = new Camera();  
  
int nWhiteBalance;  
  
nWhiteBalance = m_Cam.Get_WhiteBalance();
```

4.2.2.17 Get_Info

This function gets dll information.

```
void Get_Info(StringBuilder info);
```

Parameters

info

string buffer for information

Return Values

None

Remarks

None

4.2.2.18 SetAfMode

This function sets the Auto Focus mode.

```
int SetAfMode(int nMode);
```

Parameters

Manual AF

0 : Enables AF only when pressing AF button.

Always AF

1 : Enables AF before shooting.

Auto AF

2: Performs AF by automatically recognizing preview image change.

Return Values

Returns AF mode as currently set.

C#

```
private Camera m_Cam;

m_Cam = new Camera();

void M3Camera_SetAF(int nMode)
{
    switch(nMode)
    {
        case 0:
            m_Cam.SetAfMode(0);
        case 1:
            m_Cam.SetAfMode(1);
        case 2:
            m_Cam.SetAfMode(2);
    }
}
```

4.3 GPS

This section provides description of the functions and DLLs which are used to manage the GPS module.

Required Files

For C++

Required header:

M3GpsParse.h

Required lib:

M3GpsParse.Lib

Required DLL:

M3GpsParse.DLL

For C#

Required DLL:

M3GpsParse.DLL

Supported Product

M3 ORANGE

4.3.1 Reference and Function List for C++

Definitions

Status Return value

Please refer to the below table for the status value definition.

Status Value Definition	Code	Meaning
TRUE	1	Success
FALSE	0	General Error

Structure

GPSDop

```
struct GPSDop{
    double dPDop;
    double dHDop;
    double dVDop;
};
```

GPSSatellite

```
struct GPSSatellite{
    int nID;
    int nElevation;
    int nAzimuth;
    int nSNR;
};
```

GPSParseInfo

```
struct GPSParseInfo{
    struct    GPSSatellite mSat[SATELLITE_COUNT];
    struct    GPSDop mDop;
    int       nSatInUse;
    int       nSatNum;
    BOOL      bSatInfo;
    double    dHeading;
    double    dVelocity;
    BOOL      bNorthLatitude;
    BOOL      bEastLongitude;
    double    dLatitude;
    double    dLongitude;
    double    dAltitude;
    double    dUTCDate;
    double    dUTCTime;
    int       nPosFix;
    int       nGPSStatus;
    CString   mNMEAMsg;
};
```

Parameters

M3GPS_ModuleRestart() functions use parameters defined as below.

Parameter Value Definition	Code	Meaning
GPS_COLD_START	0	Cold start command for GPS
GPS_WARM_START	1	Warm start command for GPS
GPS_HOT_START	2	Hot start command for GPS

Windows Message

DLL sends the below message to the user to inform the data received from the satellite. It sends the message thru windows handle that is registered when GPS open.

Parameter Value Definition	Code	Meaning
WM_USER_RECVDATA	WM_USER+10000	Message sent from GPS module to the user

4.3.1.1 M3GPS_Open

The **M3GPS_Open** function opens a COM port to a GPS module and performs initialization.

```
BOOL M3GPS_Open(HWND hMainWnd, TCHAR* tzComPort, int nBaudRate);
```

Parameters

hMainWnd

Register a window handle to received data from GPS module.

tsComPort

Configures the GPS Com port for M3 handheld device.

nBaudRate

Configures the Baudrate for communication with M3 handheld device.

Return Values

If the function succeeds, the return value is nonzero (True).

If the function fails, the return value is zero (False).

Remarks

GPS receives data from the satellite. The data is sent by the DLL thru WM_USER_RECEIVEDATA. User can process the data after reception.

C++

```
BEGIN_MESSAGE_MAP(CGpsParseDemoDlg, CDialog)
ON_MESSAGE(WM_USER_RECVDATA, OnRecvGPSData)
END_MESSAGE_MAP()

void OpenGps(TCHAR* tzCom, int nBaudRate)
{
    M3GPS_Open(m_hWnd, tzCom, nBaudRate);
}

long CGpsParseDemoDlg::OnRecvGPSData(WPARAM wParam, LPARAM lParam)
{
    GPSParseInfo *pInf = (GPSParseInfo*)wParam;
    return 0;
}
```

4.3.1.2 M3GPS_Close

The **M3GPS_Close** function closes a COM port to a GPS module.

```
BOOL M3GPS_Close();
```

Parameters

None

Return Values

If the function succeeds, the return value is nonzero (True).

If the function fails, the return value is zero (False).

Remarks

GPS uses serial communication. Hence, closing will disconnect the connection with the satellite and DLL will NOT send messages.

C++

```
void CloseGps()
{
    M3GPS_Close();
}
```

4.3.1.3 M3GPS_ModuleRestart

The **M3GPS_ModuleRestart** function resets the GPS module.

```
BOOL M3GPS_ModuleRestart(int nStartType);
```

Parameters

nStartType

Configures the restart type: Cold, Warm or Hot start.

Return Values

If the function succeeds, the return value is nonzero (True).

If the function fails, the return value is zero (False).

Remarks

For faster re-connection after an initial connection, GPS module remembers the data such as location and time. Deleting those data will reset the module.

Cold Start: A condition in which the GPS receiver can arrive at a navigation solution without initial position, time, and current Ephemeris.

Warm Start: Start mode of the GPS receiver when current position, clock offset, and approximate GPS time are input by the user. Ephemeris data is not available.

Hot Start: Start mode of the GPS receiver when current position, clock offset, approximate GPS time, and current ephemeris data are all available.

C++

```
void ModuleRestart()
{
    M3GPS_ModuleRestart(GPS_COLD_START);
}
```

4.3.1.4 M3GetGpsInfoProc

M3GetGpsInfoProc, a function pointer, is used to obtain GPS data. This function is only available in C++

```
typedef void (*M3GetGpsInfoProc)(GPSParseInfo info);
```

Parameters

info

GPSParseInfo, structure type, gets data from GPS satellite.

Return Values

None

Remarks

GPS module receives data from GPS satellite when it is opened. Data is transferred to the device by serial communication. M3GpsParse.dll sends data received from GPS module by two means; message method and function pointer method. Former is introduced in Tutorial section and latter is described below.

C++

```
typedef void (*M3GetGpsInfoProc)(GPSParseInfo info);

void open()
{
    TCHAR tzCom[8] = {0x00};
    wsprintf(tzCom, L"COM2:"); // Input Comport
    M3GPS_Open(m_hWnd, tzCom, 9600, fnParseGps);
}
```

```
void fnParseGps(GPSParseInfo info)
{
    // DLL sends GPS data as GPSParseInfo structure type.
    // Use GPS data with info.
}
```

4.3.2 Reference and Function List for C#

Definitions

Status Return value

Please refer to the below table for the status value definition.

Status Value Definition	Code	Meaning
TRUE	1	Success
FALSE	0	General Error

Structure

GPSDop

```
[StructLayout(LayoutKind.Sequential)]
public struct GPS_DOP
{
    public double dPDop;    // Position dilution of precision
    public double dHDop;    // Horizontal dilution of precision
    public double dVDop;    // Vertical dilution of precision
};
```

GPSSatellite

```
[StructLayout(LayoutKind.Sequential)]
public struct GPS_SATELLITE
{
    public int nID;          // Satellite PRN number
    public int nElevation;   // Elevation, degrees
    public int nAzimuth;     // Azimuth, degrees
    public int nSNR;         // Signal to noise ration in dBHz
};
```

GPSParseInfo

```
[DllImport("M3GpsParse.dll")]
private static extern bool M3GPS_Open(IntPtr hMainWnd, string tzComPort, int nBaudRate);

[DllImport("M3GpsParse.dll")]
private static extern bool M3GPS_Close();

[DllImport("M3GpsParse.dll")]
private static extern bool M3GPS_ModuleRestart(int nStartType);

public bool Gps_Open(IntPtr hMainWnd, string tzComPort, int nBaudRate)
{
    return M3GPS_Open(hMainWnd, tzComPort, nBaudRate);
}

public bool Gps_Close()
{
    return M3GPS_Close();
}

public bool Gps_ModuleRestart(int nStarType)
{
    return M3GPS_ModuleRestart(nStarType);
}
```

Parameters

M3GPS_ModuleRestart() functions use parameters defined as below.

Parameter Value Definition	Code	Meaning
GPS_COLD_START	0	Cold start command for GPS
GPS_WARM_START	1	Warm start command for GPS
GPS_HOT_START	2	Hot start command for GPS

Windows Message

DLL sends the below message to the user to inform the data received from the satellite. It sends the message thru windows handle that is registered when GPS open.

Parameter Value Definition	Code	Meaning
WM_USER_RECVDATA	0x0400+10000	Message sent from GPS module to the user

4.3.2.1 Class_Gps_Parse.GPS_Open

The **Class_Gps_Parse.GPS_Open** function opens a COM port to a GPS module and performs initialization.

```
Bool Class_Gps_Parse.Gps_Open(IntPtr hMainWnd, string tzComPort, int nBaudRate);
```

Parameters

hMainWnd

Register a window handle to received data from GPS module.

tzComPort

Configures the GPS Com port for M3 handheld device.

nBaudRate

Configures the Baudrate for communication with M3 handheld device.

Return Values

If the function succeeds, the return value is nonzero (True).

If the function fails, the return value is zero (False).

Remarks

GPS receives data from the satellite. The data is sent by the DLL thru WM_USER_RECEIVEDATA. User can process the data after reception.

C#

```
namespace GpsParseDemoNet{
    public partial class Form_GPS : Form{
        Class_Gps_Parse cGps;
        MsgWindow MsgWin;
        IntPtr m_hWnd;
        private void connect_Gps(string strComport){
            cGps = new Class_Gps_Parse();
            MsgWin = new MsgWindow(this);
            m_hWnd = MsgWin.Hwnd;
            cGps.Gps_Open(MsgWin.Hwnd, strComport, 9600);
        }
        public long OnRecvGpsData(IntPtr wParam, IntPtr lParam){
            Class_Gps_Parse.GPS_PARSE_INFO info
                = new Class_Gps_Parse.GPS_PARSE_INFO();
            info = (Class_Gps_Parse.GPS_PARSE_INFO)
                Marshal.PtrToStructure(wParam,typeof(Class_Gps_Parse.GPS_PARSE_INFO));
            return 0;
        }
    }
    public class MsgWindow : MessageWindow{
        private Form_GPS msgform;
        public MsgWindow(Form GPS form){
            this.msgform = form;
        }
        protected override void WndProc(ref Message msg){
            switch (msg.Msg)
            {
                case Class_Gps_Parse.WM_USER_RECVDATA:
                    this.msgform.OnRecvGpsData(msg.WParam, msg.LParam);
                    break;
            }
            base.WndProc(ref msg);
        }
    }
}
```

4.3.2.2 Class_Gps_Parse.Gps_Close

The **Class_Gps_Parse.GPS_Close** function closes a COM port to a GPS module.

```
Bool Class_Gps_Parse.Gps_Open(IntPtr hMainWnd, string tzComPort, int nBaudRate);
```

Parameters

None

Return Values

If the function succeeds, the return value is nonzero (True).

If the function fails, the return value is zero (False).

Remarks

GPS uses serial communication. Hence, closing will disconnect the connection with the satellite and DLL will NOT send messages.

C#

```
void CloseGps()
{
    cGps.Gps_Close();
}
```

4.3.2.3 Class_Gps_Parse.Gps_ModuleRestart

The **Class_Gps_Parse.GPS_ModuleRestart** function resets the GPS module.

```
BOOL M3GPS_ModuleRestart(int nStartType);
```

Parameters

nStartType

Configures the restart type: Cold, Warm or Hot start.

Return Values

If the function succeeds, the return value is nonzero (True).

If the function fails, the return value is zero (False).

Remarks

For faster re-connection after an initial connection, GPS module remembers the data such as location and time. Deleting those data will reset the module.

Cold Start: A condition in which the GPS receiver can arrive at a navigation solution without initial position, time, and current Ephemeris.

Warm Start: Start mode of the GPS receiver when current position, clock offset, and approximate GPS time are input by the user. Ephemeris data is not available.

Hot Start: Start mode of the GPS receiver when current position, clock offset, approximate GPS time, and current ephemeris data are all available.

C#

```
void ModuleRestart()
{
    cGps.Gps_ModuleRestart(Class_Gps_Parse.GPS_COLD_START);
}
```

4.4 RFID

This section provides description of the functions and DLLs which are used to manage the RFID module.

Required Products

For C++

Required header:

CFReaderLib.h

Required lib:

CFReader.lib

Required DLL:

CFReader.dll

For C#

Required DLL:

CFReader.dll, CFReaderDLLWrapper.dll

Supported Product

M3 ORANGE needs a RFID module.

4.4.1 Reference and Function List for C++

Structure

readerConfig

```
struct readerConfig
{
    long baudRate;
    char protocol;
    unsigned char stationID;
};
```

presetSettings

```
struct presetSettings
{
    long baudRate;
    char protocol;
};
```

4.4.1.1 RDR_OpenComm

The **RDR_OpenComm** function initializes and opens a specified communication device.

```
char RDR_OpenComm(char* device, char autodetect, struct presetSettings* settings);
```

Parameters

Device

0 terminated string containing the communication device.

"com1", "com2"...

Only serial devices are supported.

The device buffer size has the range of 10 bytes.

autodetect

0 : The auto detection of the baud rate and the protocol is deactivated. Values from settings are used.

1 : The normal auto detection of the baud rate and the protocol is activated.

2 : The fast auto detection of the baud rate and the protocol is activated. Older reader devices may not be found.

4 : If this bit is additionally set, not common baud rates are disabled (19200, 38400, 230400). Use this to speed up the autodetect procedure.

If MSB (80h) is set the power management for WindowsCE will be deactivated.

settings

Pointer to the structure of preset settings.

Return Values

If the function succeeds, the return value is a handle of the opened communication device.

If the communication port is opened on a Windows CE device successfully the return value is one.

If the function fails, the return value is zero.

C++

```
#include "CFReaderLib.h"

char com_port[10];

sprintf(com_port, "MOC%s", "1"); // M3Sky Rfid port
if(RDR_OpenComm(com_port, 1, NULL) == 0)
{
    AfxMessageBox(_T("OpenComm Failed"));
    return;
}
```

4.4.1.2 RDR_OpenReader

The **RDR_OpenReader** function opens a reader on specified communication device with specified ID.

```
void* RDR_OpenReader(unsigned char id, short knownReader);
```

Parameters

id

Valid station ID of the reader.

knownReader

0 : auto detect reader type

1 : unknown reader type

2 : Mifare 0.14e / f

3 : Mifare 0.15d

4 : Mifare 1.0

6 : ISO 0.9v

8 : MULTITAG 0.10b

10 : MULTITAG 0.12a

11 : MULTITAG 0.12bg
12 : Mifare 0.15e
15 : all Bootloader 0.x / 1.x
16 : all Dual 2.x
18 : all MULTITAG 1.x
19 : all LFX 1.x
20 : all MultilSO 1.x

Return Values

If the function succeeds, the return value is a handle of the opened reader.
If the function fails, the return value is zero.

C++

```
#include "CFReaderLib.h"

if(RDR_OpenReader(1, 0) == 0)
{
    AfxMessageBox(_T("OpenReader Failed"));
    return;
}
```

4.4.1.3 RDR_SendCommand

The **RDR_SendCommand** function sends a command to a reader specified by its handle.

```
long RDR_SendCommand(char* command, char* data);
```

Parameters

command

00 terminated string with the command.

data

0 terminated string containing the data.

Return Values

If the function succeeds, the return value is greater than zero. If the function fails, the return value is zero. If the command is not available for the reader, the return value is -1.

Remarks

Unsupported commands were sent to the reader without modifications.

C++

```
#include "CFReaderLib.h"

char buffer[514];
memset(buffer, 0x00, 514);

RDR_SendCommand("c", "");

int nCount = 0;
while(nCount++ < 100)
{
    RDR_GetData(buffer);
}

RDR_AbortContinuousReadExt();

RDR_EmptyCommRcvBuffer();
```

4.4.1.4 RDR_SendCommandGetData

The **RDR_SendCommandGetData** function sends a command to a reader specified by its handle and receives data.

```
char* RDR_SendCommandGetData(char* command, char* data, char* buffer);
```

Parameters

command

0 terminated string with the command.

data

0 terminated string containing the data.

buffer

A buffer of the return value.

The buffer size has the range of 514 bytes.

Return Values

Refer to RDR_GetData with continuous receive mode is off.

C++

```
#include "CFReaderLib.h"

char buffer[514];
memset(buffer, 0x00, 514);

// "ot" Select Tag Type command, 'o' Select tag type and 't' is 'activate all
type'
RDR_SendCommandGetData("o", "t", buffer);

RDR_EmptyCommRcvBuffer();
```

4.4.1.5 RDR_SetReaderConfig

The **RDR_SetReaderConfig** function sets and applies the reader with specified configuration. The communication device even adapted to this new settings.

```
RDR_SetReaderConfig(struct readerConfig* config);
```

Parameters

config

Pointer to structure of the configuration data.

For structure definition of reader configuration see header files.

Remarks

This function does not work if unknown reader type is used.

Use this function for only one reader at the same time. Otherwise the communication to other devices will fail.

Be sure the used interface supports the baud rate.

C++

```
#include "CFReaderLib.h"

readerConfig settings = RDR_GetReaderConfig();

settings.protocol = 0;

RDR_SetReaderConfig(&settings);
```

4.4.1.6 RDR_GetReaderConfig

The **RDR_GetReaderConfig** function returns current reader configuration.

