



# **Programmer's Guide: Using the M3 T SDK**

**Version: 1.2.0.T**

**Release Date: December 23, 2010**

# 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 Initialization .....                | 18 |
| 2.2.2 Close Camera .....                  | 18 |
| 2.2.3 Capture Image .....                 | 19 |
| 2.2.4 Flash On / Off .....                | 19 |
| 2.2.5 Preview Start / Stop.....           | 20 |
| 2.3 GPS .....                             | 21 |
| 2.3.1 Open GPS .....                      | 21 |
| 2.3.1 Close GPS .....                     | 21 |
| 2.3.3 Receive Message to GPS Module ..... | 22 |
| 2.4 Scanner 1D .....                      | 23 |
| 2.4.1 Initialization .....                | 23 |
| 2.4.2 Default Settings.....               | 23 |
| 2.4.3 ScanRead and GetDecodeData .....    | 27 |
| 2.4.4 Close Scanner .....                 | 29 |
| 2.5 Scanner 2D (Imager).....              | 30 |
| 2.5.1 Initialization .....                | 30 |
| 2.5.2 Default Settings.....               | 31 |
| 2.5.3 ScanRead and GetDecodeData .....    | 31 |
| 2.5.4 Close Scanner .....                 | 35 |
| 3.0 Samples.....                          | 36 |
| 3.1 Bluetooth .....                       | 36 |

|       |   |           |
|-------|---|-----------|
| 3.2   | Camera.....                               | 36        |
| 3.3   | GPS .....                                 | 36        |
| 3.4   | Scanner 1D (Software Decoder only).....   | 36        |
| 3.5   | Scanner 2D (Imager).....                  | 36        |
| 4.0   | References (Function Lists) .....         | <b>37</b> |
| 4.1   | Bluetooth .....                           | 38        |
| 4.1.1 | Reference and Function List for C++ ..... | 39        |
| 4.1.2 | Reference and Function List for C# .....  | 76        |
| 4.1.3 | Error Codes for Bluetooth.....            | 88        |
| 4.2   | Camera.....                               | 96        |
| 4.2.1 | Reference and Function List for C++ ..... | 97        |
| 4.2.2 | Reference and Function List for C# .....  | 108       |
| 4.3   | GPS .....                                 | 116       |
| 4.3.1 | Reference and Function List for C++ ..... | 117       |
| 4.3.2 | Reference and Function List for C# .....  | 121       |
| 4.4   | Scanner 1D .....                          | 125       |
| 4.4.1 | Reference and Function List for C++ ..... | 126       |
| 4.4.2 | Reference and Function List for C# .....  | 135       |
| 4.5   | Scanner 2D (Imager).....                  | 159       |
| 4.5.1 | Reference and Function List for C++ ..... | 160       |
| 4.5.2 | Reference and Function List for C# .....  | 171       |

# 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 2003 (7.1) (.NET Framework 1.1) – Visual C++, Visual C#, Visual Basic .NET
- Visual Studio 2005 (8.0) (.NET Framework 2.0) – Visual C++, Visual C#, Visual Basic .NET

### Development System Requirements

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

### Development Platform Requirements

- M3 T Platform SDK must be installed on your computer to be able to develop softwares using this SDK.
  - After installation, please select 'M3Plus' at the platform selection on EVC.
  - Click [HERE](#) to download M3 T Platform.

## Release History

### M3 T SDK Version 1.2.0

- Re-structured SDK manual (2010.12.06)
  - Separate manual is provided for M3 T
  - Added Tutorial, Samples and References (function lists)

## 1.0 Introduction

This document is a reference guide for the software developer's kit (SDK) for M3 T (WinCE 5.0). 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.  |
| HSDPA <sup>[1]</sup> | An SMS can be sent or received and it also allows data communication through the network.   |
| GPS                  | Gathers information on current location through satellite. COM3 is used for GPS.  |
| 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 (MSD10G) module is used.<br><b>(WLAN SDK is NOT included in this document)</b> |

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

HSDPA<sup>[1]</sup> : HSDPA is not support at the moment. It will be available in Q1 2011.

## 2.0 Tutorial

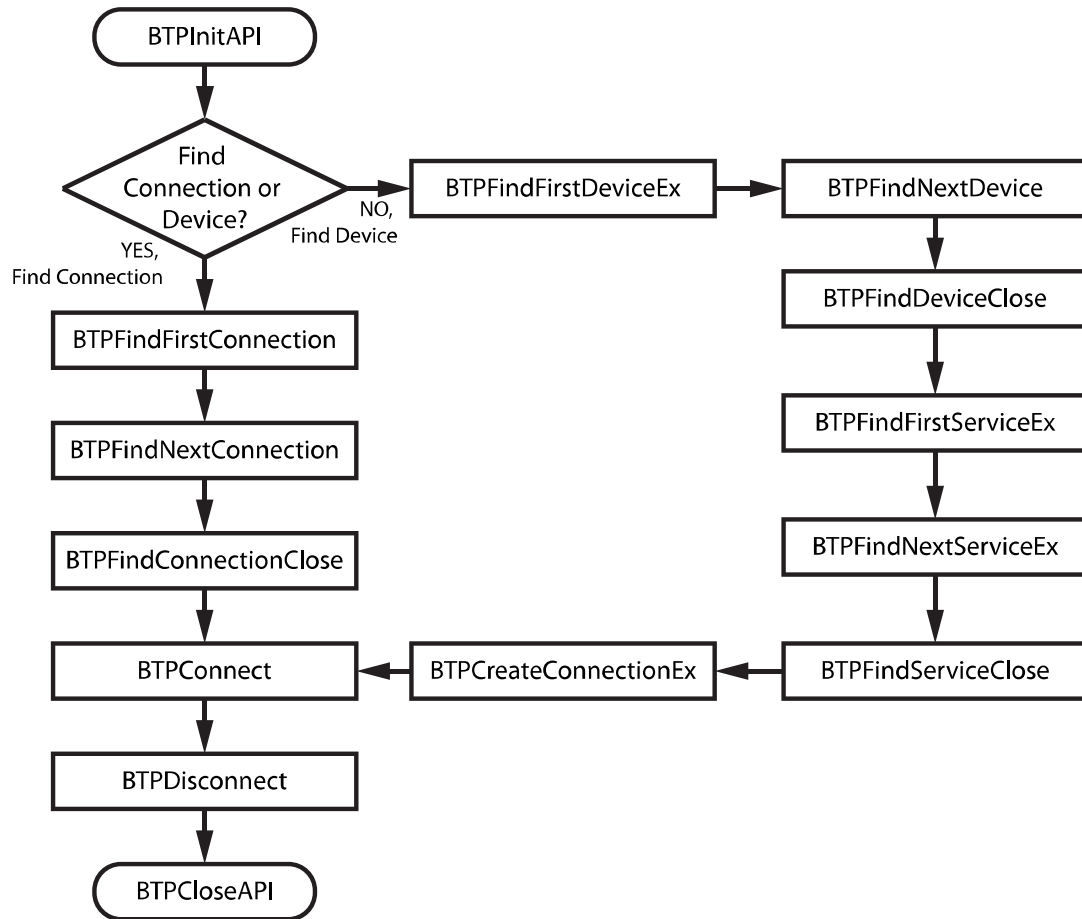
This chapter describes the basic usage of M3 Mobile SDK functions in a step-by-step manner. In this tutorial section, the following topics are treated.

| Section    | Topic                         |
|------------|-------------------------------|
| Bluetooth  | Initialization                |
|            | Create Connection             |
|            | Connect                       |
|            | Find Connection               |
|            | Find Device                   |
|            | Find Service                  |
|            | Delete Connection             |
|            | Disconnect                    |
|            | Close                         |
| Camera     | Initialization                |
|            | Close Camera                  |
|            | Capture Image                 |
|            | Flash On / Off                |
|            | Preview Start / Stop          |
| GPS        | Open GPS                      |
|            | Close GPS                     |
|            | Receive Message to GPS Module |
| 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 an default option.

Please refer to below flow charts 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_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");

```

## 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#

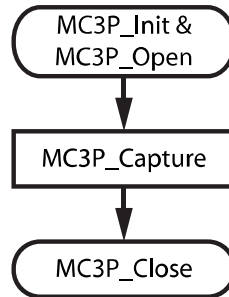


```
using BTEAPIdllNet;  
  
M3BTEDllNet BteDll;  
  
BteDll = new M3BTEDllNet();  
  
if (BteDll.M3BTPClose())  
    MessageBox.Show("close success");  
else  
    MessageBox.Show("fail");
```

## 2.2 Camera

This SDK is applicable to 3.0 mega pixel camera.

Please refer to below flow charts for Bluetooth.



**Simple Camera flow chart**

### 2.2.1 Initialization

In Camera Init, MC3P\_Init and MC3P\_Open function are needed. First, MC3P\_Init function needs to be called. Its parameters are HWND which runs camera and Window Control Handle which outputs the preview. Then, MC3P\_Open starts the camera program by inputting the Window Handle of registered Control.

#### C++

```
CStatic m_ctrlpreview;
CM3P_Camera m_m3p_camera; // CM3P_Camera is the camera dll class.

void CameraInit()
{
    // m_ctrlpreview is a static tool on camera dlg
    m_m3p_camera.MC3P_Init(m_hWnd,m_ctrlpreview.m_hWnd))
    //return device type 0: 6300, 1: 6400, 2:6500

    if(!m_m3p_camera.MC3P_Open())
    {
        AfxMessageBox(L"Com Open Error");
        return FALSE;
    }
}
```

#### C#

```
using M3p_cam_net;
private M3p_cam_net.CamCore camctrl;
public CameraInit()
{
    camctrl = new CamCore();
    camctrl.MC3P_Init(this.Handle, pictureBox1.Handle);

    if(!camctrl.MC3P_Open())
    {
        MessageBox.Show("open error");
    }
}
```

### 2.2.2 Close Camera

If preview is running when trying to terminate the program, preview should be stopped first. Then, the camera program is closed through Close Camera.

## C++

```
void Camera Close()  
{  
    m_m3p_camera.MC3P_PreViewStop();  
    m_m3p_camera.MC3P_Close();  
}
```

## C#

```
void CameraClose()  
{  
    camctrl.MC3P_Close();  
}
```

### 2.2.3 Capture Image

Capture Image captures the Still Shot of Preview screen.

## C++

```
void CaptureImage()  
{  
    m_m3p_camera.MC3P_Capture();  
}
```

## C#

```
void CaptureImage()  
{  
    camctrl.MC3P_Capture();  
}
```

### 2.2.4 Flash On / Off

Flash On/Off function is helping camera captures.

## C++

```
void CameraFlashOn(BOOL bOn)  
{  
    if(bOn == TRUE)  
    {  
        m_m3p_camera.MC3P_FlashON();  
    }  
    else  
    {  
        m_m3p_camera.MC3P_FlashOFF();  
    }  
}
```

## C#

```
void CameraFlashOn(BOOL bOn)  
{  
    if(bOn == TRUE)  
    {  
        camctrl.MC3P_FlashON();  
    }  
    else  
    {  
        camctrl.MC3P_FlashOFF();  
    }  
}
```

### 2.2.5 Preview Start / Stop

Preview Start/Stop toggles the preview of the image that are input to the camera.

#### C++

```
void PreviewStart(BOOL bOn)
{
    if(bOn == TRUE)
    {
        m_m3p_camera.MC3P_PreviewStart();
    }
    else
    {
        m_m3p_camera.MC3P_PreviewStop();
        m_ctrpreview.Invalidate();
    }
}
```

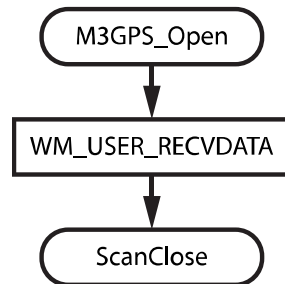
#### C#

```
void PreviewStart(BOOL bOn)
{
    if(bOn == TRUE)
    {
        camctrl.MC3P_PreviewStart();
    }
    else
    {
        camctrl.MC3P_PreviewStop();
        m_ctrpreview.Invalidate();
    }
}
```

## 2.3 GPS

In M3 T, GPS can be used via COM3.

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");
    }
}
```

### 2.3.1 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 to 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();

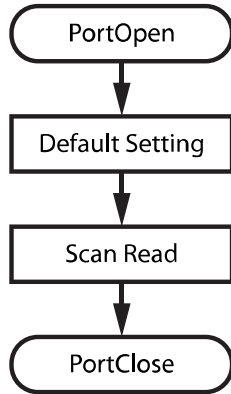
    try
    {
        info = (Class_Gps_Parse.GPS_PARSE_INFO)Marshal.PtrToStructure(wParam,
            typeof(Class_Gps_Parse.GPS_PARSE_INFO));

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

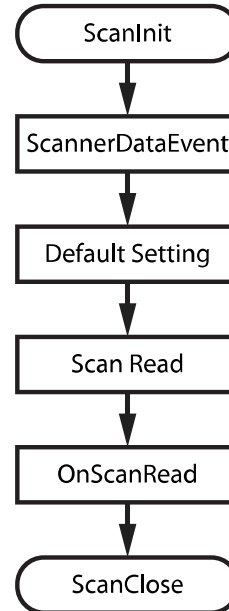
## 2.4 Scanner 1D

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#**

### 2.4.1 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 : 115200

#### C++

```
[ Header File]

#include "KScanBar.h"
#include "MC9000_IoCtl.h" // M3Pos Power Control

CKScan m_KScan;
[CPP File]
// The Scanner Port Open
Int nPort = 6;

bRet = m_KScan.Open(nPort, FALSE, CBR_115200, FALSE, NULL);
if(bRet == FALSE)
{
    ::MessageBox(NULL, L"Error : Scanner Open Failed", NULL, MB_TOPMOST);
    Return FALSE;
}
```

In C#, unlike in C++, ScanCtrl.ScanInit function does all necessary steps at once. ScanCtrl.ScanInit() 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 System.Threading; // Sleep
using MCSSLibNet;

namespace M3ScanTest_Net
{
    public partial class M3Scanner : Form
    {
        private MCSSLibNet.ScannerControl ScanCtrl;
        public int m_bResult;

        public M3Scanner()
        {
            InitializeComponent();
            ScanCtrl = new ScannerControl();
            ScanCtrl.ScannerDataEvent += new ScannerDataDelegate(OnScanRead);
        }
        private void M3Scanner_Load(object sender, EventArgs e)
        {
            // Scanner Initializtion
            m_nResult = ScanCtrl.ScanInit();
            if(m_nResult != 0)
            {
                MessageBox.Show("Error : Scanner Init Failed");
            }
        }
    }
}
```

### 2.4.2 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 CMainSheet::DefaultSetting()
{
    // KSCANREAD kRead initialization
    memset(&kRead, 0, sizeof(kRead));
    kRead.nSize = sizeof(kRead);

    // KSCANREADEx2 kReadEx2 initialization
```



```

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

kRead.nTimeInSeconds          = 10;           // time out
kRead.nMinLength              = 2;           // minimum data length
kRead.nSecurity                = 1           // security level
kReadEx2.XmitAIMID            = DISABLE;     // Transmit AIMID

// KScanReadCallBack
kRead.pUserData                = this;
kRead.fnCallBack               = KScanReadCallBack;
kRead.dwFlags                  = KSCAN_FLAG_WIDESCANANGLE;

//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;
}

```

In C#, unlike in C++, ScanCtrl.Default\_Setting function does all necessary initialization and restores to its default settings.