```
struct readerConfig RDR_GetReaderConfig();
```

Return Values

The function returns a structure of configuration data of a reader.

C++

```
#include "CFReaderLib.h"

readerConfig settings = RDR_GetReaderConfig();

settings.protocol = 0;

RDR_SetReaderConfig(&settings);
```

4.4.1.7 RDR_GetData

The **RDR_GetData** function receives data from a reader specified with its handle.

```
char* RDR_GetData(char* buffer);
```

Parameters

buffer

A buffer of the return value. The buffer size has the range of 514 bytes.

Return Values

The function returns a 0 terminated string containing received data.

C++

```
#include "CFReaderLib.h"

char buffer[514];
memset(buffer, 0x00, 514);

RDR_SendCommand("c", "");

int nCount = 0;
while(nCount++ < 100)
{
    RDR_GetData(buffer);
}

RDR_AbortContinuousReadExt();

RDR_EmptyCommRcvBuffer();
```

4.4.1.8 RDR_AbortContinuousRead

The **RDR_AbortContinuousRead** function aborts the continuous read command in ASCII mode and empties the buffer.

```
RDR_AbortContinuousRead();
```

Parameters

None

Return Values

None

```
C++
#include "CFReaderLib.h"

char buffer[514];
memset(buffer, 0x00, 514);

RDR_SendCommand("c", "");

int nCount = 0;
while(nCount++ < 100)
{
    RDR_GetData(buffer);
}

RDR_AbortContinuousReadExt();

RDR_EmptyCommRcvBuffer();
```

4.4.1.9 RDR_EmptyCommRcvBuffer

The **RDR_EmptyCommRcfBuffer** function empties the receive buffer of the communication device.

```
RDR_EmptyCommRcvBuffer();
```

Parameters

None

Return Values

None

```
C++
#include "CFReaderLib.h"

char buffer[514];
memset(buffer, 0x00, 514);

RDR_SendCommand("c", "");

int nCount = 0;
while(nCount++ < 100)
{
    RDR_GetData(buffer);
}

RDR_AbortContinuousReadExt();

RDR_EmptyCommRcvBuffer();
```

4.4.2 Command List for C++

4.4.2.1 Antenna power on/off

This command controls the antenna power. It can be used to decrease the power consumption of the reader.

<pre>public byte RDR_OpenComm(string device, byte autodetect, ACG_CFReader.presetSettings settings);</pre>	
Command	Data
'pon'	Switch reader on
'poff'	Put reader in standby mode
Answer	Description
'p'	Positive acknowledge

Power on

The reader leaves standby mode and is ready for the next command. Sending a tag command (i.e. select, continuous read) the reader is powered up.

Power off

The reader enters standby mode. Power consumption is decreased. All tags in the antenna field are powered off and reset. Standby mode is only entered manually. To switch off the whole unit, pin 16 (Enable) has to be set to logic low.

<pre>C++ #include "CFReaderLib.h" if(bif(bAntenna) RDR_SendCommand("pon", ""); else RDR_SendCommand("poff", ""); RDR_EmptyCommRcvBuffer();</pre>	
--	--

4.4.2.2 Set tag type

This command sets up the reader for a specific tag type. The continuous read function will speed up because only this type of tag is addressed. After a reset, the reader starts as defined in its start-up configuration. 't' which is activatve to all tags is default. Read/Write speed can be faster by setting a tag type to a specific one.

Command	Data
'o'	ISO type (1 byte) 'a' ISO 14443 Type A 'b' ISO 14443 Type B 'd' ICODE UID 'e' ICODE EPC 'i' ICODE 's' SR176 't' activate all tags 'v' ISO 15693
Answer	Description
'OA' 'OB' 'OD' 'OE' 'OI' 'OS'	String of tag type

'OT'	
'OV'	

```
C++
#include "CFReaderLib.h"

char buffer[514];
memset(buffer, 0x00, 514);

// "ot" Select Tag Type command, 'o' Select tag type and 't' is 'activate all
type'
RDR_SendCommandGetData("o", "t", buffer);

RDR_EmptyCommRcvBuffer();
```

4.4.2.3 Select

This command selects a single card in the antenna field. It can only be used in single tag mode. If successfully executed, the command returns the UID of the selected card. The reader detects the length of the UID automatically.

Command	Data
's'	None

Answer	Description
Data	Serial number
'N'	Error : No Tag in the field

Select a single tag

No previous continuous read is required. The command executes an automatic field reset.

Multiple tags

This command is designed for fast access of a single tag in the field. If multiple cards are used the 'm' instruction has to be used instead.

```
C++
#include "CFReaderLib.h"

char buffer[514];
memset(buffer, 0x00, 514);

RDR_SendCommandGetData("s", "", buffer);

RDR_EmptyCommRcvBuffer();
```

4.4.2.4 Multi-Tag Selection / List

This command detects several tags at the same time. It replaces the fast select command ('s') in multiple tag surroundings. The Multi-Tag List command lists all tags with their serial numbers. Use the Multi-Tag Select command to select a single tag. Each tag has to be selected separately.

Command	Data
'm'	Serial number (n bytes). <CR> (1 byte)

Answer	Description
Data	Serial number
'N'	Error : No Tag in the field

Multi-tag list

Sending a as the first parameter, the reader returns a list of all tags present in the antenna field. In the end the total number of tags detected is returned.

Reading distance

Each card needs a specific amount of power. The reader always provides the same power level. Therefore, the reading distance will decrease if more tags are present. Basically, the reading distance depends on the tag, the antenna and the tuning of the antenna.

Multi-tag select

Using the serial number with as parameter, the corresponding tag will be selected. High-level interactions can be performed addressing only this card. All other tags remain silent.

Attention

It could be possible that ISO 15693 tags are interfered from ISO14443 type B and SR 176 tags. In this case the ISO 15693 tag will always answers on a multi list command even there had been no field reset before. In this case deactivate ISO 14443 B and SR 176 tags.

```
C++
#include "CFReaderLib.h"

char buffer[514];

char cDataCR = 0x0D;    // <CR>
RDR_SendComand("m", &cDataCR );

memset(buffer, 0x00, 514);
RDR_GetData(buffer);

int nTag = atoi(buffer);

for(int i = 0; i < nTag; i ++ )
{
    memset(buffer, 0x00, 514);
    RDR_GetData(buffer);
    // buffer had a Serial
}

RDR_EmptyCommRcvBuffer();
```

4.4.2.5 Read block

This command reads a data block on a card. The size of the returned data depends on the tag used. The block address range depends on the tag as well.

Command	Data
'r'	Block address (1 byte), valid range 00h – 40h
'rb'	Block address (1 byte)

Answer	Description
Data	data block (depends on tag type)
'F'	Error: read failure
'N'	Error : No Tag in the field
'O'	Error: Operation mode failure
'R'	Error: Out of range

Read failure 'F'

This error is returned if either the reader receives bad data or the block address exceeds the block address range of the sector.

No tag in field 'N'

The The tag does not respond. There is either no tag present or addressed.

Operation mode failure 'O'

The presented tag is not ISO14443 type A, SR 176, ICode, ICode-UID and ISO 15693 compliant. For ISO 14443 type A only mifare® tags are supported

Out of range failure 'R'

The block address of the 'r' command is higher than 40h. The block address of the 'r' command conflicts with other commands, therefore the block address has to be limited to 40h. Use the 'rb' command instead.

```
C++
#include "CFReaderLib.h"

char buffer[514];
memset(buffer, 0x00, 514);

RDR_SendCommandGetData("s", "", buffer);
// buffer have a serial number of tag

memset(buffer, 0x00, 514);
RDR_EmptyCommRcvBuffer();

RDR_SendCommandGetData("rb", "01", buffer);
// buffer have the 01 block data

RDR_EmptyCommRcvBuffer();
```

4.4.2.6 Continuous Read

The reader device reads and displays serial numbers continuously while one or more tags remain in the field. This command stops if any character is sent to the reader module. The reader module returns the character 'S' (53h). The reader supports different tag types at the same time. To increase the reading performance switch to a single tag mode. If more than one tag of the same type should be detected at the same time, the Multitag flag must be activated. The response data length depends on the tag type.

Command	Data
'c'	None

Answer	Description
Data	Serial number (n bytes)
'N'	Error : No Tag in the field (only binary protocol)

Multitag continuous read mode

If the Multitag flag is set in the Protocol Configuration (PCON) register the reader reads multiple tags continuously.

Auto start

The continuous read mode is started automatically in ASCII mode. The auto start flag must be set in the PCON register.

Noisy Environment

If the Noisy Environment flag is set, the continuous read mode can only be aborted with the '.' character. This is only valid in ASCII mode.

Binary mode

This command is fully supported in binary protocol mode except the test continuous read command and the noisy environment flag.

Do not use this command on bus system environment in binary mode, because the continuous read mode will take possession of the bus system.

Simple access control applications

Serial numbers are always sent plain. Data encryption is activated after a successful login. For simple access control applications the use read-only blocks for the identification of the tag is recommended.

Reading any block (even the manufacturer block) of the transponder will increase your security.

```
C++
#include "CFReaderLib.h"

char buffer[514];
memset(buffer, 0x00, 514);

RDR_SendCommand("c", "");

int nCount = 0;
while(nCount++ < 100)
{
    RDR_GetData(buffer);
}

RDR_AbortContinuousReadExt();

RDR_EmptyCommRcvBuffer();
```

4.4.2.7 Write block

This command writes data to a block. A read is done automatically after every write to ensure correct writing.

Command	Data
'w'	Block address (1 byte), valid range 00h – 40h. Data (n bytes)
'wb'	Block address (1 byte) Data (n bytes)

Answer	Description
Data	data block (depends on tag type)
'F'	Error: read failure
'N'	Error : No Tag in the field
'O'	Error: Operation mode failure
'R'	Error: Out of range

Write failure 'F'

This error is displayed if bad transmission conditions are given. If the block address exceeds the physical number of blocks of a tag, this error is shown.

No tag error 'N'

This error is returned if no tag is present or the card does not respond.

Operation mode failure 'O'

The presented tag is not ISO14443 type A, SR 176, ICode, ICode-UID and ISO 15693 compliant. For ISO 14443 type A only mifare® tags are supported.

Out of range failure 'R'

The block address of the 'w' command is higher than 40h. The block address of the 'w' command conflicts with other commands, therefore the block address has to be limited to 40h. Use the 'wb' command instead.

```
C++
#include "CFReaderLib.h"

char buffer[514];
memset(buffer, 0x00, 514);

RDR_SendCommandGetData("s", "", buffer);
// buffer have a serial number of tag
```

```
memset(buffer, 0x00, 514);
RDR_EmptyCommRcvBuffer();

RDR_SendCommandGetData("wb", "0511223344", buffer);
// Writes data 11223344 on block 05.

RDR_EmptyCommRcvBuffer();
```

4.4.2.8 Reset

This command executes a power on (software) reset. New configuration settings will be loaded. It resets all tags in the antenna field.

Command	Data
'x'	None

Answer	Description
<i>MultiISO 1.0'</i>	

Reset Timing

The power up timing depends on environmental conditions such as voltage ramp up. For handheld devices the timing can vary based on the charge state of the battery.

```
C++
#include "CFReaderLib.h"

char buffer[514];
memset(buffer, 0x00, 514);

RDR_SendCommandGetData("x", "", buffer);

RDR_EmptyCommRcvBuffer();
```

4.4.2.9 Get Version

This command returns the current version of the reader module.

Command	Data
'v'	None

Answer	Description
<i>MultiISO 1.0'</i>	

```
C++
#include "CFReaderLib.h"

char buffer[514];
memset(buffer, 0x00, 514);

RDR_SendCommandGetData("v", "", buffer);
// buffer had the current version of the reader module

RDR_EmptyCommRcvBuffer();
```

4.4.3 Reference and Function List for C#

Structure

presetSettings

```
public struct presetSettings
{
    public int baudRate;
    public int protocol;
}
```

readerConfig

```
public struct readerConfig
{
    public int baudRate;
    public byte protocol;
    public byte stationID;
}
```

4.4.3.1 RDR_OpenComm

The **RDR_OpenComm** function initializes and opens a specified communication device.

```
public byte RDR_OpenComm(string device, byte autodetect,
ACG_CFReader.presetSettings settings);
```

Parameters

Device

0 terminated string containing the communication device.

"com1", "com2"...

Only serial devices are supported.

The device buffer size has the range of 10 bytes.

autodetect

0 : The auto detection of the baud rate and the protocol is deactivated. Values from settings are used.

1 : The normal auto detection of the baud rate and the protocol is activated.

2 : The fast auto detection of the baud rate and the protocol is activated. Older reader devices may not be found.

4 : If this bit is additionally set, not common baud rates are disabled (19200, 38400, 230400). Use this to speed up the autodetect procedure.

If MSB (80h) is set the power management for WindowsCE will be deactivated.

settings

Pointer to the structure of preset settings.

Return Values

If the function succeeds, the return value is a handle of the opened communication device.

If the communication port is opened on a Windows CE device successfully the return value is one.

If the function fails, the return value is zero.

C#

```
using CFReaderDLLWrapper;

public ACG_CFReader dllACG;
dllACG = new ACG_CFReader();

ACG_CFReader.presetSettings settings;
settings.baudRate = 9600;
settings.protocol = 0;

int i = dllACG.RDR_OpenComm("MOC1", 6, settings);

if (i == 0)
    MessageBox.Show("OpenComm failed");
```

4.4.3.2 RDR_OpenReader

The **RDR_OpenReader** function opens a reader on specified communication device with specified ID.

```
public byte RDR_OpenReader(byte id, short knownReader);
```

Parameters

id

Valid station ID of the reader.

knownReader

0 : auto detect reader type

1 : unknown reader type

2 : Mifare 0.14e / f

3 : Mifare 0.15d

4 : Mifare 1.0

6: ISO 0.9v
8: MULTITAG 0.10b
10: MULTITAG 0.12a
11: MULTITAG 0.12bg
12: Mifare 0.15e
15: all Bootloader 0.x / 1.x
16: all Dual 2.x
18: all MULTITAG 1.x
19: all LFX 1.x
20: all MultISO 1.x

Return Values

If the function succeeds, the return value is a handle of the opened reader.

If the function fails, the return value is zero.

C#

```
using CFReaderDLLWrapper;

public ACG_CFReader dllACG;
dllACG = new ACG_CFReader();

if(dllACG.RDR_OpenReader(1, 0) == 0)
{
    MessageBox.Show("OpenReader Failed");
}
```

4.4.3.3 RDR_SendCommand

The **RDR_SendCommand** function sends a command to a reader specified by its handle.

```
public void RDR_SendCommand(string command, string data);
```

Parameters

command

0 terminated string with the command.

data

0 terminated string containing the data.

Return Values

None

Remarks

Unsupported commands were sent to the reader without modifications.

C#

```
using CFReaderDLLWrapper;

public ACG_CFReader dllACG;
dllACG = new ACG_CFReader();

string buffer = "";

dllACG.RDR_SendCommand("c", "");

int nCount = 0;
while(nCount++ < 100)
{
    dllACG.RDR_GetData(ref buffer);
}
```

```
dllACG.RDR_AbortContinuousReadExt();  
  
dllACG.RDR_EmptyCommRcvBuffer();
```

4.4.3.4 RDR_SendCommandGetData

The **RDR_SendCommandGetData** function sends a command to a reader specified by its handle and receives data.

```
public void RDR_SendCommandGetData(string command, string data, ref string  
buffer);
```

Parameters

command

0 terminated string with the command.

data

0 terminated string containing the data.

buffer

A buffer of the return value. The buffer size has the range of 514 bytes.

Return Values

None

```
C#  
using CFReaderDLLWrapper;  
  
public ACG_CFReader dllACG;  
dllACG = new ACG_CFReader();  
  
string buffer = "";  
  
// "ot" Select Tag Type command, 'o' Select tag type and 't' is 'activate all  
type'  
dllACG.RDR_SendCommandGetData("o", "t", ref buffer);  
  
dllACG.RDR_EmptyCommRcvBuffer();
```

4.4.3.5 RDR_SetReaderConfig

The **RDR_SetReaderConfig** function sets and applies the reader with specified configuration. The communication device even adapted to this new settings.

```
public void RDR_SetReaderConfig(ref ACG_CFReader.readerConfig settings);
```

Parameters

setting

structure of the configuration data. For structure definition of reader configuration see header files.

Remarks

This function does not work if unknown reader type is used.

Use this function for only one reader at the same time. Otherwise the communication to other devices will fail.
Be sure the used interface supports the baud rate.

```
C#  
using CFReaderDLLWrapper;  
  
public ACG_CFReader dllACG;  
dllACG = new ACG_CFReader();
```

```
ACG_CFReader.readerConfig settings = dllACG.RDR_GetReaderConfig();
settings.protocol = 0;
dllACG.RDR_SetReaderConfig(ref settings);
```

4.4.3.6 RDR_GetReaderConfig

The **RDR_GetReaderConfig** function returns current reader configuration.

```
public ACG CFReader.readerConfig RDR_GetReaderConfig();
```

Parameters

None

Return value

The function returns a structure of configuration data of a reader.

```
C#
using CFReaderDLLWrapper;

public ACG_CFReader dllACG;
dllACG = new ACG_CFReader();

ACG_CFReader.readerConfig settings = dllACG.RDR_GetReaderConfig();
settings.protocol = 0;

dllACG.RDR_SetReaderConfig(ref settings);
```

4.4.3.7 RDR_GetData

The **RDR_GetData** function receives data from a reader specified with its handle.

```
public void RDR_GetData(ref string buffer);
```

Parameters

buffer

A buffer of the return value. The buffer size has the range of 514 bytes.