- Enable all barcodes.
- Timeout: 10 / SecurityLevel: 1 / AsyncMode.

## C#

```
ScanCtrl.Default_Setting();
```

### 2.4.3 ScanRead and GetDecodeData

This function reads and gets decoded data.

m\_KScan.Read() function is called to Trigger On scanner.  
KScanReadCallBack() function acquires the actual scan data.

## C++

```
int KSCANAPI KScanReadCallBack(LPVOID pRead);
...
void CMainSheet::M3_ScanRead()
{
    CString      Str;
    BOOL         bRet;
    m_bReading   = TRUE;

    if (m_bReading)
    {
        // Reading already in progress, now cancel it.
        bRet = m_KScan.ReadCancel();
    }

    m_bReading = TRUE;
    bRet = m_KScan.Read(&kRead);
    if (!bRet) {
        ::MessageBox(NULL, L"Error: ScanRead Failed", NULL, MB_TOPMOST);
        m_bReading = FALSE;
    }
}

int KSCANAPI KScanReadCallBack(LPVOID pRead)
{
    int          Status;
    int          Type;

    CMainSheet* lpCls = (CMainSheet*)((PKSCANREAD)pRead)->pUserData;
    Status = ((PKSCANREAD)pRead)->out_Status;

    switch(Status) {
    case KSCAN_RET_TIMEOUT:           // Timed out.
        Status = 0;
        break;
    case KSCAN_RET_USER_CANCEL:       // User called stop
        Status = 0;
        break;

    case KSCAN_RET_NORMAL:           // Barcode was read
    case KSCAN_RET_TYPE_UNKNOWN:
        Status = 0;
        if(((PKSCANREAD)pRead)->out_Type == -1)
            break;
        if(((PKSCANREAD)pRead)->out_Status == KSCAN_RET_NORMAL)
        {
            if(!lpCls->m_bReading)return 0;
            Type = ((PKSCANREAD)pRead)->out_Type;
            info.gstrCodeTypeName = _T("");
            lpCls->SetBarCodeString(Type);
            info.gstrBarCodeValue = _T("");
            info.gstrBarCodeValue = ((PKSCANREAD)pRead)->out_Barcode;
            ::PostMessage(lpCls->m_hWnd, WM_DATA_READ, 0, 0);
        }
    }
}
```

```

        lpCls->LoadResourceSound();
    }
    lpCls->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
    break;
}
return Status;
}

```

- 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 MCSSLibNet.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;
    }

    m_bReading = false;
    ScanCtrl.ScanRead();
}

public void OnScanRead(object sender, ScannerDataArgs e)
{
    if (LV_SCANDATA.Items.Count > 6)
        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));
    }
    m_bReading = false;
}

```

#### 2.4.4 Close Scanner

Terminating the scanner is done by closing the com port.

In M3 T, closing the scanner port will automatically cut the power of the scanner module.

##### C++

```
BOOL bRet = FALSE;
Int i = 0;
DWORD dwCpldID = CPLD_ID_SCAN_ON;
DWORD dwOut = 0;
for(i=0;i<3;i++)
{
    // Scanner Port Close
    bRet = m_KScan.Close();
    if(bRet)
        break;
    Sleep(100);
}
If(bRet == FALSE)
    ::MessageBox(NULL, L"Error: Scanner Close Failed", NULL, MB_TOPMOST);
```

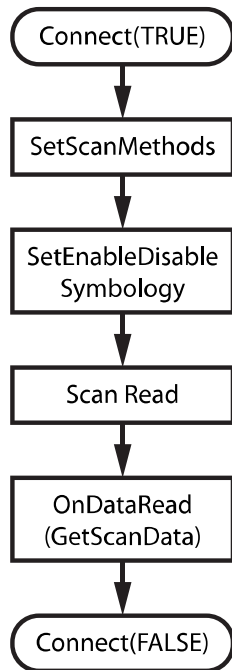
##### C#

```
private void M3Scanner_Closing(object sender, CancelEventArgs e)
{
    ScanCtrl.ScanClose();
}
```

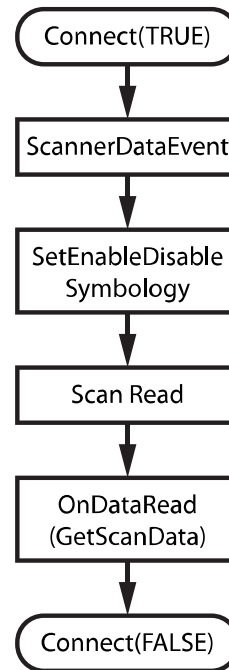
## 2.5 Scanner 2D (Imager)

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#**

### 2.5.1 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");
    }
};
}

```

### 2.5.2 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);

```

### 2.5.3 ScanRead and GetDecodeData

This function reads and get 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(TimeoutTextBox.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;
    }
}

```

#### **2.5.4 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_2.exe     | BTE_sample_2.exe     | 1.0.1   | 2010-12-14 | Visual Studio 2005 |
| C#.Net | BTE_sample_CS_ce.exe | BTE_sample_CS_ce.exe | 1.0.1   | 2010-12-14 | Visual Studio 2005 |

### 3.2 Camera

| Type   | Demo  | Source               | Version | Date       | Tool               |
|--------|---|----------------------|---------|------------|--------------------|
| C++    | M3P_CameraTest.exe<br>M3P_Camera.dll                      | M3P_CameraTest.exe   | 2.0.2   | 2010-10-27 | Visual Studio 2005 |
| C#.Net | M3P_Cam_Net_Test.exe<br>M3p_cam_net.dll<br>M3P_Camera.dll | M3P_Cam_Net_Test.exe | 1.0.1   | 2010-12-03 | Visual Studio 2005 |

### 3.3 GPS

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

### 3.4 Scanner 1D (Software Decoder only)

| Type   | Demo  | Source             | Version | Date       | Tool                    |
|--------|---|--------------------|---------|------------|-------------------------|
| C++    | M3ScanTest.exe<br>ScanEmul.exe<br>KScanBar.dll      | M3ScanTest.exe     | 3.0.0   | 2010-06-21 | eMbedded Visual C++ 4.0 |
| C#.Net | M3ScanTest_Net.exe<br>MCSSLib.dll<br>MCSSLibNet.dll | M3ScanTest_Net.exe | 3.1.0   | 2010-10-27 | Visual Studio 2005      |
| VB.Net | M3ScanTest_VB.exe<br>MCSSLib.dll<br>MCSSLibNet.dll  | M3ScanTest_VB.exe  | 2.1.0   | 2010-10-27 | Visual Studio 2005      |

### 3.5 Scanner 2D (Imager)

| Type   | Demo   | Source             | Version | Date       | Tool                    |
|--------|--|--------------------|---------|------------|-------------------------|
| C++    | M3ScanTest.exe<br>ScanEmul.exe<br>TestCam.exe<br>M3MobileImager.dll                  | M3ScanTest.exe     | 2.1.0   | 2010-04-22 | eMbedded Visual C++ 4.0 |
|        |  | TestCam.exe        | 2.0.0   | 2010-03-08 | eMbedded Visual C++ 4.0 |
| C#.Net | M3ScanTest_Net.exe<br>TestCam_Net.exe<br>M3MobileImager.dll<br>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 T. APIs are described using C language. Applications are created using Visual C++ 6.0 and they are compatible with Visual C++ 7.0 / 7.1 / 8.0 or higher.