```
C#
using CFReaderDLLWrapper;

public ACG_CFReader dllACG;
dllACG = new ACG_CFReader();

string buffer = "";

dllACG.RDR_SendCommand("c", "");

int nCount = 0;
while(nCount++ < 100)
{
    dllACG.RDR_GetData(ref buffer);
}

dllACG.RDR_AbortContinuousReadExt();

dllACG.RDR_EmptyCommRcvBuffer();
```

4.4.3.8 RDR_AbortContinuousRead

The **RDR_AbortContinuousRead** function aborts the continuous read command in ASCII mode and empties the buffer.

```
public void RDR_EmptyCommRcvBuffer();
```

Parameters

None

Return Values

None

C#

```
using CFReaderDLLWrapper;

public ACG_CFReader dllACG;
dllACG = new ACG_CFReader();

string buffer = "";

dllACG.RDR_SendCommand("c", "");

int nCount = 0;
while(nCount++ < 100)
{
    dllACG.RDR_GetData(ref buffer);
}

dllACG.RDR_AbortContinuousReadExt();

dllACG.RDR_EmptyCommRcvBuffer();
```

4.4.3.9 RDR_EmptyCommRcvBuffer

The **RDR_EmptyCommRcfBuffer** function empties the receive buffer of the communication device.

```
public void RDR_EmptyCommRcvBuffer();
```

Parameters

None

Return Values

None

C#

```
using CFReaderDLLWrapper;

public ACG_CFReader dllACG;
dllACG = new ACG_CFReader();

string buffer = "";

dllACG.RDR_SendCommand("c", "");

int nCount = 0;
while(nCount++ < 100)
{
    dllACG.RDR_GetData(ref buffer);
}
```

```
}  
  
dllACG.RDR_AbortContinuousReadExt();  
  
dllACG.RDR_EmptyCommRcvBuffer();
```

4.4.4 Command List for C#

4.4.4.1 Antenna power on/off

This command controls the antenna power. It can be used to decrease the power consumption of the reader.

Command	Data
'pon'	Switch reader on.
'poff'	Put reader in standby mode.

Answer	Description
'P'	Positive acknowledge

Power on

The reader leaves standby mode and is ready for the next command. Sending a tag command (i.e. select, continuous read) the reader is powered up.

Power off

The reader enters standby mode. Power consumption is decreased. All tags in the antenna field are powered off and reset. Standby mode is only entered manually. To switch off the whole unit, pin 16 (Enable) has to be set to logic low.

```
C#
using CFReaderDLLWrapper;

public ACG_CFReader dllACG;
dllACG = new ACG_CFReader();

string buffer = "";

if(bAntenna)
    dllACG.RDR_SendCommand("pon", "");
else
    dllACG.RDR_SendCommand("poff", "");

dllACG.RDR_EmptyCommRcvBuffer();
```

4.4.4.2 Set tag type

This command sets up the reader for a specific tag type. The continuous read function will speed up because only this type of tag is addressed. After a reset, the reader starts as defined in its start-up configuration.

't' which is activatve to all tags is default. Read/Write speed can be faster by setting a tag type to a specific one.

Command	Data
'o'	ISO type (1 byte)
	'a' ISO 14443 Type A
	'b' ISO 14443 Type B
	'd' ICODE UID
	'e' ICODE EPC
	'i' ICODE
	's' SR176
	't' activate all tags
	'v' ISO 15693

Answer	Description
'OA'	String of tag type
'OB'	
'OD'	
'OE'	

'OI'	
'OS'	
'OT'	
'OV'	

```
C#
using CFReaderDLLWrapper;

public ACG_CFReader dllACG;
dllACG = new ACG_CFReader();

string buffer = "";

// "ot" Select Tag Type command, 'o' Select tag type and 't' is 'activate all type'
dllACG.RDR_SendCommandGetData("o", "t", ref buffer);

dllACG.RDR_EmptyCommRcvBuffer();
```

4.4.4.3 Select

This command selects a single card in the antenna field. It can only be used in single tag mode. If successfully executed, the command returns the UID of the selected card. The reader detects the length of the UID automatically.

Command	Data
's'	None

Answer	Description
Data	Serial number
'N'	Error : No Tag in the field

Select a single tag

No previous continuous read is required. The command executes an automatic field reset.

Multiple tags

This command is designed for fast access of a single tag in the field. If multiple cards are used the 'm' instruction has to be used instead.

```
C#
using CFReaderDLLWrapper;

public ACG CFReader dllACG;
dllACG = new ACG_CFReader();

string buffer = "";

dllACG.RDR_SendCommandGetData("s", "", ref buffer);

dllACG.RDR_EmptyCommRcvBuffer();
```

4.4.4.4 Multi-Tag Selection / List

This command detects several tags at the same time. It replaces the fast select command ('s') in multiple tag surroundings. The Multi-Tag List command lists all tags with their serial numbers. Use the Multi-Tag Select command to select a single tag. Each tag has to be selected separately.

Command	Data
'm'	Serial number (n bytes). <CR> (1 byte)

Answer	Description
Data	Serial number
'N'	Error : No Tag in the field

Multi-tag list

Sending a as the first parameter, the reader returns a list of all tags present in the antenna field. In the end the total number of tags detected is returned.

Reading distance

Each card needs a specific amount of power. The reader always provides the same power level. Therefore, the reading distance will decrease if more tags are present. Basically, the reading distance depends on the tag, the antenna and the tuning of the antenna.

Multi-tag select

Using the serial number with as parameter, the corresponding tag will be selected. High-level interactions can be performed addressing only this card. All other tags remain silent.

Attention

It could be possible that ISO 15693 tags are interfered from ISO14443 type B and SR 176 tags. In this case the ISO 15693 tag will always answers on a multi list command even there had been no field reset before. In this case deactivate ISO 14443 B and SR 176 tags.

```
C#
using CFReaderDLLWrapper;

public ACG_CFReader dllACG;
dllACG = new ACG_CFReader();

string buffer = "";
char strDataCR = 0x0D; // <CR>
dllACG.RDR_SendComand("m", &cDataCR );

memset(buffer, 0x00, 514);
RDR_GetData(buffer);

int nTag = atoi(buffer);

for(int i = 0; i < nTag; i ++)
{
    memset(buffer, 0x00, 514);
    RDR_GetData(buffer);
    // buffer had a Serial
}

RDR_EmptyCommRcvBuffer();
```

4.4.4.5 Read block

This command reads a data block on a card. The size of the returned data depends on the tag used. The block address range depends on the tag as well.

Command	Data
'r'	Block address (1 byte), valid range 00h – 40h
'rb'	Block address (1 byte)

Answer	Description
Data	data block (depends on tag type)
'F'	Error: read failure
'N'	Error : No Tag in the field
'O'	Error: Operation mode failure

'R'	Error: Out of range
-----	---------------------

Read failure 'F'

This error is returned if either the reader receives bad data or the block address exceeds the block address range of the sector.

No tag in field 'N'

The The tag does not respond. There is either no tag present or addressed.

Operation mode failure 'O'

The presented tag is not ISO14443 type A, SR 176, ICode, ICode-UID and ISO 15693 compliant. For ISO 14443 type A only mifare® tags are supported

Out of range failure 'R'

The block address of the 'r' command is higher than 40h. The block address of the 'r' command conflicts with other commands, therefore the block address has to be limited to 40h. Use the 'rb' command instead.

C#

```
using CFReaderDLLWrapper;

public ACG_CFReader dllACG;
dllACG = new ACG_CFReader();

string buffer = "";
dllACG.RDR_SendCommandGetData("s", "", ref buffer);
// buffer have a serial number of tag

dllACG.RDR_EmptyCommRcvBuffer();

dllACG.RDR_SendCommandGetData("rb", "01", ref buffer);
// buffer have the 01 block data

dllACG.RDR_EmptyCommRcvBuffer();
```

4.4.4.6 Continuous Read

The reader device reads and displays serial numbers continuously while one or more tags remain in the field. This command stops if any character is sent to the reader module. The reader module returns the character 'S' (53h). The reader supports different tag types at the same time. To increase the reading performance switch to a single tag mode. If more than one tag of the same type should be detected at the same time, the Multitag flag must be activated. The response data length depends on the tag type.

Command	Data
'c'	None

Answer	Description
Data	Serial number (n bytes)
'N'	Error :No Tag in the field (only binary protocol)

Multitag continuous read mode

If the Multitag flag is set in the Protocol Configuration (PCON) register the reader reads multiple tags continuously.

Auto start

The continuous read mode is started automatically in ASCII mode. The auto start flag must be set in the PCON register.

Noisy Environment

If the Noisy Environment flag is set, the continuous read mode can only be aborted with the '.' character. This is only valid in ASCII mode.

Binary mode

This command is fully supported in binary protocol mode except the test continuous read command and the noisy environment flag.

Do not use this command on bus system environment in binary mode, because the continuous read mode will take possession of the bus system.

Simple access control applications

Serial numbers are always sent plain. Data encryption is activated after a successful login. For simple access control applications the use read-only blocks for the identification of the tag is recommended. Reading any block (even the manufacturer block) of the transponder will increase your security.

```
C#
using CFReaderDLLWrapper;

public ACG_CFReader dllACG;
dllACG = new ACG_CFReader();

string buffer = "";

dllACG.RDR_SendCommand("c", "");

int nCount = 0;
while(nCount++ < 100)
{
    dllACG.RDR_GetData(ref buffer);
}

dllACG.RDR_AbortContinuousReadExt();

dllACG.RDR_EmptyCommRcvBuffer();
```

4.4.4.7 Write block

This command writes data to a block. A read is done automatically after every write to ensure correct writing.

Command	Data
'w'	Block address (1 byte), valid range 00h – 40h. Data (n bytes)
'wb'	Block address (1 byte) Data (n bytes)

Answer	Description
Data	data block (depends on tag type)
'F'	Error: read failure
'N'	Error : No Tag in the field
'O'	Error: Operation mode failure
'R'	Error: Out of range

Write failure 'F'

This error is displayed if bad transmission conditions are given. If the block address exceeds the physical number of blocks of a tag, this error is shown.

No tag error 'N'

This error is returned if no tag is present or the card does not respond.

Operation mode failure 'O'

The presented tag is not ISO14443 type A, SR 176, ICode, ICode-UID and ISO 15693 compliant. For ISO 14443 type A only mifare® tags are supported.

Out of range failure 'R'

The block address of the 'w' command is higher than 40h. The block address of the 'w' command conflicts

with other commands, therefore the block address has to be limited to 40h. Use the 'wb' command instead.

```
C#
using CFReaderDLLWrapper;

public ACG CFReader dllACG;
dllACG = new ACG_CFReader();

string buffer = "";

dllACG.RDR_SendCommandGetData("s", "", ref buffer);
// buffer have a serial number of tag

memset(buffer, 0x00, 514);
dllACG.RDR_EmptyCommRcvBuffer();

dllACG.RDR_SendCommandGetData("wb", "0511223344", ref buffer);
// Writes data 11223344 on block 05.

dllACG.RDR_EmptyCommRcvBuffer();
```

4.4.4.8 Reset

This command executes a power on (software) reset. New configuration settings will be loaded. It resets all tags in the antenna field.

Command	Data
'x'	None

Answer	Description
<i>MultiISO 1.0'</i>	

Command

Answer

Reset Timing

The power up timing depends on environmental conditions such as voltage ramp up. For handheld devices the timing can vary based on the charge state of the battery.

```
C#
using CFReaderDLLWrapper;

public ACG_CFReader dllACG;
dllACG = new ACG_CFReader();

string buffer = "";

dllACG.RDR_SendCommandGetData("x", "", ref buffer);

dllACG.RDR_EmptyCommRcvBuffer();
```

4.4.4.9 Get Version

This command returns the current version of the reader module.

Command	Data
---------	------

'v'	None
Answer	Description
<i>MultiISO 1.0'</i>	

```
C#
using CFReaderDLLWrapper;

public ACG_CFReader dllACG;
dllACG = new ACG_CFReader();

string buffer = "";

dllACG.RDR_SendCommandGetData("v", "", ref buffer);
// buffer had the current version of the reader module

dllACG.RDR_EmptyCommRcvBuffer();
```


4.5 Scanner 1D

This section provides description of the functions and DLLs which are used to manage 1D scanner module.

Required Files

For C++

Required header:

```
KScnaBar.h  
Option.h  
Option_defs.h
```

Required lib:

```
KScanBar.lib
```

Required DLL:

```
KScanBar.dll
```

For C#

Required DLL:

```
KScanBar.dll  
KScanBarNet.dll
```

Supported Product

M3 ORANGE with 1D scanner that uses software decoder.

4.5.1 Reference and Function List for C++

Definitions

Status Return value

Please refer to the below table for the status value definition.

Status Value Definition	Code	Meaning
TRUE	1	Success
FALSE	0	General Error

Structure

1D Scanner

```
typedef struct tagKSCANREAD {
    int                nSize;
    unsigned            nTimeInSeconds;
    unsigned long       dwFlags;
    unsigned long       dwReadType;
    int                nMinLength;
    int                nSecurity;
    KS_FN_CALLBACK      fnCallBack;
    LPVOID              pUserData;
    LPVOID              pReadEx;
    int                out_Status;
    int                out_Type;
    char                out_Barcode[2711];
} KSCANREAD;

typedef KSCANREAD * PKSCANREAD;

typedef struct tagKSCANREADEX {
    int                nI2of5MinLength;
    int                nI2of5MaxLength;
    int                nCodabarMinLength;
    int                nCodabarMaxLength;

    #if defined(QUIETZONE_CHECK_OPTION_APPEND) && !defined(ALWAYS_IGNORE_QUIETZONE)
        unsigned long dwIgnoreQZCheck;    // Bit : Code128, EAN,
    #endif

    HWND              hwnd;
    DWORD             UserMsg;
} KSCANREADEX;

typedef KSCANREADEX* PKSCANREADEX;

typedef struct tagKSCANREADEX2 {

    char Signature[8];          //"KSCANEX2"

    HWND              hwnd;
    DWORD             UserMsg;
    DEC_UPCEAN        UpcA;
    DEC_UPCEAN        UpcE;
    DEC_UPCEAN        Ean13;
    DEC_UPCEAN        Ean8;
    DEC_CODE39         Code39;
    DEC_CODE128        Code128;
    DEC_CODE93         Code93;
    DEC_CODE35         Code35;
    DEC_CODE11         Code11;    // CheckDigit:0, Disable, 1:1 Digit, 2:2 Digit
```

```

DEC_CODE25      Code25;
DEC_CODABAR     Codabar;
DEC_MSI         Msi;
DEC_PLESSEY     Plessey;
DEC_GS1         Gs1;
DEC_GS1_LIMITED Gs1Limited;
DEC_GS1_EXPANDED Gs1Expanded;
DEC_TELEPEN     Telepen;

//DATA_FORMAT DataFormat;

ABLE           XmitAIMID; //2009.09.30

} KSCANREADEX2;

typedef KSCANREADEX2* PKSCANREADEX2;

typedef struct _MCBarCodeType
{
    // BARCODE 26
    BYTE bMC_UPCA;
    BYTE bMC_UPCA_ADDON;
    BYTE bMC_UPCE;
    BYTE bMC_EAN13;
    BYTE bMC_EAN13_ADDON;
    BYTE bMC_BOOKLAND;
    BYTE bMC_EAN8;
    BYTE bMC_CODE39;
    BYTE bMC_CODE32;
    BYTE bMC_PZN;
    BYTE bMC_CODE128;
    BYTE bMC_UCCEAN128;
    BYTE bMC_CODE93;
    BYTE bMC_CODE35;
    BYTE bMC_CODE11;
    BYTE bMC_I2OF5;
    BYTE bMC_CODE25_ITF14;
    BYTE bMC_CODE25_MATRIX;
    BYTE bMC_CODE25_DLOGIC;
    BYTE bMC_CODE25_INDUSTRY;
    BYTE bMC_CODE25_IATA;
    BYTE bMC_CODABAR;
    BYTE bMC_COUPON;
    BYTE bMC_MSI;
    BYTE bMC_PLESSEY;
    BYTE bMC_GS1;
    BYTE bMC_GS1_LIMITED;
    BYTE bMC_GS1_EXPANDED;
}BarCodeType,* PBarCodeType;

typedef struct _MCBarCodeTypeEx
{
    // BARCODE 26
    BYTE bMC_UPCA;
    BYTE bMC_UPCA_ADDON;
    BYTE bMC_UPCE;
    BYTE bMC_EAN13;
    BYTE bMC_EAN13_ADDON;
    BYTE bMC_BOOKLAND;
    BYTE bMC_EAN8;
    BYTE bMC_CODE39;
    BYTE bMC_CODE32;
    BYTE bMC_PZN;
    BYTE bMC_CODE128;

```

```

    BYTE bMC_UCCEAN128;
    BYTE bMC_CODE93;
    BYTE bMC_CODE35;
    BYTE bMC_CODE11;
    BYTE bMC_I2OF5;
    BYTE bMC_CODE25_ITF14;
    BYTE bMC_CODE25_MATRIX;
    BYTE bMC_CODE25_DLOGIC;
    BYTE bMC_CODE25_INDUSTRY;
    BYTE bMC_CODE25_IATA;
    BYTE bMC_CODABAR;
    BYTE bMC_COUPON;
    BYTE bMC_MSI;
    BYTE bMC_PLESSEY;
    BYTE bMC_GS1;
    BYTE bMC_GS1_LIMITED;
    BYTE bMC_GS1_EXPANDED;
    BYTE bMC_TELEPEN;
    BYTE bMC_TYPE1;
    BYTE bMC_TYPE2;
    BYTE bMC_TYPE3;
    BYTE bMC_TYPE4;
    BYTE bMC_TYPE5;
    BYTE bMC_TYPE6;
    BYTE bMC_TYPE7;
}BarCodeTypeEx,* PBarCodeTypeEx;

typedef struct _MCReadOption
{
    BYTE bMC_WIDESCANANGLE;
    BYTE bMC_RETURNCHECK;
    BYTE bMC_ERRORCHECK;
    BYTE bMC_HIGHFILTERMODE;
}ReadOption,* PReadOption;

typedef struct _MCModuleOption
{
    int nMC_TimeOutSec;
    int nMC_MinLen;
    int nMC_SecurityLevel;
}ModuleOption,* PModuleOption;

```

Parameters

M3M_SetParameter() functions use parameters defined as below.

```

#define KSCAN_RET_TYPE_EAN_13          0
#define KSCAN_RET_TYPE_EAN_8          1
#define KSCAN_RET_TYPE_UPCA           2
#define KSCAN_RET_TYPE_UPCE           3
#define KSCAN_RET_TYPE_CODE_39        4
#define KSCAN_RET_TYPE_ITF_14         5
#define KSCAN_RET_TYPE_CODE_128       6
#define KSCAN_RET_TYPE_CODE_I25       7
#define KSCAN_RET_TYPE_CODA_BAR       8
#define KSCAN_RET_TYPE_UCCEAN_128     9
#define KSCAN_RET_TYPE_CODE_93        10
#define KSCAN_RET_TYPE_CODE_35        11
#define KSCAN_RET_TYPE_PDF417         12
#define KSCAN_RET_TYPE_MACRO_PDF417   13
#define KSCAN_RET_TYPE_BOOKLAND       14
#define KSCAN_RET_TYPE_MSI            15
#define KSCAN_RET_TYPE_PZN            16

```

```

#define KSCAN_RET_TYPE_PLESSEY 17
#define KSCAN_RET_TYPE_MACRO_PDF417_INC 18
#define KSCAN_RET_TYPE_MACRO_PDF417_ERROR 19
#define KSCAN_RET_TYPE_CODE25_MATRIX 20
#define KSCAN_RET_TYPE_CODE25_DLOGIC 21
#define KSCAN_RET_TYPE_CODE25_INDUSTY 22
#define KSCAN_RET_TYPE_CODE25_IATA 23
#define KSCAN_RET_TYPE_CODE25_GTIN14 24
#define KSCAN_RET_TYPE_CODE25_DPL 25
#define KSCAN_RET_TYPE_CODE25_DPI 26
#define KSCAN_RET_TYPE_CODE11 27
#define KSCAN_RET_TYPE_CODE32 28
#define KSCAN_RET_TYPE_COUPONCODE 29
#define KSCAN_RET_TYPE_CODABLOCK_A 30
#define KSCAN_RET_TYPE_CODABLOCK_F 31
#define KSCAN_RET_TYPE_GS1 32 // RSS_14
#define KSCAN_RET_TYPE_GS1_LIMITED 33 // RSS_LIMITED
#define KSCAN_RET_TYPE_GS1_EXPANDED 34 // RSS_EXPENDED
#define KSCAN_RET_TYPE_STANDARD2OF5 35
#define KSCAN_RET_TYPE_TELEPEN 36
#define KSCAN_RET_TYPE_UNKNOWN 0xFF

```

Enum

```

typedef enum {
    DISABLE=0,
    ENABLE
}ABLE;

typedef enum {
    FULL_ASCII=0,
    STD_ASCII
}XMIT_ASCII;

typedef enum {
    NO_Supp=0,
    WITH_OR_WITHOUT
}SUPP;

typedef enum {
    XMIT_NUMBER=0,
    NO_XMIT_NUMBER
}XMIT_NUM_SYSTEM;

typedef enum {
    XMIT_CHECK_DIGIT=0,
    NO_XMIT_CHECK_DIGIT
}XMIT_CHECK_CHAR;

typedef enum {
    NO_XMIT=0,
    XMIT
}XMIT_STARTSTOP;

typedef enum {
    AS_UPCA=0,
    AS_EAN13,
    AS_UPCE,
    AS_EAN8,
    AS_CODE32,
    AS_UCCEAN128,
    AS_BOOKLAND
}FORMAT;

```

```

typedef enum {
    ANGLE_WIDE=0,
    ANGLE_NARROW
}ENGINE_ANGLE;

typedef enum {
    FILTER_WIDE=0,
    FILTER_NARROW
}ENGINE_FILTER;

typedef enum {
    MODULO10 = 0,
    MODULO11,
    MODULO1010,
    MODULO1110
}MOD_METHOD;

typedef enum {
    CODE25KIND_INTER = 0x01,    // 1,
    CODE25KIND_MATRIX = 0x02,   // 2,
    CODE25KIND_DLOGIC = 0x04,   // 4,
    CODE25KIND_INDUSTY = 0x08,  // 8,
    CODE25KIND_IATA = 0x10,     // 16,
    CODE25KIND_ITF14 = 0x20,    // 32,
}CODE25_KIND;

// Jiyong modified - spain
typedef enum {
    DIGIT0 = 0,
    DIGIT1,
    DIGIT2
}CHECK_DIGIT;

```

4.5.1.1 KScanOpen

The **KScanOpen** function opens a COM port to a Scan Module, and performs initialization.

```

BOOL KScanOpen(
    int CommNumber = 0,
    BOOL CommDetect = TRUE,
    DWORD BaudRate = 230400,
    BOOL BaudDetect = FALSE,
    DWORD ExFlags = 0
);

```

Parameters

CommNumber

Specifies the serial port number to be opened. Range: 0~16. When 0 is specified automatic scan of Com port is enabled. The default is 0.

CommDetect

Specifies automatic scanning of Com port number when Scan Module was not present at CommNumber. The order of scanning is: CommNumber, 1...16. The default is TRUE.

BaudRate

Specifies baud rate for Com port. The default is 230400.

BaudDetect

Specifies automatic scanning of baud rate when Scan Module was not found using BaudRate specified. The order of scanning is: BaudRate, 230400, CBR_38400, CBR_57600. The default is TRUE.

ExFlags

Specifies optional flags for future compatibility. Must be set to zero.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

It is necessary to pair this function with KScanClose for proper operation and resource management.

Depending on operating system, Com port hardware, Com port device driver and their implementation, this function may take a long time (2+ seconds) to successfully open and initialize the Scan Module. It is recommended to use a verified numbers for *CommNumber* and *BaudRate* to minimize execution time. It should also be noted that Com port number designated by operating system can be changed due to configuration changes usually caused by reinstallation and/or addition/deletion of hardware/software. It is recommended to utilize *CommDetect* initially, and to reuse detected Com port number for further operation in the same session.

The maximum Baud Rate defined is 115,200 bps on desktop operating systems. Depending on the platform and its operating system implementation, the maximum Baud Rate defined varies. For baud rates beyond defined maximum, there are a few techniques employed. Please refer to the platform and serial device driver reference.

C++

```
BOOL bRet = KScanOpen(6, FALSE, 230400, FALSE, NULL);
if(!bRet)
{
    ::MessageBox(NULL, L"ScanOpen Failed", NULL, MB_TOPMOST);
    return FALSE;
}
```

4.5.1.2 KScanClose

The **KScanClose** function closes the COM port to a KoamTac Scan Module, and may put the Scan Module into sleep mode.

BOOL KScanClose();

Parameters

None

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

It is necessary to pair Close() with KScanOpen for proper operation and resource management.

C++

```
BOOL bRet = FALSE;
for(i=0;i<3;i++)
{
    bRet = KScanClose()
    if(bRet)
        break;
    Sleep(100);
}

if(!bRet)
{
    ::MessageBox(NULL, L"ScanClose Failed", NULL, MB_TOPMOST);
    return FALSE;
}
```

```
}
```

4.5.1.3 KScanRead

The **KScanRead** function reads a barcode from the Scan Module.

```
BOOL KScanRead();
```

Parameters

None

Return Values

If the function succeeds, the return value is nonzero.

If the function fails or times out, the return value is zero.

Remarks

This function is valid only after a successful call to KScanOpen.

```
C++
void CM3ScanTestDlg::ScanReadNB()
{
    BOOL      bRet;

    if(m_bKeyFlag == FALSE)
    {
        if (m_bReading)
        { // Reading already in progress, now cancel it.
            bRet = KScanReadCancel();

            if (bRet)
            {
                m_bReading = FALSE;
                return;
            }
            else
            {
                ::MessageBox(NULL, KScanGetLastErrorMsg(), NULL, NULL);
            }
        }
        m_bKeyFlag = TRUE;

        if(m_bCon)
            m_bReading = TRUE;
        else
            m_bReading = FALSE;

        bRet = KScanRead();
        if (!bRet)
        {
            m_bReading = FALSE;
        }
    }
}
```

4.5.1.4 KScanReadCancel

The **KScanReadCancel** function cancels KScanRead function that was initiated as non-blocking operation. Its use is discouraged.

```
BOOL KScanReadCancel();
```

Parameters

None.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails or times out, the return value is zero.

Remarks

This function is valid only after a successful non-blocking call to KScanRead. Please note that this function is not applicable when read operation was initiated as blocking read.

Please refer to included sample program for details of non-blocking operations. Non-blocking operation is a delicate process that provides applications with more control how the scan module reads barcodes. It is recommended that users take advantage of simple blocking read operation unless non-blocking operation is absolutely necessary.

The reading process makes uses of multiple threads and synchronization primitives. An orderly termination via the call back function is always preferred. The use of **ReadCancel** function should be avoided to terminate the read. Using the **ReadCancel** function multiple time to terminate the read operation may lead to unstable application.

ReadCancel tries to maintain the system integrity while terminating the threads and restoring the synchronization primitives. Only when the these primitives do not respond, does it forcibly terminate them. The function thus sometimes takes up to 1 second to complete its tasks.

```
C++
void CM3ScanTestDlg::ReadCancel()
{
    KScanReadCancel();
}
```

4.5.1.5 SetOption

The **SetOption** function set barcode symbol the Scan Module.

```
void SetOption(
    PKSCANREAD    pRead
);
```

Parameters

pRead

Specifies a pointer to the KSCANREAD Structure.

Return Values

None

Remarks

None

4.5.1.6 SetBarCodeType

The **SetBarCodeType** function Set Structure of Symbology.

```
void SetBarCodeType(ref BarCodeType pBCT)
```

Parameters

BarCodeType

Specifies pointer to the KScanBar Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

4.5.1.7 GetBarCodeType

The **GetBarCodeType** function Set Structure of Symbology.

```
void GetBarCodeType (BarCodeType pBCT)
```

Parameters

GarCodeType

Specifies pointer to the KScanBar Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner

4.5.1.8 SetReadOption

The **SetReadOption** function set Structure of ReadOption.

```
void SetReadOption(ref ReadOption pRDO)
```

Parameters

ReadOption

Specifies pointer to the KScanBar Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

4.5.1.9 GetReadOption

The **GetReadOption** function gets Structure of ReadOption.

```
void GetReadOption(ReadOption pRDO)
```

Parameters

ReadOption

Specifies pointer to the KScanBar Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

4.5.1.10 SetModuleOption

The **SetModuleOption** function set Structure of ModuleOption

```
void SetModuleOption(ref ModuleOption pMDO)
```

Parameters

ModuleOption

Specifies pointer to the KScanBar Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

4.5.1.11 GetModuleOption

The **GetModuleOption** function gets Structure of ModuleOption.

```
void GetModuleOption(ModuleOption pMDO)
```

Parameters

ModuleOption

Specifies pointer to the KScanBar Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

4.5.1.12 SetReturn

The **SetReturn** functions to set read method.

```
void SetReturn(  
    int          type  
    KS_FN_CALLBACK fnCallBack  
) ;
```

Parameters

type

Set to return Method.

fnCallBack

Specifies call back function for CallBack read Method. If type is not RETURN_CALLBACK, fnCallBack is NULL.

Return Values

None

Remarks

type value

RETURN_MESSAGE - Return by Message Type (WM_SCANDATA)

RETURN_CALLBACK - Return by CallBack function

RETURN_KEYMSG - Return by Key Message

4.5.1.13 SetContinue

The **SetContinue** functions to set continuous scan mode.

```
void SetScanContinue(  
    int    nContinue  
    BOOL   bForever  
) ;
```

Parameters

nContinue

Set to continuous mode.

bForever

Scan read continue or not when the scan time is out.

Return Values

None

Remarks

None

4.5.1.14 KScanGetVersionInfo

The **KScanGetVersionInfo** function returns a pointer to a string with version information associated with the current port.

```
LPCTSTR KScanGetVersionInfo();
```

Parameters

None

Return Values

The function returns a pointer to a string with version information associated with the current port. If no Scan Module is open, the return information excludes hardware version information.

Remarks

None

4.5.1.15 KScanGetLastErrorMsg

The **KScanGetLastErrorMsg** function returns a pointer to a string with description of the last error occurred in KScanBar.dll.

```
LPCTSTR KScanGetLastErrorMsg();
```

Parameters

None

Return Values

The function returns a pointer to a string with description of the last error occurred in KScanBar.dll.

Remarks

None

4.5.1.16 KScanSetBarcodeString

The **KScanSetBarcodeString** function gets barcode name

```
void KScanSetBarcodeString(int nType)
```

Parameters

nType

Specifies integer to the barcode name.

Return Values

None

Remarks

Use after successful read function.

4.5.1.17 KScanGetScanData

The **KScanGetScanData** function gets barcode data.

```
void KScanGetScanData(TCHAR *szBarData, TCHAR *szBarType)
```

Parameters

szBarData

Specifies pointer to the barcode data.

szBarType

Specifies pointer to the barcode typeReturn Values.

Return Values

None

Remarks

Use after successful read function.

4.5.2 Reference and Function List for C#

Definitions

Status Return value

Please refer to the below table for the status value definition.

Status Value Definition	Code	Meaning
TRUE	1	Success
FALSE	0	General Error

Structure

1D Scanner

```
public struct MCBARCODEType
{
    // BARCODE 26
    public bool bMC_UPCA;
    public bool bMC_UPCA_ADDON;
    public bool bMC_UPCE;
    public bool bMC_EAN13;
    public bool bMC_EAN13_ADDON;
    public bool bMC_BOOKLAND;
    public bool bMC_EAN8;
    public bool bMC_CODE39;
    public bool bMC_CODE32;
    public bool bMC_PZN;
    public bool bMC_CODE128;
    public bool bMC_UCCEAN128;
    public bool bMC_CODE93;
    public bool bMC_CODE35;
    public bool bMC_CODE11;
    public bool bMC_I2OF5;
    public bool bMC_CODE25_ITF14;
    public bool bMC_CODE25_MATRIX;
    public bool bMC_CODE25_DLOGIC;
    public bool bMC_CODE25_INDUSTRY;
    public bool bMC_CODE25_IATA;
    public bool bMC_CODABAR;
    public bool bMC_COUPON;
    public bool bMC_MSI;
    public bool bMC_PLESSEY;
    public bool bMC_GS1;
    public bool bMC_GS1_LIMITED;
    public bool bMC_GS1_EXPANDED;
}

public struct MCBARCODETypeEx
{
    public bool bMC_UPCA;
    public bool bMC_UPCA_ADDON;
    public bool bMC_UPCE;
    public bool bMC_EAN13;
    public bool bMC_EAN13_ADDON;
    public bool bMC_BOOKLAND;
    public bool bMC_EAN8;
    public bool bMC_CODE39;
    public bool bMC_CODE32;
    public bool bMC_PZN;
    public bool bMC_CODE128;
    public bool bMC_UCCEAN128;
    public bool bMC_CODE93;
```

```

    public bool bMC_CODE35;
    public bool bMC_CODE11;
    public bool bMC_I2OF5;
    public bool bMC_CODE25_ITF14;
    public bool bMC_CODE25_MATRIX;
    public bool bMC_CODE25_DLOGIC;
    public bool bMC_CODE25_INDUSTRIY;
    public bool bMC_CODE25_IATA;
    public bool bMC_CODABAR;
    public bool bMC_COUPON;
    public bool bMC_MSI;
    public bool bMC_PLESSEY;
    public bool bMC_GS1;
    public bool bMC_GS1_LIMITED;
    public bool bMC_GS1_EXPANDED;
    public bool bMC_TYPE1;
    public bool bMC_TYPE2;
    public bool bMC_TYPE3;
    public bool bMC_TYPE4;
    public bool bMC_TYPE5;
    public bool bMC_TYPE6;
    public bool bMC_TYPE7;
    public bool bMC_TYPE8;
    public bool bMC_TYPE9;
    public bool bMC_TYPE10;
}

public struct MCReadOption
{
    public bool bMC_WIDESCANANGLE;
    public bool bMC_RETURNCHECK;
    public bool bMC_ERRORCHECK;
    public bool bMC_HIGHFILTERMODE;
}

public struct MCModuleOption
{
    public int nMC_TimeOutSec;
    public int nMC_MinLen;
    public int nMC_SecurityLevel;
}

public struct MCBAROption_UPCA
{
    public bool bMC_UPCA_Enable;
    public bool bMC_UPCA_XNum;
    public bool bMC_UPCA_XCD;
    public bool bMC_UPCA_AS_EAN13;
    public bool bMC_UPCA_AddOn;
}

public struct MCBAROption_UPCE
{
    public bool bMC_UPCE_Enable;
    public bool bMC_UPCE_XNum;
    public bool bMC_UPCE_XCD;
    public int nMC_UPCE_Convert;
}

public struct MCBAROption_EAN13
{
    public bool bMC_EAN13_Enable;
    public bool bMC_BOOKLAND_Enable;
    public bool bMC_EAN13_XCD;
}

```