For accessing C / C++ style API in Visual Basic 6.0, we supply M3 Mobile SDK 1.2.0 Type Library for Visual Basic 6.0. 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 Bluetooth 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:

BTEAPI.dll, BTEAPI.dllNet.dll

### Supported Product

M3 T with Stonestreet One BT Stack

#### 4.1.1 Reference and Function List for C++

##### Definitions

##### Structure

**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;
    BTP_Service_Find ServiceHandle;
    //not used in version 2. For future enhancements.
    unsigned int ServiceIndex;
    //not used in version 2. For future enhancements.
    union
    {
        BTPSerialPortProfileInfo_t RemoteSerialPortProfileInfo;
        BTPHIDProfileInfo_t RemoteHIDProfileInfo;
    }ProfileInformation;
}BTP_Connection_Info_Ex_t;
```

##### **BTP\_Connection\_Info\_Ex\_t – Field Descriptions**

|                                    |  |
|------------------------------------|--|
| <b>Size</b>                        | Fill with the size of the BTP_Connection_Info_Ex_t structure.  |
| <b>Version</b>                     | Not used by the application code in this version.  |
| <b>ConnectionID</b>                | Unique identifier for the connection supplied by a call to BTPCreateConnnection or BTPCreateConnnectionEx.   |
| <b>BD_ADDR</b>                     | Address of the remote device hosting the connection.   |
| <b>LocalCOMPort</b>                | 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.  |
| <b>MajorVersion</b>                | Major version number of the connection's profile.  |
| <b>MinorVersion</b>                | Minor version number of the connection's profile.  |
| <b>ConnectionAttributes</b>        | Bit-mask defining the connection's attributes. Valid attributes include:<br>BTP_CONNECTION_NONE<br>BTP_CONNECTION_REMEMBERED<br>BTP_CONNECTION_ACTIVE<br>BTP_CONNECTION_ALL  |
| <b>ProfileType</b>                 | Defines the type of profile used by the connection. Valid values include:<br>BTP_PROFILE_SPP<br>BTP_PROFILE_HID_DEVICE<br>BTP_PROFILE_UNKNOWN  |
| <b>ListHandle</b>                  | Not used Currently. For future enhancements.   |
| <b>ListIndex</b>                   | Not used Currently. For future enhancements.   |
| <b>RemoteSerialPortProfileInfo</b> | 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_Ex_t |

|                             |   |
|-----------------------------|---|
| <b>RemoteHIDProfileInfo</b> | <p>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_Ex_t</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> |
|-----------------------------|---|

|   |
|---|
| <p><b>BTP_Device_Info_t</b> – This structure defines a remote Bluetooth device.</p> <pre> /* Structure containing information regarding a remote device. */ typedef struct _tagBTP_Device_Info_t {     BD_ADDR_t BD_ADDR;     Class_of_Device_t ClassOfDevice;     Word_t DeviceAttributes;     char Name[BLUETOOTH_MAX_NAME_SIZE]; } BTP_Device_Info_t; </pre> |
|---|

|  |   |
|--|---|
| <b>BTP_Device_Info_t – Field Description</b> |   |
| <b>BD_ADDR</b>                               | Address of the remote Bluetooth device.   |
| <b>ClassOfDevice</b>                         | Bit-mask list of features that determine the class of device for the remote Bluetooth device.   |
| <b>DeviceAttributes</b>                      | Bit-mask defining the remote device's attributes. Valid attributes include:<br>BTP_DEVICE_NONE<br>BTP_DEVICE_AUTHENTICATED<br>BTP_DEVICE_REMEMBERED<br>BTP_DEVICE_CONNECTED<br>BTP_DEVICE_ALL |
| <b>Name</b>                                  | Name of the remote Bluetooth device.  |

|   |
|---|
| <p><b>BTP_Device_Query_Ex_t</b> – This structure provides parameters for device searches and supports searching for HID devices only.</p> <pre> /* Structure specifying the criteria for device searches. */ 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; </pre> |
|---|



### BTP\_Device\_Query\_Ex\_t – Field Description

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

**BTP\_Service\_Info\_Ex\_t** – This structure defines a service offered by a remote Bluetooth device. This currently supports SPP and HID service information.

```
/* Structure containing remote service or Favorite device 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      ServiceIndex; //not used in version 2. For future enhancements

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

### BTP\_Service\_Info\_Ex\_t – Field Description

|                     |   |
|---------------------|---|
| <b>Size</b>         | Fill with the size of the BTP_Service_Info_Ex_t structure.  |
| <b>Version</b>      | Not used by the application code in this version.   |
| <b>ProfileType</b>  | Defines the type of profile for the remote service. Valid values include:<br><br>BTP_PROFILE_SPP<br>BTP_PROFILE_HID_DEVICE<br>BTP_PROFILE_UNKNOWN |
| <b>MajorVersion</b> | Major version number of the service's profile.  |
| <b>MinorVersion</b> | Minor version number of the service's profile.  |
| <b>ServiceName</b>  | Name of the remote service  |
| <b>ListIndex</b>    | Not used currently. For future enhancements.  |

|                                    |  |
|------------------------------------|--|
| <b>RemoteSerialPortProfileInfo</b> | 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.  |
| <b>RemoteHIDProfileInfo</b>        | <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> |

|  |
|--|
| <b>BTP_Service_Query_t</b> – This structure defines the parameters for an SDP service query.   |
| <pre> /* Structure specifying criteria for service queries. */ typedef struct _tagBTP_Service_Query_t {     BD_ADDR_t      BD_ADDR;     Word_t         NumberServiceUUID;     SDP_UUID_Entry_t *Service; } BTP_Service_Query_t; </pre> |

**BTP\_Service\_Query\_t** – Field Description

|                          |   |
|--------------------------|---|
| <b>BD_ADDR</b>           | Address of the remote Bluetooth device whose services are being queried.  |
| <b>NumberServiceUUID</b> | 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.                          |
| <b>Service</b>           | 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. |

|  |
|--|
| <b>SDP_UUID_Entry_t</b> – This structure defines the parameters for an SDP query.  |
| <pre> typedef struct _tagSDP_UUID_Entry_t {     SDP_Data_Element_Type_t SDP_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; </pre> |

**SDP\_UUID\_Entry\_t** – Field Description

|                          |   |
|--------------------------|---|
| <b>Data_Element_Type</b> | Specifies which field from the UUID_Value union is valid. Valid values include:<br>deUUID_16<br>deUUID_32<br>deUUID_128 |
| <b>UUID_Value</b>        | 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;
```

**BTP\_Connection\_Info\_t** – Holds the information about a connection to be made or searched. This currently supports SPP connections.

```
/* Structure containing information about a Bluetooth connection. */
typedef struct _tagBTP_Connection_Info_t
{
    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

|                             |   |
|-----------------------------|---|
| <b>ConnectionID</b>         | Unique identifier for the connection supplied by a call to BTPCreateConnnection.  |
| <b>BD_ADDR</b>              | Address of the remote device hosting the connection.  |
| <b>RFCOMMPort</b>           | RFCOMM server port used for the connection.   |
| <b>LocalCOMPort</b>         | 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.                                       |
| <b>MajorVersion</b>         | Major version number of the connection's profile.   |
| <b>MinorVersion</b>         | Minor version number of the connection's profile.   |
| <b>ConnectionAttributes</b> | Bit-mask defining the connection's attributes. Valid attributes include:<br><br>BTP_CONNECTION_NONE<br>BTP_CONNECTION_REMEMBERED<br>BTP_CONNECTION_ACTIVE<br>BTP_CONNECTION_ALL |
| <b>ProfileType</b>          | Defines the type of profile used by the connection. Valid values include:<br><br>BTP_PROFILE_SPP<br>BTP_PROFILE_UNKNOWN   |

**BTP\_Connection\_Query\_t** – This structure specifies search criteria for connection searches.

```
/* Structure specifying search criteria for device searches. */
typedef struct _tagBTP_Connection_Query_t
{
    Word_t ConnectionAttributes;
} BTP_Connection_Query_t;
```

**BTP\_Connection\_Query\_t** – Field Description

|                             |  |
|-----------------------------|--|
| <b>ConnectionAttributes</b> | Bit-mask that filters the enumeration of connections by attribute. Valid attributes include:<br><br>BTP_CONNECTION_NONE<br>BTP_CONNECTION_REMEMEBER<br>BTP_CONNECTION_ACTIVE<br>BTP_CONNECTION_ALL |
|-----------------------------|--|

#### 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_Ex_t *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_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"crete 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_Ex_t *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.

*DDeviceInfo*

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 device, 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_Ex_t *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_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);
}

```

#### 4.1.1.17 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 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;

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

#### BT\_SERVICE

```
struct BT_SERVICE
{
    byte    protocolDescriptor; // Protocol descriptor specifying the protocol
    byte    profileDescriptor;  // Profile descriptor specifying the profile
    ushort   identifier;        // Protocol specific identifier
    byte[]   serviceName;       // Service name, if a name was registered
}
```

The **BT\_SERVICE** structure defines service related information.

### Members

#### *protocolDescriptor*

Specifies the underlying protocol. Can be one of the following.

| Value         | Meaning                   |
|---------------|---------------------------|
| <b>L2CAP</b>  | Specifies L2CAP protocol  |
| <b>RFCOMM</b> | Specifies RFCOMM protocol |

#### *profileDescriptor*

Specifies the underlying protocol. Can be one of the following.

| Value      | Meaning                         |
|------------|---------------------------------|
| <b>SPP</b> | Specifies Serial Port Profile   |
| <b>OPP</b> | Specifies Object Push Profile   |
| <b>FTP</b> | Specifies File Transfer Profile |

#### *identifier*

Protocol specific identifier. Will be PSM when protocol descriptor is L2CAP and server channel when protocol descriptor is RFCOMM.

#### *serviceName*

Specifies the name of the service, if a name was registered.

#### BT\_DEVICE

```
struct BT_DEVICE
{
    byte[]    bdAddress;    // Bluetooth device address
    byte      pageScanRepetitionMode;
                // page scan repetition mode of the device
    byte      pageScanPeriodMode; // page scan period mode of the device
    byte      pageScanMode;       // page scan mode of the device
    byte      serviceClass;       // service class of the device
    byte      majorDeviceClass;   // major device class of the device
    byte      minorDeviceClass;   // minor device class of the device
    byte      clockOffset[2];     // clock offset of the device
}
```

The **BT\_DEVICE** structure defines the characteristics of a Bluetooth device.

### Members

*bdAddress*

Specifies address of the Bluetooth device.

*pageScanRepetitionMode*

Specifies page scan repetition mode of the Bluetooth device. This member can be one of the following values.

| Value   | Meaning                      |
|---------|------------------------------|
| PSRM_R0 | Page scan repetition mode R0 |
| PSRM_R1 | Page scan repetition mode R1 |
| PSRM_R2 | Page scan repetition mode R2 |

*pageScanPeriodMode*

Specifies page scan period mode of the Bluetooth device. This member can be one of the following values.

| Value   | Meaning |
|---------|---------|
| PSRM_P0 | P0      |
| PSRM_P1 | P1      |
| PSRM_P2 | P2      |

*pageScanRepetitionMode*

Specifies page scan mode of the Bluetooth device. This member can be one of the following values.

| Value            | Meaning                     |
|------------------|-----------------------------|
| PSM_MANDATORY    | Mandatory page scan mode    |
| PSM_OPTIONAL_I   | Optional page scan mode I   |
| PSM_OPTIONAL_II  | Optional page scan mode II  |
| PSM_OPTIONAL_III | Optional page scan mode III |

*serviceClass*

Specifies service class of the Bluetooth device. This member can be one of the following values.

| Value                           | Meaning                                 |
|---------------------------------|---|
| BT_SC_UNSPECIFIED               | Unspecified service class               |
| BT_SC_LIMITED_DISCOVERABLE_MODE | Limited discoverable mode service class |
| BT_SC_NETWORKING                | Networking service class                |
| BT_SC_RENDERING                 | Rendering service class                 |
| BT_SC_CAPTURING                 | Capturing service class                 |
| BT_SC_OBJECT_TRANSFER           | Object Transfer service class           |
| BT_SC_AUDIO                     | Audio service class                     |
| BT_SC_TELEPHONY                 | Telephony service class                 |
| BT_SC_INFORMATION               | Information service class               |

*majorDeviceClass*

Specifies the minor device class of the Bluetooth device. This member can be one of the following values depending on the major device class.

| Value               | Meaning                           |
|---------------------|-----------------------------------|
| BT_MAJOR_MISC       | Miscellaneous                     |
| BT_MAJOR_COMPUTER   | Computer (Desktop, Notebook, ...) |
| BT_MAJOR_PHONE      | Phone (Cellular, Codeless, ...)   |
| BT_MAJOR_LAP        | LAN/Network access point.         |
| BT_MAJOR_AUDIO      | Audio/video                       |
| BT_MAJOR_PERIPHERAL | Peripheral (Mouse, ...)           |

*minorDeviceClass*

Specifies the minor device class of the Bluetooth device. This member can be one of the following values depending on the major device class.

1. Major device class: BT\_MAJOR\_COMPUTER

| Value                 | Meaning               |
|-----------------------|-----------------------|
| BT_MINOR_UNCLASSIFIED | Uncategorized.        |
| BT_MINOR_DESKTOP      | Desktop workstation.  |
| BT_MINOR_SERVER       | Server-class computer |
| BT_MINOR_LAPTOP       | Laptop                |
| BT_MINOR_HANDHELD     | Handheld PC/PDA       |
| BT_MINOR_PALM         | Wearable computer     |

2. Major device class: BT\_MAJOR\_PHONE

| Value                 | Meaning        |
|-----------------------|----------------|
| BT_MINOR_UNCLASSIFIED | Uncategorized. |
| BT_MINOR_CELLULAR     | Cellular       |
| BT_MINOR_CORDLESS     | Cordless       |
| BT_MINOR_SMARTPHONE   | Smart phone    |
| BT_MINOR_WIREDMODEM   | Wired modem    |

3. Major device class: BT\_MAJOR\_LAP

| Value                    | Meaning                 |
|--------------------------|-------------------------|
| BT_MINOR_UNCLASSIFIED    | Uncategorized.          |
| BT_MINOR_FULLY_AVAILABLE | Service fully available |
| BT_MINOR_17              | Service 17% utilized    |
| BT_MINOR_33              | Service 33% utilized    |
| BT_MINOR_50              | Service 50% utilized    |
| BT_MINOR_67              | Service 67% utilized    |
| BT_MINOR_99              | Service 99% utilized    |
| BT_MINOR_NO_SERVICE      | Service not service     |

4. Major device class: BT\_MAJOR\_AUDIO

| Value                 | Meaning                                |
|-----------------------|--|
| BT_MINOR_UNCLASSIFIED | Uncategorized.                         |
| BT_MINOR_HS_PROFILE   | Device conforms to the Headset profile |

*clockOffset*

Specifies the clock offset of the Bluetooth device

## BT\_UUID