```

    public bool bMC_EAN13_AddOn;
}

public struct MCBarResult_EAN8
{
    public bool bMC_EAN8_Enable;
    public bool bMC_EAN8_XCD;
    public bool nMC_EAN8_AS_EAN13;
}

public struct MCBarResult_CODE39
{
    public bool bMC_CODE39_Enable;
    public bool bMC_CODE32_Enable;
    public bool bMC_PZN_Enable;
    public bool bMC_CODE39_CDV;
    public bool bMC_CODE39_XCD;
    public bool bMC_CODE39_FullASCII;
    public int nMC_CODE39_MinLen;
    public int nMC_CODE39_MaxLen;
}

public struct MCBarResult_CODE128
{
    public bool bMC_CODE128_Enable;
    public bool bMC_UCCEAN128_Enable;
    public int nMC_CODE128_MinLen;
    public int nMC_CODE128_MaxLen;
}

public struct MCBarResult_CODE93
{
    public bool bMC_CODE93_Enable;
    public int nMC_CODE93_MinLen;
    public int nMC_CODE93_MaxLen;
}

public struct MCBarResult_CODE35
{
    public bool bMC_CODE35_Enable;
}

public struct MCBarResult_CODE11
{
    public bool bMC_CODE11_Enable;
    public bool bMC_CODE11_XCD;
    public int nMC_CODE11_CDV;
    public int nMC_CODE11_MinLen;
    public int nMC_CODE11_MaxLen;
}

public struct MCBarResult_I2OF5
{
    public bool bMC_I2OF5_Enable;
    public bool bMC_ITF14_Enable;
    public bool bMC_MATRIX2OF5_Enable;
    public bool bMC_DLOGIG_Enable;
    public bool bMC_INDUSTRY_Enable;
    public bool bMC_IATA_Enable;
    public bool bMC_I2OF5_CDV;
    public bool bMC_I2OF5_XCD;
    public int nMC_I2OF5_MinLen;
    public int nMC_I2OF5_MaxLen;
}

```



```

public struct MCBAROption_CODABAR
{
    public bool bMC_CODABAR_Enable;
    public bool bMC_CODABAR_XSS;
    public int nMC_CODABAR_MinLen;
    public int nMC_CODABAR_MaxLen;
}

public struct MCBAROption_MSI
{
    public bool bMC_MSI_Enable;
    public bool bMC_MSI_CDV;
    public bool bMC_MSI_XCD;
    public int nMC_MSI_MinLen;
    public int nMC_MSI_MaxLen;
}

public struct MCBAROption_PLESSEY
{
    public bool bMC_PLESSEY_Enable;
    public bool bMC_PLESSEY_CDV;
    public bool bMC_PLESSEY_XCD;
    public int nMC_PLESSEY_MinLen;
    public int nMC_PLESSEY_MaxLen;
}

public struct MCBAROption_GS1
{
    public bool bMC_GS1_Enable;
    public bool bMC_GS1LIM_Enable;
    public bool bMC_GS1EXP_Enable;
}

public struct MCBAROption_TELEPEN
{
    public bool bMC_TELEPEN_Enable;
    public bool bMC_TELEPEN_OldStyle;
}

```

4.5.2.1 ScannerControl.ScanOpen

The **ScanOpen** function opens a COM port to Scan Module, and performs initialization.

```
int ScanOpen();
```

Parameters

None

Return Values

If the function succeeds, the return value is ERROR_NONE = 0

If the function fails, the return value is POWER_ON_ERROR = -1

If the function fails, the return value is PORT_OPEN_ERROR = -3

Remarks

It is necessary to pair this function with ScanClose() for proper operation and resource management.

If not call ScanClose() after a successful call to ScanInit(), next call to ScanInit() will fail

Depending on operating system, Com port hardware, Com port device driver and their implementation, this function may take a long time (2+ seconds) to successfully open and initialize the Scan Module.

```
C#
private KScanbarNet.ScannerControl ScanCtrl;
ScanCtrl = new ScannerControl();

// SideKey Setting
RegistryKey rk =
Registry.LocalMachine.OpenSubKey("ControlPanel\\KeyPad\\SideKey", true);

if (rk == null)
{
    rk = Registry.LocalMachine.CreateSubKey("ControlPanel\\KeyPad\\SideKey");
}

else
{
    rk.SetValue("RightDownKey", 3);
}

// Scanner Open
ScanCtrl.ScanOpen();
```

4.5.2.2 ScannerControl.ScanClose

The **ScanClose** function closes the COM port to a Scan Module.

```
int ScanClose ();
```

Parameters

None

Return Values

If the function fails, the return value is POWER_OFF_ERROR = -2

If the function fails, the return value is PORT_CLOSE_ERROR = -4

If the function succeeds, the return value is ERROR_NONE = 0

Remarks

It is necessary to pair Close() with ScanInit() for proper operation and resource management.

If not call this function after a successful call to ScanInit(), next call to ScanInit() will fail.

```
C#
for (int i = 0; i < 3; i++)
{
```

```

    m_bResult = ScanCtrl.ScanClose();
    Thread.Sleep(300);
    if (m_bResult == true)
        break;
}

```

4.5.2.3 ScannerControl.ScanRead

The **ScanRead** function reads a barcode from the Scan Module.

```
int ScanRead ();
```

Parameters

None

Return Values

If the function succeeds, the return value is nonzero.

If the function fails or times out, the return value is zero.

Remarks

This function is valid only after a successful call to ScanInit().

```

C#
public void ScanRead()
{
    if (m_bReading == true)
    {
        ScanCtrl.ScanReadCancel();
        m_bReading = false;
        return;
    }

    if (m_bContinue == false)
        m_bReading = false;
    else
        m_bReading = true;

    ScanCtrl.ScanRead();
}

```

4.5.2.4 ScannerControl.ScanReadCancel

The **ReadCancel** function cancels ScanRead that was initiated as non-blocking operation. Its use is strongly discouraged

```
int ScanReadCancel ();
```

Parameters

None

Return Values

If the function succeeds, the return value is nonzero.

If the function fails or times out, the return value is zero.

Remarks

This function is valid only after a successful non-blocking call to ScanRead(). Please note that this function is not applicable when read operation was initiated as blocking read.

If not call to Readcancel(), Barcode reading is cancelled after timeout.

ReadCancel tries to maintain the system integrity while terminating the threads and restoring the synchronization primitives. Only when the these primitives do not respond, does it forcibly terminate them. The function thus sometimes takes up to 1 second to complete its tasks.

```
C#
ScanCtrl.ScanReadCancel();
```

4.5.2.5 ScannerControl.SetBarCode_Type

The **SetBarCode_Type** function set Structure of Symbology.

```
void SetBarCode_Type(ref BarCodeType pBCT)
```

Parameters

BarCodeType

Specifies pointer to the KScanBar Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

```
C#
private KScanbarNet.MCBarCodeType M3BarCodeType;

M3BarCodeType = new MCBarCodeType();

M3BarCodeType.bMC_UPCA = TRUE;
M3BarCodeType.bMC_UPCE = TRUE;
M3BarCodeType.bMC_EAN13 = TRUE;
M3BarCodeType.bMC_BOOKLAND = TRUE;
M3BarCodeType.bMC_EAN8 = TRUE;
M3BarCodeType.bMC_CODE39 = TRUE;
M3BarCodeType.bMC_CODE32 = TRUE;
M3BarCodeType.bMC_PZN = TRUE;
M3BarCodeType.bMC_CODE128 = TRUE;
M3BarCodeType.bMC_UCCEAN128 = TRUE;
M3BarCodeType.bMC_CODE93 = TRUE;
M3BarCodeType.bMC_CODE35 = TRUE;
M3BarCodeType.bMC_CODE11 = TRUE;
M3BarCodeType.bMC_I2OF5 = TRUE;
M3BarCodeType.bMC_MSI = TRUE;
M3BarCodeType.bMC_PLESSEY = TRUE;
M3BarCodeType.bMC_CODABAR = TRUE;
M3BarCodeType.bMC_GS1 = TRUE;
M3BarCodeType.bMC_GS1_LIMITED = TRUE;
M3BarCodeType.bMC_GS1_EXPANDED = TRUE;

ScanCtrl.SetBarCodeType(ref M3BarCodeType);
```

4.5.2.6 ScannerControl.GetBarCode_Type

The **GetBarCode_Type** function gets Structure of Symbology.

```
void GetBarCode_Type(out BarCodeType pBCT)
```

Parameters

BarCodeType

Specifies pointer to the KScanBar Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

```
C#
private KScanbarNet.MCBarCodeType M3BarCodeType;

M3BarCodeType = new MCBarCodeType();
// Barcode
public bool m_bUpca;
public bool m_bUpce;
public bool m_bEan13;
public bool m_bBookland;
public bool m_bEan8;
public bool m_bCode39;
public bool m_bCode32;
public bool m_bPzn;
public bool m_bCode128;
public bool m_bUccean128;
public bool m_bCode93;
public bool m_bCode35;
public bool m_bCode11;
public bool m_bI2of5;
public bool m_bMsi;
public bool m_bPlessey;
public bool m_bCodabar;
public bool m_bGsl;
public bool m_bGslLim;
public bool m_bGslExp;

ScanCtrl.GetBarCodeType(out M3BarCodeType);

m_bUpca = M3BarCodeType.bMC_UPCA;
m_bUpce = M3BarCodeType.bMC_UPCE;
m_bEan13 = M3BarCodeType.bMC_EAN13;
m_bBookland = M3BarCodeType.bMC_BOOKLAND;
m_bEan8 = M3BarCodeType.bMC_EAN8;
m_bCode39 = M3BarCodeType.bMC_CODE39;
m_bCode32 = M3BarCodeType.bMC_CODE32;
m_bPzn = M3BarCodeType.bMC_PZN;
m_bCode128 = M3BarCodeType.bMC_CODE128;
m_bUccean128 = M3BarCodeType.bMC_UCCEAN128;
m_bCode93 = M3BarCodeType.bMC_CODE93;
m_bCode35 = M3BarCodeType.bMC_CODE35;
m_bCode11 = M3BarCodeType.bMC_CODE11;
m_bI2of5 = M3BarCodeType.bMC_I2OF5;
m_bMsi = M3BarCodeType.bMC_MSI;
m_bPlessey = M3BarCodeType.bMC_PLESSEY;
m_bCodabar = M3BarCodeType.bMC_CODABAR;
m_bGsl = M3BarCodeType.bMC_GS1;
m_bGslLim = M3BarCodeType.bMC_GS1_LIMITED;
m_bGslExp = M3BarCodeType.bMC_GS1_EXPANDED;
```

4.5.2.7 ScannerControl.SetModule_Option

The **SetModule_Option** function set Structure of ModuleOption.

```
void SetModule_Option(ref ModuleOption pMDO)
```

Parameters

ModuleOption

Specifies pointer to the KScanBar Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
private KScanbarNet.MCModuleOption M3ModuleOption;

M3ModuleOption = new MCModuleOption();
M3ModuleOption.nMC_TimeOutSec = 10;
M3ModuleOption.nMC_SecurityLevel = 1;
M3ModuleOption.nMC_MinLen = 3

ScanCtrl.SetModuleOption(ref M3ModuleOption);
```

4.5.2.8 ScannerControl.GetModule_Option

The **GetModule_Option** function gets Structure of ModuleOption.

```
void GetModule_Option(out ModuleOption pMDO)
```

Parameters

ModuleOption

Specifies pointer to theKScanBar Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
public int m_nSecurityLevel;
public int m_nTimeOut;
public int m_nMinLen;

private KScanbarNet.MCModuleOption M3ModuleOption;

M3ModuleOption = new MCModuleOption();

ScanCtrl.GetModuleOption(out M3ModuleOption);

m_nSecurityLevel = M3ModuleOption.nMC_TimeOutSec;
m_nTimeOut = M3ModuleOption.nMC_SecurityLevel;
m_nMinLen = M3ModuleOption.nMC_MinLen;
```

4.5.2.9 ScannerControl.SetRead_Option

The **SetRead_Option** function set Structure of ReadOption.

```
void SetRead_Option(ref ReadOption pRDO)
```

Parameters

ReadOption

Specifies pointer to the KScanBar Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

```
C#
private KScanbarNet.MCReadOption M3ReadOption;

M3ReadOption = new MCReadOption();

M3ReadOption.bMC_WIDESCANANGLE = true;
M3ReadOption.bMC_HIGHFILTERMODE = false;
M3ReadOption.bMC_RETURNCHECK = true;
M3ReadOption.bMC_ERRORCHECK = false;

ScanCtrl.SetReadOption(ref M3ReadOption);
```

4.5.2.10 ScannerControl.GetRead_Option

The **GetRead_Option** function gets Structure of ReadOption.

```
void GetRead_Option(out ReadOption pRDO)
```

Parameters

ReadOption

Specifies pointer to the KScanBar Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

```
C#
public bool m_bERRORCHECK;
public bool m_bHIGHFILTERMODE;
public bool m_bRETURNCHECK;
public bool m_bWIDESCANANGLE;

private KScanbarNet.MCReadOption M3ReadOption;

M3ReadOption = new MCReadOption();

ScanCtrl.GetReadOption(out M3ReadOption);

m_bWIDESCANANGLE = M3ReadOption.bMC_WIDESCANANGLE;
m_bHIGHFILTERMODE = M3ReadOption.bMC_HIGHFILTERMODE;
m_bRETURNCHECK = M3ReadOption.bMC_RETURNCHECK;
m_bERRORCHECK = M3ReadOption.bMC_ERRORCHECK;
```

4.5.2.11 ScannerControl.GetVersionInfo

The **GetVersionInfo** function returns a string with version information.

```
string GetVersionInfo()
```

Parameters

None

Return Values

The function returns a string with version information associated with the current port.

Remarks

If no Scan Module is open, the return information excludes hardware version information.

4.5.2.12 ScannerControl.SetReturnMode

The **SetReturn** functions to set read method.

```
void SetReturnMode(int type)
```

Parameters

type

Set to return Method.

Return Values

None

Remarks

type value

0 - Return by Event

2 - Return by Key Message

4.5.2.13 ScannerControl.SetScan_Continue

The **SetContinue** functions to set continuous scan mode.

```
void SetScanContinue(  
    int      nContinue  
    BOOL     bForever  
) ;
```

Parameters

nContinue

Set to continuous mode.

bForever

Scan read continue or not when the scan time is out.

Return Values

None

Remarks

None

4.5.2.14 ScannerControl. SetBarOptionUPCA

The **SetBarOptionUPCA** function set the option of UPC-A Barcode.

```
void SetBarOptionUPCA(ref MCBAROption UPCA pUpca);
```

Parameters

MCBarOption_UPCA

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_UPCA pUpca = new MCBAROption_UPCA();
```

```
pUpca.bMC_UPCA_Enable = m_bUpca = UpcaDlg.m_bEnable;
pUpca.bMC_UPCA_XNum = UpcaDlg.m_bXNum;
pUpca.bMC_UPCA_XCD = UpcaDlg.m_bXCD;
pUpca.bMC_UPCA_AS_EAN13 = UpcaDlg.m_bUPCAasEAN13;
pUpca.bMC_UPCA_AddOn = UpcaDlg.m_bAddOn;

ScanCtrl.SetBarOptionUPCA(ref pUpca);
```

4.5.2.15 ScannerControl. GetBarOptionUPCA

The **GetBarOptionUPCA** function gets the option of UPC-A Barcode.

```
void GetBarOptionUPCA(out MCBAROption_UPCA pUpca);
```

Parameters

MCBarOption_UPCA

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_UPCA pUpca = new MCBAROption_UPCA();

ScanCtrl.GetBarOptionUPCA(out pUpca);

UpcaDlg.m_bEnable = pUpca.bMC_UPCA_Enable;
UpcaDlg.m_bXNum = pUpca.bMC_UPCA_XNum;
UpcaDlg.m_bXCD = pUpca.bMC_UPCA_XCD;
UpcaDlg.m_bUPCAasEAN13 = pUpca.bMC_UPCA_AS_EAN13;
UpcaDlg.m_bAddOn = pUpca.bMC_UPCA_AddOn;
```

4.5.2.16 ScannerControl. SetBarOptionUPCE

The **SetBarOptionUPCE** function sets the option of UPC-E Barcode.

```
void SetBarOptionUPCE(ref MCBAROption_UPCE pUpce);
```

Parameters

MCBarOption_UPCE

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_UPCE pUpce = new MCBAROption_UPCE();

pUpce.bMC_UPCE_Enable = m_bUpce = UpceDlg.m_bEnable;
pUpce.bMC_UPCE_XNum = UpceDlg.m_bXNum;
pUpce.bMC_UPCE_XCD = UpceDlg.m_bXCD;
pUpce.nMC_UPCE_Convert = UpceDlg.m_nConvert;

ScanCtrl.SetBarOptionUPCE(ref pUpce);
```

4.5.2.17 ScannerControl. GetBarOptionUPCE

The **GetBarOptionUPCE** function gets the option of UPC-E Barcode.

```
void GetBarOptionUPCE(out MCBAROption_UPCE pUpce);
```

Parameters

MCBarOption_UPCE

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_UPCE pUpce = new MCBAROption_UPCE();  
  
ScanCtrl.GetBarOptionUPCE(out pUpce);  
  
UpceDlg.m_bEnable = pUpce.bMC_UPCE_Enable;  
UpceDlg.m_bXNum = pUpce.bMC_UPCE_XNum;  
UpceDlg.m_bXCD = pUpce.bMC_UPCE_XCD;  
UpceDlg.m_nConvert = pUpce.nMC_UPCE_Convert;
```

4.5.2.18 ScannerControl. SetBarOptionEAN13

The **SetBarOptionEAN13** function sets the option of EAN-13 Barcode.

```
void SetBarOptionEAN13(ref MCBAROption_EAN13 pEan13);
```

Parameters

MCBarOption_EAN13

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_EAN13 pEan13 = new MCBAROption_EAN13();  
  
pEan13.bMC_EAN13_Enable = m_bEan13 = Ean13Dlg.m_bEnable;  
pEan13.bMC_BOOKLAND_Enable = m_bBookland = Ean13Dlg.m_bBookland;  
pEan13.bMC_EAN13_XCD = Ean13Dlg.m_bXCD;  
pEan13.bMC_EAN13_AddOn = Ean13Dlg.m_bAddOn;  
  
ScanCtrl.SetBarOptionEAN13(ref pEan13);
```

4.5.2.19 ScannerControl. GetBarOptionEAN13

The **GetBarOptionEAN13** function gets the option of EAN-13 Barcode.

```
void GetBarOptionEAN13(out MCBAROption_EAN13 pEan13);
```

Parameters

MCBarOption_EAN13

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_EAN13 pEan13 = new MCBarOption_EAN13();  
  
ScanCtrl.GetBarOptionEAN13(out pEan13);  
  
Ean13Dlg.m_bEnable = pEan13.bMC_EAN13_Enable;  
Ean13Dlg.m_bBookland = pEan13.bMC_BOOKLAND_Enable;  
Ean13Dlg.m_bXCD = pEan13.bMC_EAN13_XCD;  
Ean13Dlg.m_bAddOn = pEan13.bMC_EAN13_AddOn;
```

4.5.2.20 ScannerControl. SetBarOptionEAN8

The **SetBarOptionEAN8** function set the option of EAN-8 Barcode.

```
void SetBarOptionEAN8(ref MCBarOption_EAN8 pEan8);
```

Parameters

MCBarOption_EAN8

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_EAN8 pEan8 = new MCBarOption_EAN8();  
  
pEan8.bMC_EAN8_Enable = m_bEan8 = Ean8Dlg.m_bEnable;  
pEan8.bMC_EAN8_XCD = Ean8Dlg.m_bXCD;  
pEan8.nMC_EAN8_AS_EAN13 = Ean8Dlg.m_bEAN8asEAN13;  
  
ScanCtrl.SetBarOptionEAN8(ref pEan8);
```

4.5.2.21 ScannerControl. GetBarOptionEAN8

The **GetBarOptionEAN8** function get the option of EAN-8 Barcode.

```
void GetBarOptionEAN8(out MCBarOption_EAN8 pEan8);
```

Parameters

MCBarOption_EAN8

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```

MCBarOption_EAN8 pEan8 = new MCBarOption_EAN8();

ScanCtrl.GetBarOptionEAN8(out pEan8);

Ean8Dlg.m_bEnable = pEan8.bMC_EAN8_Enable;
Ean8Dlg.m_bXCD = pEan8.bMC_EAN8_XCD;
Ean8Dlg.m_bEAN8asEAN13 = pEan8.nMC_EAN8_AS_EAN13;

```

4.5.2.22 ScannerControl. SetBarOptionCODE39

The **SetBarOptionCODE39** function set the option of CODE39 Barcode.

```
void SetBarOptionCODE39(ref MCBarOption_CODE39 pCode39);
```

Parameters

MCBarOption_CODE39

Specifies pointer to the MCSSLibNet Structure.

Return Values

None.

Remarks

Use after successful initialization of the scanner.

C#

```

MCBarOption_CODE39 pCode39 = new MCBarOption_CODE39();

pCode39.bMC_CODE39_Enable = m_bCode39 = Code39Dlg.m_bEnable;
pCode39.bMC_CODE32_Enable = m_bCode32 = Code39Dlg.m_bCode32;
pCode39.bMC_PZN_Enable = m_bPzn = Code39Dlg.m_bPzn;
pCode39.bMC_CODE39_CDV = Code39Dlg.m_bCDV;
pCode39.bMC_CODE39_XCD = Code39Dlg.m_bXCD;
pCode39.bMC_CODE39_FullASCII = Code39Dlg.m_bFullASCII;
pCode39.nMC_CODE39_MinLen = Code39Dlg.m_nMinLen;
pCode39.nMC_CODE39_MaxLen = Code39Dlg.m_nMaxLen;

ScanCtrl.SetBarOptionCODE39(ref pCode39);

```

4.5.2.23 ScannerControl. GetBarOptionCODE39

The **GetBarOptionCODE39** function gets the option of CODE39 Barcode.

```
void GetBarOptionCODE39(out MCBarOption_CODE39 pCode39);
```

Parameters

MCBarOption_CODE39

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```

MCBarOption_CODE39 pCode39 = new MCBarOption_CODE39();

ScanCtrl.GetBarOptionCODE39(out pCode39);

Code39Dlg.m_bEnable = pCode39.bMC_CODE39_Enable;

```

```

Code39Dlg.m_bCode32 = pCode39.bMC_CODE32_Enable;
Code39Dlg.m_bPzn = pCode39.bMC_PZN_Enable;
Code39Dlg.m_bCDV = pCode39.bMC_CODE39_CDV;
Code39Dlg.m_bXCD = pCode39.bMC_CODE39_XCD;
Code39Dlg.m_bFullASCII = pCode39.bMC_CODE39_FullASCII;
Code39Dlg.m_nMinLen = pCode39.nMC_CODE39_MinLen;
Code39Dlg.m_nMaxLen = pCode39.nMC_CODE39_MaxLen;

```

4.5.2.24 ScannerControl. SetBarOptionCODE128

The **SetBarOptionCODE128** function sets the option of CODE128 Barcode.

```
void SetBarOptionCODE128(ref MCBAROption CODE128 pCode128);
```

Parameters

MCBarOption_CODE128

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```

MCBarOption_CODE128 pCode128 = new MCBAROption_CODE128();

pCode128.bMC_CODE128_Enable = m_bCode128 = Code128Dlg.m_bEnable;
pCode128.bMC_UCCEAN128_Enable = m_bUccean128 = Code128Dlg.m_bUccean128;
pCode128.nMC_CODE128_MinLen = Code128Dlg.m_nMinLen;
pCode128.nMC_CODE128_MaxLen = Code128Dlg.m_nMaxLen;

ScanCtrl.SetBarOptionCODE128(ref pCode128);

```

4.5.2.25 ScannerControl. GetBarOptionCODE128

The **GetBarOptionCODE128** function gets the option of CODE128 Barcode.

```
void GetBarOptionCODE128(out MCBAROption CODE128 pCode128);
```

Parameters

MCBarOption_CODE128

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```

MCBarOption_CODE128 pCode128 = new MCBAROption_CODE128();

ScanCtrl.GetBarOptionCODE128(out pCode128);

Code128Dlg.m_bEnable = pCode128.bMC_CODE128_Enable;
Code128Dlg.m_bUccean128 = pCode128.bMC_UCCEAN128_Enable;
Code128Dlg.m_nMinLen = pCode128.nMC_CODE128_MinLen;
Code128Dlg.m_nMaxLen = pCode128.nMC_CODE128_MaxLen;

```

4.5.2.26 ScannerControl. SetBarOptionCODE93

The **SetBarOptionCODE93** function sets the option of CODE93 Barcode.

```
void SetBarOptionCODE93(ref MCBAROption CODE93 pCode93);
```

Parameters

MCBarOption_CODE93

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_CODE93 pCode93 = new MCBAROption_CODE93();  
  
pCode93.bMC_CODE93_Enable = m_bCode93 = Code93Dlg.m_bEnable;  
pCode93.nMC_CODE93_MinLen = Code93Dlg.m_nMinLen;  
pCode93.nMC_CODE93_MaxLen = Code93Dlg.m_nMaxLen;  
  
ScanCtrl.SetBarOptionCODE93(ref pCode93);
```

4.5.2.27 ScannerControl. GetBarOptionCODE93

The **GetBarOptionCODE93** function gets the option of CODE93 Barcode.

```
void GetBarOptionCODE93(out MCBAROption CODE93 pCode93);
```

Parameters

MCBarOption_CODE93

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_CODE93 pCode93 = new MCBAROption_CODE93();  
  
ScanCtrl.GetBarOptionCODE93(out pCode93);  
  
Code93Dlg.m_bEnable = pCode93.bMC_CODE93_Enable;  
Code93Dlg.m_nMinLen = pCode93.nMC_CODE93_MinLen;  
Code93Dlg.m_nMaxLen = pCode93.nMC_CODE93_MaxLen;
```

4.5.2.28 ScannerControl. SetBarOptionCODE35

The **SetBarOptionCODE35** function sets the option of CODE35 Barcode.

```
void SetBarOptionCODE35(ref MCBAROption CODE35 pCode35);
```

Parameters

MCBarOption_CODE35

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_CODE35 pCode35 = new MCBarOption_CODE35();  
  
pCode35.bMC_CODE35_Enable = m_bCode35 = Code35Dlg.m_bEnable;  
  
ScanCtrl.SetBarOptionCODE35(ref pCode35);
```

4.5.2.29 ScannerControl. GetBarOptionCODE35

The **GetBarOptionCODE35** function gets the option of CODE35 Barcode.

```
void GetBarOptionCODE35(out MCBarOption_CODE35 pCode35);
```

Parameters

MCBarOption_CODE35

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_CODE35 pCode35 = new MCBarOption_CODE35();  
  
ScanCtrl.GetBarOptionCODE35(out pCode35);  
  
Code35Dlg.m_bEnable = pCode35.bMC_CODE35_Enable;
```

4.5.2.30 ScannerControl. SetBarOptionCODE11

The **SetBarOptionCODE11** function sets the option of CODE11 Barcode.

```
void SetBarOptionCODE11(ref MCBarOption_CODE11 pCode11);
```

Parameters

MCBarOption_CODE11

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_CODE11 pCode11 = new MCBarOption_CODE11();  
  
pCode11.bMC_CODE11_Enable = m_bCode11 = Code11Dlg.m_bEnable;  
pCode11.bMC_CODE11_XCD = Code11Dlg.m_bXCD;  
pCode11.nMC_CODE11_CDV = Code11Dlg.m_nCDV;  
pCode11.nMC_CODE11_MinLen = Code11Dlg.m_nMinLen;  
pCode11.nMC_CODE11_MaxLen = Code11Dlg.m_nMaxLen;
```

```
ScanCtrl.SetBarOptionCODE11(ref pCode11);
```

4.5.2.31 ScannerControl. GetBarOptionCODE11

The **GetBarOptionCODE11** function gets the option of CODE11 Barcode.

```
void GetBarOptionCODE11(out MCBarOption CODE11 pCode11);
```

Parameters

MCBarOption_CODE11

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_CODE11 pCode11 = new MCBarOption_CODE11();

ScanCtrl.GetBarOptionCODE11(out pCode11);

Code11Dlg.m_bEnable = pCode11.bMC_CODE11_Enable;
Code11Dlg.m_bXCD = pCode11.bMC_CODE11_XCD;
Code11Dlg.m_nCDV = pCode11.nMC_CODE11_CDV;
Code11Dlg.m_nMinLen = pCode11.nMC_CODE11_MinLen;
Code11Dlg.m_nMaxLen = pCode11.nMC_CODE11_MaxLen;
```

4.5.2.32 ScannerControl. SetBarOptionI2OF5

The **SetBarOptionCODEI2OF5** function sets the option of Interleaved 2of5 Barcode.

```
void SetBarOptionCODEI2OF5(ref MCBarOption I2OF5 pI2of5);
```

Parameters

MCBarOption_I2OF5

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_I2OF5 pI2of5 = new MCBarOption_I2OF5();

pI2of5.bMC_I2OF5_Enable = m_bI2of5 = I2of5Dlg.m_bEnable;
pI2of5.bMC_ITF14_Enable = I2of5Dlg.m_bItf14;
pI2of5.bMC_MATRIX2OF5_Enable = I2of5Dlg.m_bMatrix2of5;
pI2of5.bMC_DLOGIG_Enable = I2of5Dlg.m_bDlogic;
pI2of5.bMC_INDUSTRIY_Enable = I2of5Dlg.m_bIndustry;
pI2of5.bMC_IATA_Enable = I2of5Dlg.m_bIata;
pI2of5.bMC_I2OF5_CDV = I2of5Dlg.m_bCDV;
pI2of5.bMC_I2OF5_XCD = I2of5Dlg.m_bXCD;
pI2of5.nMC_I2OF5_MinLen = I2of5Dlg.m_nMinLen;
pI2of5.nMC_I2OF5_MaxLen = I2of5Dlg.m_nMaxLen;

ScanCtrl.SetBarOptionI2OF5(ref pI2of5);
```


4.5.2.33 ScannerControl. GetBarOptionI2OF5

The **GetBarOptionCODEI2OF5** function gets the option of Interleaved 2of5 Barcode.

```
void GetBarOptionI2OF5(out MCBAROption_I2OF5 pI2of5);
```

Parameters

MCBarOption_I2OF5

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_I2OF5 pI2of5 = new MCBAROption_I2OF5();

ScanCtrl.GetBarOptionI2OF5(out pI2of5);

I2of5Dlg.m_bEnable = pI2of5.bMC_I2OF5_Enable;
I2of5Dlg.m_bItf14 = pI2of5.bMC_ITF14_Enable;
I2of5Dlg.m_bMatrix2of5 = pI2of5.bMC_MATRIX2OF5_Enable;
I2of5Dlg.m_bDlogic = pI2of5.bMC_DLOGIG_Enable;
I2of5Dlg.m_bIndustry = pI2of5.bMC_INDUSTRY_Enable;
I2of5Dlg.m_bIata = pI2of5.bMC_IATA_Enable;
I2of5Dlg.m_bCDV = pI2of5.bMC_I2OF5_CDV;
I2of5Dlg.m_bXCD = pI2of5.bMC_I2OF5_XCD;
I2of5Dlg.m_nMinLen = pI2of5.nMC_I2OF5_MinLen;
I2of5Dlg.m_nMaxLen = pI2of5.nMC_I2OF5_MaxLen;
```

4.5.2.34 ScannerControl. SetBarOptionCODABAR

The **SetBarOptionCODABAR** function sets the option of CODABAR Barcode.

```
void SetBarOptionCODABAR(ref MCBAROption_CODABAR pCodabar);
```

Parameters

MCBarOption_CODABAR

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_CODABAR pCodabar = new MCBAROption_CODABAR();

pCodabar.bMC_CODABAR_Enable = m_bCodabar = CodabarDlg.m_bEnable;
pCodabar.bMC_CODABAR_XSS = CodabarDlg.m_bXSS;
pCodabar.nMC_CODABAR_MinLen = CodabarDlg.m_nMinLen;
pCodabar.nMC_CODABAR_MaxLen = CodabarDlg.m_nMaxLen;

ScanCtrl.SetBarOptionCODABAR(ref pCodabar);
```

4.5.2.35 ScannerControl. GetBarOptionCODABA

The **GetBarOptionCODABAR** function gets the option of CODABAR Barcode.

```
void GetBarOptionCODABAR(out MCBAROption_CODABAR pCodabar);
```

Parameters

MCBarOption_CODABAR

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_CODABAR pCodabar = new MCBAROption_CODABAR();

ScanCtrl.GetBarOptionCODABAR(out pCodabar);

CodabarDlg.m_bEnable = pCodabar.bMC_CODABAR_Enable;
CodabarDlg.m_bXSS = pCodabar.bMC_CODABAR_XSS;
CodabarDlg.m_nMinLen = pCodabar.nMC_CODABAR_MinLen;
CodabarDlg.m_nMaxLen = pCodabar.nMC_CODABAR_MaxLen;
```

4.5.2.36 ScannerControl. SetBarOptionMSI

The **SetBarOptionMSI** function sets the option of MSI Barcode.

```
void SetBarOptionMSI(ref MCBAROption_MSI pMsi);
```

Parameters

MCBarOption_MSI

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_MSI pMsi = new MCBAROption_MSI();

pMsi.bMC_MSI_Enable = m_bMsi = MsiDlg.m_bEnable;
pMsi.bMC_MSI_CDV = MsiDlg.m_bCDV;
pMsi.bMC_MSI_XCD = MsiDlg.m_bXCD;
pMsi.nMC_MSI_MinLen = MsiDlg.m_nMinLen;
pMsi.nMC_MSI_MaxLen = MsiDlg.m_nMaxLen;

ScanCtrl.SetBarOptionMSI(ref pMsi);
```

4.5.2.37 ScannerControl. GetBarOptionMSI

The **GetBarOptionMSI** function gets the option of MSI Barcode.

```
void GetBarOptionMSI(out MCBAROption_MSI pMsi);
```

Parameters

MCBarOption_MSI

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_MSI pMsi = new MCBarOption_MSI();

ScanCtrl.GetBarOptionMSI(out pMsi);

MsiDlg.m_bEnable = pMsi.bMC_MSI_Enable;
MsiDlg.m_bCDV = pMsi.bMC_MSI_CDV;
MsiDlg.m_bXCD = pMsi.bMC_MSI_XCD;
MsiDlg.m_nMinLen = pMsi.nMC_MSI_MinLen;
MsiDlg.m_nMaxLen = pMsi.nMC_MSI_MaxLen;
```

4.5.2.38 ScannerControl. SetBarOptionPLESSEY

The **SetBarOptionPLESSEY** function sets the option of PLESSEY Barcode.

```
void SetBarOptionPLESSEY(ref MCBarOption_PLESSEY pPlessey);
```

Parameters

MCBarOption_PLESSEY

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_PLESSEY pPlessey = new MCBarOption_PLESSEY();

pPlessey.bMC_PLESSEY_Enable = m_bPlessey = PlesseyDlg.m_bEnable;
pPlessey.bMC_PLESSEY_CDV = PlesseyDlg.m_bCDV;
pPlessey.bMC_PLESSEY_XCD = PlesseyDlg.m_bXCD;
pPlessey.nMC_PLESSEY_MinLen = PlesseyDlg.m_nMinLen;
pPlessey.nMC_PLESSEY_MaxLen = PlesseyDlg.m_nMaxLen;

ScanCtrl.SetBarOptionPLESSEY(ref pPlessey);
```

4.5.2.39 ScannerControl. GetBarOptionPLESSEY

The **GetBarOptionPLESSEY** function gets the option of PLESSEY Barcode.