```
struct _BT_UUID
{
    byte    uuid[16]; // UUID value
    byte    length;   // number of bytes used to specify the UUID
}
```

The **BT\_UUID** structure defines the Universally Unique Identifier (UUID) used in SDAP transactions.

### Members

*uuid*

Specifies the value of the UUID

*length*

The number of bytes used to specify the UUID, length of 2,4 or 16 are valid.

#### 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;
BteDll.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 Error Codes for Bluetooth

| Error Code                             | Description  |
|--|--|
| BT_NO_ERROR                            | No error has occurred.<br>Value : 0x0000000  |
| BT_INVALID_PARAMETER                   | A parameter that was passed to the API was invalid.<br>Value : 0x30F0001               |
| BT_INSUFFICIENT_RESOURCES              | Insufficient system resources to complete the API.<br>Value : 0x30F0002                |
| BT_INVALID_HANDLE                      | The handle that was passed to the API was invalid.<br>Value : 0x30F0003                |
| BT_NOT_INITIALIZED                     | The bluetooth stack and hardware module has not been initialised.<br>Value : 0x30F0004 |
| BT_REQUEST_CANCELED                    | The request has been canceled.<br>Value : 0x30F0005                                    |
| BT_USER_DE_INITIALIZED                 | The user initiated deinitialisation.<br>Value : 0x10F0006                              |
| BT_NOT_OPENED                          | The bluetooth stack was not Opened for use.<br>Value : 0x30F0007                       |
| BT_TRIAL_PERIOD_EXPIRED                | Trial period expired.<br>Value : 0x30F0008   |
| BT_ALREADY_INITIALIZED                 | Stack was already initialized.<br>Value : 0x20F0009                                    |
| BT_VALID_CERTIFICATE_FILE_NOT_FOUND    | Either no certificate file or invalid certificate file.<br>Value : 0x20F0009           |
| BT_TRANSPORT_ALREADY_INITIALIZED       | The transport layer already initialized.<br>Value : 0x2010001                          |
| BT_TRANSPORT_NOT_INITIALIZED           | Transport layer not initialized.<br>Value : 0x3010007                                  |
| BT_TRANSPORT_H2_DEVICE_NOT_READY       | Attempt to retrieve device information failed.<br>Value : 0x3110001                    |
| BT_TRANSPORT_H2_COMPONENT_MISSING      | Component required for the usb transport missing.<br>Value : 0x3110002                 |
| BT_TRANSPORT_H2_READ_FAILURE           | Attempt to read from the device failed.<br>Value : 0x4110003                           |
| BT_TRANSPORT_H2_INVALID_PACKET_ERROR   | Invalid bluetooth packet.<br>Value : 0x4110004   |
| BT_TRANSPORT_H2_WRITE_FAILED           | Attempt to write to the device failed.<br>Value : 0x4110005                            |
| BT_TRANSPORT_H4_BAD_PORT               | Port with the specified name does not exist.<br>Value : 0x3100001                      |
| BT_TRANSPORT_H4_PORT_IN_USE            | Specified port is currently in use.<br>Value : 0x3100002                               |
| BT_TRANSPORT_H4_UNSUPPORTED_PARAMETERS | Specified parameters not supported by the transport layer.<br>Value : 0x3100003        |
| BT_TRANSPORT_H4_PARAMETERS_ERROR       | Invalid communication parameter (Port parameters).<br>Value : 0x3100005                |
| BT_TRANSPORT_H4_READ_FAILURE           | Attempt to read from the device failed.<br>Value : 0x4100006                           |
| BT_TRANSPORT_H4_INVALID_PACKET_ERROR   | Invalid bluetooth packet.<br>Value : 0x4100007   |
| BT_TRANSPORT_H4_WRITE_FAILED           | Attempt to write to the device failed.<br>Value : 0x4100008                            |
| BT_TRANSPORT_BCSP_BAD_PORT             | Port with the specified name does not exist.   |



|   |  |
|---|--|
|   | Value : 0x3130001  |
| <b>BT_TRANSPORT_BCSP_PORT_IN_USE</b>            | Specified port is currently in use.<br>Value : 0x3130002   |
| <b>BT_TRANSPORT_BCSP_UNSUPPORTED_PARAMETERS</b> | Specified parameters not supported by the transport layer.<br>Value : 0x3130003  |
| <b>BT_TRANSPORT_BCSP_LINK_SETUP_FAILED</b>      | BCSP link set up between host and host controller failed.<br>Value : 0x3130005   |
| <b>BT_TRANSPORT_BCSP_READ_FAILURE</b>           | Attempt to read from the device failed.<br>Value : 0x4130006   |
| <b>BT_TRANSPORT_BCSP_INVALID_PACKET_ERROR</b>   | Invalid bluetooth packet.<br>Value : 0x4130007   |
| <b>BT_TRANSPORT_BCSP_WRITE_FAILED</b>           | Attempt to write to the device failed.<br>Value : 0x4130008  |
| <b>BT_HCI_UNKNOWN_COMMAND</b>                   | Unknown hci command<br>Value : 0x3020001   |
| <b>BT_HCI_NO_EXISTING_CONNECTION</b>            | Host has issued a command which requires an existing connection and there is currently no connection corresponding to the specified device Address.<br>Value : 0x3020002 |
| <b>BT_HCI_HARDWARE_FAILURE</b>                  | Command cannot be executed because of a hardware failure.<br>Value : 0x3020003   |
| <b>BT_HCI_PAGE_TIMEOUT</b>                      | Page timeout occurred during connection attempt.<br>Value : 0x3020004  |
| <b>BT_HCI_AUTHENTICATION_FAILURE</b>            | Pairing or authentication failed due to incorrect PIN code or link key.<br>Value : 0x3020005   |
| <b>BT_HCI_KEY_MISSING</b>                       | PIN code(s) missing.<br>Value : 0x3020006  |
| <b>BT_HCI_MEMORY_FULL</b>                       | Host Controller does not have memory capacity to store additional parameters.<br>Value : 0x3020007   |
| <b>BT_HCI_CONNECTION_TIMEOUT</b>                | Connection timeout.<br>Value : 0x3020008   |
| <b>BT_HCI_MAX_NO_OF_CONNECTIONS</b>             | Maximum number of connections established.<br>Value : 0x3020009  |
| <b>BT_HCI_MAX_NO_OF_SCO_CONNECTIONS</b>         | Maximum number of SCO connections established.<br>Value : 0x302000A  |
| <b>BT_HCI_ACL_CONNECTION_ALREADY_EXISTS</b>     | An ACL connection to the device already exists.<br>Value : 0x302000B   |