```
void GetBarOptionPLESSEY(out MCBarOption_PLESSEY pPlessey);
```

Parameters

MCBarOption_PLESSEY

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

```
C#
MCBarOption_PLESSEY pPlessey = new MCBarOption_PLESSEY();

ScanCtrl.GetBarOptionPLESSEY(out pPlessey);

PlesseyDlg.m_bEnable = pPlessey.bMC_PLESSEY_Enable;
PlesseyDlg.m_bCDV = pPlessey.bMC_PLESSEY_CDV;
PlesseyDlg.m_bXCD = pPlessey.bMC_PLESSEY_XCD;
PlesseyDlg.m_nMinLen = pPlessey.nMC_PLESSEY_MinLen;
PlesseyDlg.m_nMaxLen = pPlessey.nMC_PLESSEY_MaxLen;
```

4.5.2.40 ScannerControl. SetBarOptionGS1

The **SetBarOptionGS1** function sets the option of GS1 Barcode.

```
void SetBarOptionGS1(ref MCBarOption_GS1 pGs1);
```

Parameters

MCBarOption_GS1

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

```
C#
MCBarOption_GS1 pGs1 = new MCBarOption_GS1();

pGs1.bMC_GS1_Enable = m_bGs1 = Gs1Dlg.m_bEnable;
pGs1.bMC_GS1LIM_Enable = m_bGs1Lim = Gs1Dlg.m_bGs1Lim;
pGs1.bMC_GS1EXP_Enable = m_bGs1Exp = Gs1Dlg.m_bGs1Exp;

ScanCtrl.SetBarOptionGS1(ref pGs1);
```

4.5.2.41 ScannerControl. GetBarOptionGS1

The **GetBarOptionGS1** function gets the option of GS1 Barcode.

```
void GetBarOptionGS1(out MCBarOption_GS1 pGs1);
```

Parameters

MCBarOption_GS1

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

```
C#
MCBarOption_GS1 pGs1 = new MCBarOption_GS1();

ScanCtrl.GetBarOptionGS1(out pGs1);

Gs1Dlg.m_bEnable = pGs1.bMC_GS1_Enable;
```

```
GslDlg.m_bGslLim = pGsl.bMC_GSLIM_Enable;  
GslDlg.m_bGslExp = pGsl.bMC_GS1EXP_Enable;
```

4.5.2.42 ScannerControl. SetBarOptionTELEPEN

The **SetBarOptionTELEPEN** function sets the option of TELEPEN Barcode.

```
void SetBarOptionTELEPEN(ref MCBAROption TELEPEN pTelepen);
```

Parameters

MCBarOption_TELEPEN

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_TELEPEN pTelepen = new MCBAROption_TELEPEN();  
  
pTelepen.bMC_TELEPEN_Enable = TelepenDlg.m_bEnable;  
pTelepen.bMC_TELEPEN_OldStyle = TelepenDlg.m_bOldStyle;  
  
ScanCtrl.SetBarOptionTELEPEN(ref pTelepen);
```

4.5.2.43 ScannerControl. GetBarOptionTELEPEN

The **GetBarOptionTELEPEN** function gets the option of TELEPEN Barcode.

```
void GetBarOptionTELEPEN(out MCBAROption TELEPEN pTelepen);
```

Parameters

MCBarOption_TELEPEN

Specifies pointer to the MCSSLibNet Structure.

Return Values

None

Remarks

Use after successful initialization of the scanner.

C#

```
MCBarOption_TELEPEN pTelepen = new MCBAROption_TELEPEN();  
  
ScanCtrl.GetBarOptionTELEPEN(out pTelepen);  
  
TelepenDlg.m_bEnable = pTelepen.bMC_TELEPEN_Enable;  
TelepenDlg.m_bOldStyle = pTelepen.bMC_TELEPEN_OldStyle;
```

4.6 Scanner 2D (Imager)

This section provides description of the functions and DLLs which are used to manage 2D scanner module.

Required Files

For C++

Required header:

M3MobileImager.h

Required lib:

M3MobileImager.lib

Required DLL:

M3MobileImager.dll

For C#

Required DLL:

M3MobileImager.dll
M3MobileImagerNet.dll

Supported Product

M3 ORANGE with 2D scanner that uses software decoder.

4.6.1 Reference and Function List for C++

Definitions

#define macro

#define SYM_ENABLE	0x00000001	Enable Symbology bit
#define SYM_CHECK_ENABLE	0x00000002	Enable usage of check character.
#define SYM_CHECK_TRANSMIT	0x00000004	Send check character.
#define SYM_START_STOP_XMIT	0x00000008	Include the start and stop characters in the decoded result string.
#define SYM_ENABLE_APPEND_MODE	0x00000010	Code39 append mode.
#define SYM_ENABLE_FULLASCII	0x00000020	Enable Code39 Full ASCII.
#define SYM_NUM_SYS_TRANSMIT	0x00000040	UPC-A/UPC-E Send Num Sys
#define SYM_2_DIGIT_ADDENDA	0x00000080	Enable 2 digit Addenda (UPC and EAN).
#define SYM_5_DIGIT_ADDENDA	0x00000100	Enable 5 digit Addenda (UPC and EAN).
#define SYM_ADDENDA_REQUIRED	0x00000200	Only allow codes with addenda (UPC and EAN).
#define SYM_ADDENDA_SEPARATOR	0x00000400	Include Addenda separator space in returned string.
#define SYM_EXPANDED_UPCE	0x00000800	Extended UPC-E.
#define SYM_UPCE1_ENABLE	0x00001000	UPC-E1 enable (use SYMBOLOGY_ENABLE for UPC-E0).
#define SYM_ENABLE_MESA_IMS	0x00020000	Mesa IMS enable.
#define SYM_ENABLE_MESA_1MS	0x00040000	Mesa 1MS enable.
#define SYM_ENABLE_MESA_3MS	0x00080000	Mesa 3MS enable.
#define SYM_ENABLE_MESA_9MS	0x00100000	Mesa 9MS enable.
#define SYM_ENABLE_MESA_UMS	0x00200000	Mesa UMS enable.
#define SYM_ENABLE_MESA_EMS	0x00400000	Mesa EMS enable.
#define SYM_TELEPEN_OLD_STYLE	0x04000000	Telepen Old Style mode.:
#define SYM_COMPOSITE_UPC	0x00002000	Enable UPC Composite codes.
#define SYM_POSICODE_LIMITED_1	0x08000000	PosiCode Limited of 1
#define SYM_POSICODE_LIMITED_2	0x10000000	PosiCode Limited of 2
#define SYM_MAXICODE_CARRIERMSGONLY	0x40000000	maxicode option
#define SYM_EAN13_ISBN	0x80000000	ean13 isbn
#define MAX_TEMPLATE_LEN	256	OCR TEMPLATE length

#define MAX_GROUP_H_LEN	256	OCR Group H length
#define MAX_GROUP_G_LEN	256	OCR Group G length
#define MAX_CHECK_CHAR_LEN	64	OCR check length

Status Return value

Please refer to the below table for the status value definition.

Status Value Definition	Code	Meaning
TRUE	1	Success
FALSE	0	General Error

Structure

ID_SYM – This enumeration is Symbology ID

```
typedef enum {
    ID_AZTEC = 0,           Aztec Code           SymFlagsRange
    ID_MESA,                Aztec Mesas       SymFlagsOnly
    ID_CODABAR,             Codabar           SymFlagsRange
    ID_CODE11,              Code 11           SymFlagsRange
    ID_CODE128,             Code 128          SymFlagsRange
    ID_CODE39,              Code 39           SymFlagsRange
    ID_CODE49,              Code 49           SymFlagsRange
    ID_CODE93,              Code 93           SymFlagsRange
    ID_COMPOSITE,           Composite Code     SymFlagsRange
    ID_DATAMATRIX,          Data Matrix       SymFlagsRange
    ID_EAN8,                EAN-8             SymFlagsOnly
    ID_EAN13,               ENA-13            SymFlagsOnly
    ID_INT25,               Interleaved 2 of 5 SymFlagsRange
    ID_MAXICODE,            MaxiCode          SymFlagsRange
    ID_MICROPDF,            Micro PDF417      SymFlagsRange
    ID_OCR,                 Ocr               SymCodeOCR
    ID_PDF417,              PDF417            SymFlagsRange
    ID_POSTNET,             Postnet           SymFlagsOnly
    ID_QR,                  QR Code           SymFlagsRange
    ID_RSS,                 Reduced Space Symbology (RSS) SymFlagsRange
    ID_UPCA,                UPC-A             SymFlagsOnly
    ID_UPCE0,               UPC-E             SymFlagsOnly
    ID_UPCE1,               UPC-E1            SymFlagsOnly
    ID_ISBT,                ISBT              SymFlagsOnly
    ID_BPO,                 British Post       SymFlagsOnly
    ID_CANPOST,             Canadian Post      SymFlagsOnly
    ID_AUSPOST,             Australian Post    SymFlagsOnly
    ID_IATA25,              Straight 2 of 5 IATA SymFlagsRange
    ID_CODABLOCK,           Codablock         SymFlagsRange
    ID_JAPOST,              Japanese Post      SymFlagsOnly
    ID_PLANET,              Planet Code        SymFlagsOnly
    ID_DUTCHPOST,           KIX (Netherlands) Post SymFlagsOnly
    ID_MSI,                 MSI Code          SymFlagsRange
    ID_TLCODE39,            TCIF Linked Code 39 (TLC39) SymFlagsOnly
    ID_TRIOPTIC,            Trioptic Code      SymFlagsOnly
    ID_CODE32,              Code 32           SymFlagsOnly
    ID_STRT25,              Straight 2 of 5 Industrial SymFlagsRange
    ID_MATRIX25,            Matrix 2 of 5      SymFlagsRange
    ID_PLESSEY,             Plessey Code      SymFlagsRange
    ID_CHINAPOST,           China Post         SymFlagsRange
    ID_KOREAPOST,           Korean Post        SymFlagsRange
    ID_TELEPEN,             Telepen           SymFlagsRange
    ID_CODE16K,             Code 16k          SymFlagsRange
    ID_POSICODE,            PosiCode          SymFlagsRange
    ID_COUPONCODE,          UPC-A with Extended Coupon Code SymFlagsOnly
}
```


ID_USPS4CB,	USPS 4-State Customer	SymFlagsOnly
ID_IDTAG,	UPU 4 State ID Tag	SymFlagsOnly
ID_GS1_128,	GS1 128	SymFlagsRange
ID_GEN_CODE128,	general code 128	SymFlagsRange
ID_ALL = 100		

```

} ID_SYM ;;

```

SetupType - This structure is enumerated type for specifying whether a read configuration item call should return the current settings or the imager default setting.

```

typedef enum {
    SETUP_DEFAULT = 0,
    SETUP_CURRENT
} SetupType;

```

IMAGER_VERSION_INFO - This structure is for version information.

```

typedef struct _tagVERSION_INFO{
    TCHAR tcAPIRev[ MAX_VERSION_STRING_LEN ];
    TCHAR tcDecoderRev[ MAX_VERSION_STRING_LEN ];
    TCHAR tcScanDriverRev[ MAX_VERSION_STRING_LEN ];
    TCHAR tcEtcInfo[1000];
    DWORD dwFirmwareVersion;
    DWORD dwFirmwareCksum
    DWORD dwEngineId
} IMAGER_VERSION_INFO, *PIMAGER_VERSION_INFO ;

```

EVENT_TYPE - Type of data causing the event notification.

```

typedef enum {
    BARCODE_EVENT = 0,
    IMAGE_EVENT,
    TEXT_MSG_EVENT,
    INTELIMG_BARCODE_EVENT,
    INTELIMG_IMAGE_EVENT,
    TRIGGER_EVENT
} EventType t, EVENT_TYPE, *PEVENT_TYPE;

```

DECODE_MSG - Data structure that holds the decoded bar code message.

```

typedef struct _tagDECODE_MSG{
    DWORD dwStructSize;
    TCHAR pchMessage[4096];
    TCHAR chCodeID;
    TCHAR chSymLetter;
    TCHAR chSymModifier;
    DWORD nLength;
} DECODE_MSG, *PDECODE_MSG;

```

SymFlagsOnly - This structure for symbologies with no specified minimum or maximum length.

```

typedef struct __tagSymFlagsOnly{
    DWORD dwFlags;
} SymFlagsOnly, *PSymFlagsOnly;

```

SymFlagsRange - This structure for symbologies with minimum and maximum length.

```

typedef struct __tagSymFlagsRange{
    DWORD dwFlags;
    DWORD dwMinLen;
    DWORD dwMaxLen;
} SymFlagsRange, *PSymFlagsRange;

```

SymCodeOCR - This structure for unusual OCR.

```

typedef enum {
    OCR_MODE_DISABLED = 0,
    OCR_MODE_A,

```

```

    OCR_MODE_B,
    OCR_MODE_MONEY,
    OCR_MODE_MICR_UNSUPPORTED,
}OCRMode;

typedef enum {
    OCR_DIRECTION_LeftToRight = 0,
    OCR_DIRECTION_TopToBottom,
    OCR_DIRECTION_RightToLeft,
    OCR_DIRECTION_BottomToTop,
}OCRDirection;

typedef struct __tagSymCodeOCR{
    OCRMode ocrMode;
    OCRDirection ocrDirection;
    TCHAR tcTemplate[ MAX_TEMPLATE_LEN ];
    TCHAR tcGroupG[MAX_GROUP_G_LEN ];
    TCHAR tcGroupH[MAX_GROUP_H_LEN ];
    TCHAR tcCheckChar[ MAX_CHECK_CHAR_LEN ];
}SymCodeOCR, *PSymCodeOCR;

```

ScanIlluminat - Enumerated integer type that identifies possible illumination modes used during image acquisition.

```

typedef enum {
    SCAN_ILLUM_AIMER_OFF = 0,
    SCAN_ILLUM_ONLY_ON,
    SCAN_AIMER_ONLY_ON,
    SCAN_ILLUM_AIMER_ON,
}ScanIlluminat

```

4.6.1.1 Connect

This function connects the engine.

```
BOOL Connect(  
    BOOL    On  
)
```

Parameters

On

If this value is true, connects the engine.

If this value is false, disconnects the engine.

Return Value

true indicates success.

false indicates failure.

4.6.1.2 ScanRead

This function causes the engine to start scanning for a decodable symbol.

```
BOOL ScanRead(  
    DWORD      dwTimeout  
    PDECODE_MSG pmsg  
)
```

Parameters

dwTimeout

Maximum time (in seconds) to attempt to decode before declaring a no decode.

A ScannerTimeoutDelegate event specifies that the timeout is whatever is currently set on the imager.

A value of 0 indicates no timeout.

pmsg

Reference to a DECODE_MSG structure to receive the data decoded.

Return Value

true indicates success.

false indicates failure.

4.6.1.3 CancelIO

This function cancels the current bar code capture.

```
BOOL CancelIO( )
```

Return Value

true indicates success.

false indicates failure.

4.6.1.4 AimerOn

This function turns the engine's aiming mechanism on or off.

```
BOOL AimerOn(  
    BOOL    On  
)
```

Parameters

On

If this value is true, turns on the aimer.
If this value is false, turns off the aimer.

Return Value

true indicates success.
false indicates failure.

4.6.1.5 LightsOn

This function turns the engine's illumination LEDs on and off.

```
BOOL LightsOn(  
    BOOL    On  
)
```

Parameters

On

If TRUE, the illumination LEDs are turned on; otherwise they're turned off.

Return Value

true indicates success.
false indicates failure.

4.6.1.6 SetScanningLightsMode

This function gives the user the ability to select what the illumination and aimer do during imaging.

```
BOOL SetScanningLightsMode(  
    ScanIlluminat    nIllumMode  
)
```

Parameters

ScanIlluminate

SCAN_ILLUM_AIMER_OFF=0	Neither aimers nor illumination
SCAN_ILLUM_ONLY_ON	Illumination only
SCAN_AIMER_ONLY_ON	Aimers only
SCAN_ILLUM_AIMER_ON	Both aimers and illumination

Return Value

true indicates success.
false indicates failure.

4.6.1.7 SetScanMethods

This function registers handle or event for scanning.

```
BOOL SetScanMethods(  
    HANDLE          hEventHandle  
    HWND            hWnd  
    EVENTCALLBACK   EventCallback  
)
```

Parameters

hEventHandle

If you want to use event, register event.

hWnd

If you want to use window message, register window handle.

EventCallback

If you want to use callback function, register callback function.
You don't use parameter, parameter is null.

Return Value

true indicates success.
false indicates failure.

4.6.1.8 GetScanResult

This function retrieves the data from the last signal event (bar code capture).

```
BOOL GetScanResult(  
    EventType_t      *pEventType  
    PVOID            pResultStruct  
)
```

Parameters

pEventType

Reference to a EVENT_TYPE structure to receive event type.

pResultStruct

Reference to a DECODE_MSG structure to receive the data decoded.

Return Value

true indicates success.
false indicates failure.

4.6.1.9 DefaultSymbology

This function sets the specified symbologies to their factory default configurations.

```
BOOL DefaultSymbology(  
    int    nSymId  
)
```

Parameters

nSymId

One of the symbology enumerated types, e.g., SYM_CODE39, SYM_OCR, or SYM_ALL to default all symbologies.