| <b>BT_HCI_COMMAND_DISALLOWED</b>                | Command disallowed.<br>Value : 0x302000C   |
| <b>BT_HCI_HOST_REJECTED_LIMITED_RESOURCES</b>   | Host rejected the incoming connection request due to limited resources.<br>Value : 0x302000D   |
| <b>BT_HCI_HOST_REJECTED_SECURITY_REASONS</b>    | Host rejected the incoming connection request due to security reasons.<br>Value : 0x302000E  |
| <b>BT_HCI_HOST_REJECTED_PERSONAL_DEVICE</b>     | Host Rejected the incoming connection request due to remote device is only a personal device.<br>Value : 0x302000F   |
| <b>BT_HCI_HOST_TIMEOUT</b>                      | Host timeout occurred before responding to an incoming connection request.<br>Value : 0x3020010  |
| <b>BT_HCI_UNSUPPORTED_FEATURE_OR_PARAMETER</b>  | One or more specified parameter(s) not supported by the hardware.<br>Value : 0x3020011   |
| <b>BT_HCI_INVALID_COMMAND_PARAMETER</b>         | One or more specified parameter values not conform to the bluetooth specification.   |

|  |  |
|--|--|
|  | Value : 0x3020012  |
| <b>BT_HCI_OTHER_END_TERMINATED_USER_ENDED_CONNECTION</b> | Connection terminated by user at the other end.<br>Value : 0x3020013   |
| <b>BT_HCI_OTHER_END_TERMINATED_LOW_RESOURCES</b>         | Connection terminated by the other end due to low resources.<br>Value : 0x3020014  |
| <b>BT_HCI_OTHER_END_TERMINATED_POWER_OFF</b>             | Other End Terminated Connection: About to Power Off.<br>Value : 0x3020015  |
| <b>BT_HCI_CONNECTION_TERMINATED_LOCAL_HOST</b>           | Connection terminated by the local host.<br>Value : 0x3020016  |
| <b>BT_HCI_REPEATED_ATTEMPTS</b>                          | Device not allowed authentication\pairing because too little time has elapsed since an unsuccessful authentication\pairing attempt.<br>Value : 0x3020017 |
| <b>BT_HCI_PAIRING_NOT_ALLOWED</b>                        | Device not allowed pairing due to some reason.<br>Value : 0x3020018  |
| <b>BT_HCI_UNKNOWN_LMP_PDU</b>                            | Unknown LMP PDU.<br>Value : 0x3020019  |
| <b>BT_HCI_UNSUPPORTED_REMOTE_FEATURE</b>                 | Remote device does not support the feature associated with the issued command.<br>Value : 0x302001A  |
| <b>BT_HCI_SCO_OFFSET_REJECTED</b>                        | SCO offset rejected.<br>Value : 0x302001B  |
| <b>BT_HCI_SCO_INTERVAL_REJECTED</b>                      | SCO interval rejected.<br>Value : 0x302001C  |
| <b>BT_HCI_SCO_AIR_MODE_REJECTED</b>                      | SCO air mode rejected.<br>Value : 0x302001D  |
| <b>BT_HCI_INVALID_LMP_PARAMETERS</b>                     | Invalid LMP parameters.<br>Value : 0x302001E   |
| <b>BT_HCI_UNSPECIFIED_ERROR</b>                          | This error code is used when no other error code is appropriate.<br>Value : 0x302001F  |
| <b>BT_HCI_UNSUPPORTED_LMP_PARAMETER</b>                  | Unsupported LMP parameters.<br>Value : 0x3020020   |
| <b>BT_HCI_ROLE_CHANGE_NOT_ALLOWED</b>                    | Role change not allowed.<br>Value : 0x3020021  |
| <b>BT_HCI_LMP_RESPONSE_TIMEOUT</b>                       | LMP response timeout.<br>Value : 0x3020022   |
| <b>BT_HCI_LMP_ERROR_TRANSACTION_COLLISION</b>            | LMP error transaction collision<br>Value : 0x3020023   |
| <b>BT_HCI_LMP_PDU_NOT_ALLOWED</b>                        | LMP PDU not allowed.<br>Value : 0x3020024  |
| <b>BT_HCI_ENCRYPTION_MODE_NOT_ACCEPTABLE</b>             | Couldnt negotiate for which encryption mode to use.<br>Value : 0x3020025   |
| <b>BT_HCI_UNIT_KEY_USED</b>                              | Link key for the specified connection cannot be changed since it is a UNIT key.<br>Value : 0x3020026   |
| <b>BT_HCI_QOS_NOT_SUPPORTED</b>                          | Requested quality of service is not supported.<br>Value : 0x3020027  |
| <b>BT_HCI_INSTANT_PASSED</b>                             | The instant at which the specified operation shall be done is in the past.<br>Value : 0x3020028  |
| <b>BT_HCI_PAIRING_WITH_UNIT_KEY_NOT_SUPPORTED</b>        | Pairing with unit key not supported.<br>Value : 0x3020029  |
| <b>BT_HCI_NOT_INITIALIZED</b>                            | HCI not initialized.<br>Value : 0x3020054  |
| <b>BT_HCI_INQUIRY_IN_PROGRESS</b>                        | Device inquiry is in progress.<br>Value : 0x3020055  |
| <b>BT_HCI_PERIODIC_INQUIRY_IN_PROGRESS</b>               | Periodic device inquiry is in progress.<br>Value : 0x3020056   |
| <b>BT_HCI_WRITE_FAILED</b>                               | Attempt to write to the device failed.   |

|   |   |
|---|---|
|   | Value : 0x4020058   |
| <b>BT_HCI_COMMAND_NOT_ALLOWED</b>               | The specified command is not allowed.<br>Value : 0x3020059  |
| <b>BT_HCI_TIMEOUT</b>                           | Timeout occurred while waiting for the hci command response.<br>Value : 0x302005A                             |
| <b>BT_HCI_NO_ACL_BUFFER</b>                     | No ACL buffer free in host controller.<br>Value : 0x302005B   |
| <b>BT_HCI_NO_SCO_BUFFER</b>                     | No SCO buffer free in host controller.<br>Value : 0x302005C   |
| <b>BT_HCI_REMOVE_PENDING</b>                    | HCI connection object being removed.<br>Value : 0x302005D   |
| <b>BT_SCO_NOT_INITIALIZED</b>                   | SCO layer not initialized.<br>Value : 0x30E0001   |
| <b>BT_SCO_REMOVE_PENDING</b>                    | SCO list being removed.<br>Value : 0x30E0002  |
| <b>BT_SCO_NO_EXISTING_ACL_CONNECTION</b>        | No ACL connection exists with the remote device with which SCO connection is to be made.<br>Value : 0x30E0003 |
| <b>BT_SCO_NO_EXISTING_SCO_CONNECTION</b>        | No SCO connection exists with the remote device.<br>Value : 0x30E0004   |
| <b>BT_SCO_MAX_NO_OF_SCO_CONNECTIONS</b>         | Maximum number of SCO connections(three) already made.<br>Value : 0x30E0005                                   |
| <b>BT_SCO_TIMEOUT</b>                           | Request timed out.<br>Value : 0x30E0006   |
| <b>BT_L2CAP_TIMEOUT_CONNECT_REQ</b>             | The wait for connect response timed out.<br>Value : 0x3040001   |
| <b>BT_L2CAP_TIMEOUT_CONFIG_REQ</b>              | The wait for config response timed out.<br>Value : 0x3040002  |