Return Value

true indicates success.
false indicates failure.

4.6.1.10 SetEnableDisableSymbology

This function sets state of the specified symbologies.

```
BOOL SetEnableDisableSymbology(  
    int    nSymId  
    BOOL   bEnable  
)
```

Parameters

nSymId

One of the symbology enumerated types , e.g., SYM_CODE39, SYM_OCR, or SYM_ALL to default all symbologies.

bEnable

If this value is true, symbology of nSymId is enable. If this value is false, symbology of nSymId is disable.

Return Value

true indicates success.

false indicates failure.

4.6.1.11 GetEnableDisableSymbology

This function gets state of the specified symbologies.

```
BOOL GetEnableDisableSymbology(  
    StupType  etType  
    int       nSymId  
    BOOL      *bEnable  
)
```

Parameters

SetType

Use SETUP_TYPE_CURRENT for the current settings, or SETUP_TYPE_DEFAULT for the customer default settings.

nSymId

One of the symbology enumerated types, e.g., SYM_CODE39, SYM_OCR, or SYM_ALL to default all symbologies.

bEnable

If this value is true, symbology of nSymId is enable. If this value is false, symbology of nSymId is disable.

Return Value

true indicates success.

false indicates failure

4.6.1.12 GetInfo

This function gets version information.

```
BOOL GetInfo(  
    PIMAGER_VERSION_INFO  info  
)
```

Parameters

verinfo

reference to a IMAGER_VERSION_INFO structure to receive version information.

Return Value

true indicates success.

false indicates failure.

4.6.1.13 ScanLed

This function turns the terminal's barcode scan led on or off.

```
BOOL ScanLed(  
    BOOL  On  
)
```

Parameters

On

If On is true, turn on barcode scan led.

If On is false, turn off barcode scan led.

Return Value

true indicates success.
false indicates failure.

4.6.1.14 ReadSymbologyFlagsOnlyConfig

This function is used to get the symbology-specific options (symbologies with no specified minimum or maximum length).

```
BOOL ReadSymbologyConfig(  
    SetupType      SetType  
    int            nSymbol  
    PSymFlagsOnly  config  
)
```

Parameters*SetType*

Use SETUP_TYPE_CURRENT for the current settings, or SETUP_TYPE_DEFAULT for the customer default settings.

nSymId

One of the symbology enumerated types , e.g., SYM_CODE39, SYM_OCR.

config

Reference to structure with the symbology-specific options.

Return Value

true indicates success.
false indicates failure.

4.6.1.15 ReadSymbologyFlagsRangeConfig

This function is used to get the symbology-specific options (symbologies with minimum and maximum length).

```
BOOL ReadSymbologyConfig(  
    SetupType      SetType  
    int            nSymbol  
    PSymFlagsRange config  
)
```

Parameters*SetType*

Use SETUP_TYPE_CURRENT for the current settings, or SETUP_TYPE_DEFAULT for the customer default settings.

nSymId

One of the symbology enumerated types, e.g., SYM_CODE39, SYM_OCR.

config

Reference to structure with the symbology-specific options.

Return Value

true indicates success.
false indicates failure.

4.6.1.16 ReadSymbologyOCRConfig

This function is used to get the ocr symbology-specific options.

```
BOOL ReadSymbologyConfig(  
    SetupType      SetType  
    PSymCodeOCR    config  
)
```

Parameters

SetType

Use SETUP_TYPE_CURRENT for the current settings, or SETUP_TYPE_DEFAULT for the customer default settings.

config

Reference to structure with the symbology-specific options.

Return Value

true indicates success.

false indicates failure.

4.6.1.17 WriteSymbologyFlagsOnlyConfig

This function is used to set the symbology-specific options (symbologies with no specified minimum or maximum length).

```
BOOL WriteSymbologyConfig(  
    int          nSymbol  
    PSymFlagsOnly config  
)
```

Parameters

nSymbol

One of the symbology enumerated types, e.g., SYM_CODE39, SYM_OCR.

config

set to structure with the symbology-specific options.

Return Value

true indicates success.

false indicates failure.

4.6.1.18 WriteSymbologyFlagsRangeConfig

This function is used to set the symbology-specific options (symbologies with minimum and maximum length).

```
BOOL WriteSymbologyConfig(  
    int          nSymbol  
    SymFlagsRange config  
)
```

Parameters

nSymbol

One of the symbology enumerated types, e.g., SYM_CODE39, SYM_OCR.

config

set to structure with the symbology-specific options.

Return Value

true indicates success.

false indicates failure.

4.6.1.19 WriteSymbologyOCRConfig

This function is used to set the ocr symbology-specific options.

```
BOOL WriteSymbologyConfig(  
    SymCodeOCR    config  
)
```

Parameters

nSymId

One of the symbology enumerated types, e.g., SYM_CODE39, SYM_OCR.

config

set to structure with the symbology-specific options.

Return Value

true indicates success.

false indicates failure.

4.6.2 Reference and Function List for C#

Definitions

Status Return value

Please refer to the below table for the status value definition.

Status Value Definition	Code	Meaning
TRUE	1	Success
FALSE	0	General Error

Structure

Scanner.SetupType Nested Type - Enumerated type for specifying whether a read configuration item call should return the current settings or the imager default setting.

```
public enum SetupType{
    SETUP_DEFAULT = 0,
    SETUP_CURRENT
}
```

Scanner.SYMID Nested Type - Symbology ID Enumeration.

```
public enum SYMID{
    ID_AZTEC = 0,           Aztec Code           SymFlagsRange
    ID_MESA,                Aztec Mesas       SymFlagsOnly
    ID_CODABAR,             Codabar           SymFlagsRange
    ID_CODE11,              Code 11           SymFlagsRange
    ID_CODE128,             Code 128          SymFlagsRange
    ID_CODE39,              Code 39           SymFlagsRange
    ID_CODE49,              Code 49           SymFlagsRange
    ID_CODE93,              Code 93           SymFlagsRange
    ID_COMPOSITE,           Composite Code     SymFlagsRange
    ID_DATAMATRIX,          Data Matrix       SymFlagsRange
    ID_EAN8,                EAN-8             SymFlagsOnly
    ID_EAN13,               ENA-13            SymFlagsOnly
    ID_INT25,               Interleaved 2 of 5 SymFlagsRange
    ID_MAXICODE,            MaxiCode          SymFlagsRange
    ID_MICROPDF,            Micro PDF417      SymFlagsRange
    ID_OCR,                 Ocr               SymCodeOCR
    ID_PDF417,              PDF417            SymFlagsRange
    ID_POSTNET,             Postnet           SymFlagsOnly
    ID_QR,                  QR Code           SymFlagsRange
    ID_RSS,                 Reduced Space Symbology (RSS) SymFlagsRange
    ID_UPCA,                UPC-A             SymFlagsOnly
    ID_UPCE0,               UPC-E             SymFlagsOnly
    ID_UPCE1,               UPC-E1            SymFlagsOnly
    ID_ISBT,                ISBT              SymFlagsOnly
    ID_BPO,                 British Post       SymFlagsOnly
    ID_CANPOST,             Canadian Post      SymFlagsOnly
    ID_AUSPOST,             Australian Post    SymFlagsOnly
    ID_IATA25,              Straight 2 of 5 IATA SymFlagsRange
    ID_CODABLOCK,           Codablock         SymFlagsRange
    ID_JAPOST,              Japanese Post      SymFlagsOnly
    ID_PLANET,              Planet Code        SymFlagsOnly
    ID_DUTCHPOST,           KIX (Netherlands) Post SymFlagsOnly
    ID_MSI,                 MSI Code          SymFlagsRange
    ID_TLCODE39,            TCIF Linked Code 39 (TLC39) SymFlagsOnly
    ID_TRIOPTIC,            Trioptic Code      SymFlagsOnly
    ID_CODE32,              Code 32           SymFlagsOnly
    ID_STRT25,              Straight 2 of 5 Industrial SymFlagsRange
    ID_MATRIX25,            Matrix 2 of 5      SymFlagsRange
    ID_PLESSEY,             Plessey Code       SymFlagsRange
}
```

ID_CHINAPOST,	China Post	SymFlagsRange
ID_KOREAPOST,	Korean Post	SymFlagsRange
ID_TELEPEN,	Telepen	SymFlagsRange
ID_CODE16K,	Code 16k	SymFlagsRange
ID_POSICODE,	PosiCode	SymFlagsRange
ID_COUPONCODE,	UPC-A with Extended Coupon Code	SymFlagsOnly
ID_USPS4CB,	USPS 4-State Customer	SymFlagsOnly
ID_IDTAG,	UPU 4 State ID Tag	SymFlagsOnly
ID_GS1_128,	GS1 128	SymFlagsRange
ID_GEN_CODE128,	general code 128	SymFlagsRange
ID_ALL = 100		

```

}

```

Scanner.IMAGER_VERSION_INFO Nested Type - Structure for version information.

```

public struct IMAGER_VERSION_INFO{
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 128)]
    public string tcAPIRev;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 128)]
    public string tcDecoderRev;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 128)]
    public string tcScanDriverRev;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 1000)]
    public string tcEtcInfo;
    public int dwFirmwareVersion;
    public int dwFirmwareCksum;
    public int dwEngineId;
}

```

Scanner.ScanIlluminat Nested Type - Enumerated integer type that identifies possible illumination modes used during image acquisition.

```

public enum ScanIlluminat{
    SCAN_ILLUM_AIMER_OFF = 0,
    SCAN_ILLUM_ONLY_ON,
    SCAN_AIMER_ONLY_ON,
    SCAN_ILLUM_AIMER_ON,
}

```

Scanner.DECODE_MSG Nested Type - Data structure that holds the decoded bar code message.

```

public struct DECODE_MSG{
    public int dwStructSize;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 4096)]
    public string pchMessage;
    public ushort chCodeID;
    public ushort chSymLetter;
    public ushort chSymModifier;
    public int nLength;
}

```

Scanner.EVENT_TYPE Nested Type - Type of data causing the event notification

```

public enum EVENT_TYPE{
    BARCODE_EVENT = 0,
    IMAGE_EVENT,
    TEXT_MSG_EVENT,
    INTELIMG_BARCODE_EVENT,
    INTELIMG_IMAGE_EVENT,
    TRIGGER_EVENT
}

```

Scanner.SymCodeOCR Nested Type - Structure for unusual OCR

```

public enum OCRMode{
    OCR_MODE_DISABLED = 0,
    OCR_MODE_A,
}

```

```

    OCR_MODE_B,
    OCR_MODE_MONEY,
    OCR_MODE_MICR_UNSUPPORTED,
} ;

public enum OCRDirection{
    OCR_DIRECTION_LeftToRight = 0,
    OCR_DIRECTION_TopToBottom,
    OCR_DIRECTION_RightToLeft,
    OCR_DIRECTION_BottomToTop,
} ;

public struct SymCodeOCR{
    public OCRMode ocrMode;
    public OCRDirection ocrDirection;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 256)]
    public string tcTemplate;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 256)]
    public string tcGroupG;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 256)]
    public string tcGroupH;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 64)]
    public string tcCheckChar;
}

```

Scanner.SymFlagsOnly Nested Type - Structure for symbologies with no specified minimum or maximum length.

```

public struct SymFlagsOnly{
    public int    nFlags;
}

```

Scanner.SymFlagRange Nested Type - Structure for symbologies with minimum and maximum length.

```

public struct SymFlagRange{
    public int    nFlags;
    public int    nMinLen;
    public int    nMaxLen;
}

```

4.6.2.1 Imager.Connect Method

This function connects the engine.

```
public bool Connect(  
    bool    On  
)
```

Parameters

On

If this value is true, connects the engine.

If this value is false, disconnects the engine.

Return Value

true indicates success.

false indicates failure.

4.6.2.2 Imager.AimerOn Method

This function turns the engine's aiming mechanism on or off.

```
public bool AimerOn(  
    bool    On  
)
```

Parameters

On

If this value is true, turns on the aimer.

If this value is false, turns off the aimer.

Return Value

true indicates success.

false indicates failure.

4.6.2.3 Imager.GetInfo Method

This function gets version information.

```
public bool GetInfo(  
    ref    IMAGER_VERSION_INFO    verinfo  
)
```

Parameters

verinfo

reference to a **IMAGER_VERSION_INFO** structure to receive version information.

Return Value

true indicates success.

false indicates failure.

4.6.2.4 Imager.LightsOn Method

This function turns the engine's illumination LEDs on and off.

```
public bool LightsOn(  
    bool    On  
)
```

Parameters

On

If TRUE, the illumination LEDs are turned on; otherwise they're turned off.

Return Value

true indicates success.

false indicates failure.

4.6.2.5 Imager.SetScanningLightsMode Method

This function gives the user the ability to select what the illumination and aimer do during imaging.

```
public bool SetScanningLightsMode(  
    ScanIlluminat    nIllumMode  
)
```

Parameters

ScanIlluminat

SCAN_ILLUM_AIMER_OFF=0 Neither aimers nor illumination

SCAN_ILLUM_ONLY_ON Illumination only

SCAN_AIMER_ONLY_ON Aimers only

SCAN_ILLUM_AIMER_ON Both aimers and illumination

Return Value

true indicates success.

false indicates failure.

4.6.2.6 Scanner.CancelIo Method

This function cancels the current bar code capture.

```
public bool CancelIo( )
```

Parameters

None

Return Value

true indicates success.

false indicates failure.

4.6.2.7 Scanner.DefaultSymbology Method

This function sets the specified symbologies to their factory default configurations.

```
public bool DefaultSymbology(  
    SYMID    nSymId  
)
```

Parameters

nSymId

One of the symbology enumerated types, e.g., SYM_CODE39, SYM_OCR, or SYM_ALL to default all symbologies.

Return Value

true indicates success.

false indicates failure.

4.6.2.8 Scanner.GetEnableDisableSymbology Method

This function gets status of the specified symbologies

```
public bool GetEnableDisableSymbology(  
    SetupType      SetType  
    SYMID          nSymId  
    ref bool       Enable  
)
```

Parameters

SetType

Use SETUP_TYPE_CURRENT for the current settings, or SETUP_TYPE_DEFAULT for the customer default settings.

nSymId

One of the symbology enumerated types, e.g., SYM_CODE39, SYM_OCR to enable or disable symbologies.

Enable

Reference to a Enable value to receive status of symbology.

If this value is true, symbology of nSymId is enable. If this value is false, symbology of nSymId is disable.

Return Value

true indicates success.

false indicates failure.

4.6.2.9 Scanner.GetScanResult Method

This function retrieves the data from the last signal event (bar code capture).

```
public bool GetScanResult(  
    ref  EVENT_TYPE EventType  
    ref  DECODE_MSG Msg  
)
```

Parameters

EventType

Reference to a EVENT_TYPE structure to receive event type.

Msg

Reference to a DECODE_MSG structure to receive the data decoded.

Return Value

true indicates success.

false indicates failure.

4.6.2.10 Scanner.ReadSymbologyConfig Method

This function is used to get the symbology-specific options.

```
public bool ReadSymbologyConfig(  
    SetupType      SetType  
    SYMID          nSymId  
    ref            SymFlagsOnly config  
)  
public bool ReadSymbologyConfig(  
    SetupType      SetType  
    SYMID          nSymId  
    ref            SymFlagsRange  config  
)  
public bool ReadSymbologyConfig(  

```

```

        SetupType      SetType
        ref            SymCodeOCR config
    )

```

Parameters

SetType

Use SETUP_TYPE_CURRENT for the current settings, or SETUP_TYPE_DEFAULT for the customer default settings.

nSymId

One of the symbology enumerated types, e.g., SYM_CODE39, SYM_OCR.

config

Reference to structure with the symbology-specific options.

Return Value

true indicates success.

false indicates failure.

4.6.2.11 Scanner.ScanLed Method

This function turns the terminal's barcode scan led on or off.

```

public bool ScanLed(
    bool    On
)

```

Parameters

On

If On is true, turn on barcode scan led.

If On is false, turn off barcode scan led.

Return Value

true indicates success.

false indicates failure.

4.6.2.12 Scanner.ScanRead Method

This function causes the engine to start scanning for a decodable symbol.

```

public bool ScanRead(
    int      dwTimeout
    ref      DECODE_MSG pmsg
)

```

Parameters

dwTimeout

Maximum time (in seconds) to attempt to decode before declaring a no decode.

A ScannerTimeoutDelegate event specifies that the timeout is whatever is currently set on the imager.

A value of 0 indicates no timeout.

pmsg

Reference to a DECODE_MSG structure to receive the data decoded.

Return Value

true indicates success.

false indicates failure.

4.6.2.13 Scanner.SetEnableDisableSymbology Method

This function sets state of the specified symbologies.

```
public bool SetEnableDisableSymbology(  
    SYMID      nSymId  
    bool       bEnable  
)
```

Parameters

nSymId

One of the symbology enumerated types, e.g., SYM_CODE39, SYM_OCR to enable or disable symbologies.

bEnable

If this value is true, symbology of nSymId is enable. If this value is false, symbology of nSymId is disabled.

Return Value

true indicates success.

false indicates failure.

4.6.2.14 Scanner.WriteSymbologyCofig Method

This function is used to set the symbology-specific options.

```
public bool WriteSymbologyCofig(  
    SYMID      nSymId  
    SymFlagsOnly  config  
)  
public bool WriteSymbologyCofig(  
    SYMID      nSymId  
    SymFlagsRange config  
)  
public bool WriteSymbologyCofig(  
    SymCodeOCR  config  
)
```

Parameters

nSymId

One of the symbology enumerated types, e.g., SYM_CODE39, SYM_OCR to enable or disable symbologies.

config

set to structure with the symbology-specific options.

Return Value

true indicates success.

false indicates failure.