| <b>BT_L2CAP_TIMEOUT_DISC_REQ</b>                | The wait for disconnect response timed out.<br>Value : 0x3040003  |
| <b>BT_L2CAP_TIMEOUT_ECHO_REQ</b>                | The wait for echo response timed out.<br>Value : 0x3040004  |
| <b>BT_L2CAP_TIMEOUT_INFO_REQ</b>                | The wait for info response timed out.<br>Value : 0x3040005  |
| <b>BT_L2CAP_TIMEOUT_CONNECT_CFM_FROM_HCI</b>    | The wait for ACL connect confirm from HCI timed out.<br>Value : 0x3040006                                     |
| <b>BT_L2CAP_TIMEOUT_DISCONNECT_RSP_FROM_HCI</b> | The wait for ACL disconnect response from HCI timed out.<br>Value : 0x3040007                                 |
| <b>BT_L2CAP_INVALID_STATE</b>                   | The current L2CAP state is invalid.<br>Value : 0x3040008  |
| <b>BT_L2CAP_ACL_CLOSE_PENDING</b>               | HCI connection being closed.<br>Value : 0x3040009   |
| <b>BT_L2CAP_ACL_CONNECTION_PENDING</b>          | Currently processing another hci_connectReq to the same device.<br>Value : 0x304000A                          |
| <b>BT_L2CAP_INVALID_GROUP</b>                   | No such L2CAP group present.<br>Value : 0x304000D   |
| <b>BT_L2CAP_INVALID_GROUP_MEMBER</b>            | Device not a member of the group.<br>Value : 0x304000E  |
| <b>BT_L2CAP_REMOVE_PENDING</b>                  | L2CAP connection object being removed.<br>Value : 0x3040010   |
| <b>BT_L2CAP_CONNECTION_ALREADY_EXIST</b>        | L2CAP connection already exists.<br>Value : 0x3040011   |
| <b>BT_L2CAP_UNKNOWN_OPTIONS</b>                 | Channel configuration failed.<br>Value : 0x3040014  |
| <b>BT_L2CAP_CONNECTION_REFUSED</b>              | L2CAP connection refused by remote machine.<br>Value : 0x3040016  |
| <b>BT_SDP_REQUEST_TIMEOUT</b>                   | Service search/service attribute/service search   |

|  |   |
|--|---|
|  | attribute requests timed out.<br>Value : 0x3060001  |
| <b>BT_SDP_UNSUPPORTED_SDP_VERSION</b>      | SDP version not supported.<br>Value : 0x3060004   |
| <b>BT_SDP_REMOVE_PENDING</b>               | SDP connection object being removed.<br>Value : 0x3060005   |
| <b>BT_SDP_INSUFFICIENT_RESOURCE</b>        | Insufficient Resources to satisfy Request.<br>Value : 0x3060006   |
| <b>BT_SDP_ATTRIBUTE_ALREADY_EXISTS</b>     | Occurs when user tries to add an existing attribute.<br>Value : 0x3060007                               |
| <b>BT_SDP_L2CAP_CONNECTION_FAILED</b>      | Failed to establish L2CAP connection.<br>Value : 0x3060008  |
| <b>BT_SDP_L2CAP_IN_CLOSING_STATE</b>       | Occurs when L2CAP is in closing state.<br>Value : 0x3060009   |
| <b>BT_SDP_SERVICE_ALREADY_EXISTS</b>       | Service already exists.<br>Value : 0x306000E  |
| <b>BT_SDP_RESULTSET_OVERFLOW</b>           | Occurs when we try to access the element after cursor position (cursor count).<br>Value : 0x3060014     |
| <b>BT_SDP_RESULTSET_UNDERFLOW</b>          | Occurs when we try to access the element before the starting position of the list.<br>Value : 0x3060015 |
| <b>BT_SDP_INVALID_CURSOR_POSITION</b>      | Occurs when the cursor position is not valid (ie before the start of the list).<br>Value : 0x3060016    |
| <b>BT_RFCOMM_TIMEOUT_TEST_REQ</b>          | Test request timed out.<br>Value : 0x3080001  |
| <b>BT_RFCOMM_TIMEOUT_SABM_REQ</b>          | SABM request timed out.<br>Value : 0x3080002  |
| <b>BT_RFCOMM_TIMEOUT_PN_REQ</b>            | PN request timed out.<br>Value : 0x3080003  |
| <b>BT_RFCOMM_TIMEOUT_MSC_REQ</b>           | MSC request timed out.<br>Value : 0x3080004   |
| <b>BT_RFCOMM_TIMEOUT_FCON_REQ</b>          | FCON request timed out.<br>Value : 0x3080005  |
| <b>BT_RFCOMM_TIMEOUT_FCOFF_REQ</b>         | FCOFF request timed out.<br>Value : 0x3080006   |
| <b>BT_RFCOMM_TIMEOUT_RPN_REQ</b>           | RPN request timed out.<br>Value : 0x3080007   |
| <b>BT_RFCOMM_TIMEOUT_RLS_REQ</b>           | RLS request timed out.<br>Value : 0x3080008   |
| <b>BT_RFCOMM_BAUDRATE_NOT_SUPPORTED</b>    | Baud rate not supported.<br>Value : 0x3080009   |
| <b>BT_RFCOMM_DATABITS_NOT_SUPPORTED</b>    | Data bit not supported.<br>Value : 0x308000A  |
| <b>BT_RFCOMM_STOPBIT_NOT_SUPPORTED</b>     | Stop bit not supported.<br>Value : 0x308000B  |
| <b>BT_RFCOMM_PARITYBIT_NOT_SUPPORTED</b>   | Parity bit not supported.<br>Value : 0x308000C  |
| <b>BT_RFCOMM_PARITYTYPE_NOT_SUPPORTED</b>  | Parity type not supported.<br>Value : 0x308000D   |
| <b>BT_RFCOMM_FLOWCONTROL_NOT_SUPPORTED</b> | Flowcontrol not supported.<br>Value : 0x308000E   |
| <b>BT_RFCOMM_XONCHAR_NOT_SUPPORTED</b>     | xon character not supported.<br>Value : 0x308000F   |
| <b>BT_RFCOMM_XOFFCHAR_NOT_SUPPORTED</b>    | xoff character not supported.<br>Value : 0x3080010  |
| <b>BT_RFCOMM_NO_UPPERLAYER_REGISTERED</b>  | No upper layer registered.<br>Value : 0x3080011   |
| <b>BT_RFCOMM_SESSION_NOT_CONNECTED</b>     | Session not in connected state.<br>Value : 0x3080012  |
| <b>BT_RFCOMM_CONNECTION_ALREADY_EXISTS</b> | The connection already exists to this channel.  |

|  |  |
|--|--|
|  | Value : 0x3080013  |
| <b>BT_RFCOMM_DLCI_NOT_CONNECTED</b>            | DLCI not in connected state.<br>Value : 0x3080014  |
| <b>BT_RFCOMM_TEST_DATA_MISMATCH</b>            | Received test data mismatch.<br>Value : 0x3080015  |
| <b>BT_RFCOMM_NO_SESSION</b>                    | Session does not exist.<br>Value : 0x3080016   |
| <b>BT_RFCOMM_FCS_MISMATCH</b>                  | FCS mismatch occurred.<br>Value : 0x3080017  |
| <b>BT_RFCOMM_MAX_NO_OF_SESSIONS</b>            | Already reached maximum number of allowable sessions.<br>Value : 0x3080018   |
| <b>BT_RFCOMM_REMOVE_PENDING</b>                | An object referenced is going to be removed.<br>Value : 0x3080019  |
| <b>BT_RFCOMM_SERVER_ALREADY_WAITING</b>        | A server is already waiting in the given server channel.<br>Value : 0x3080020                                      |
| <b>BT_SPP_PORT_ACCESS_DENIED</b>               | The specified port is already bounded with an SPP connection or in the process of binding.<br>Value : 0x3090001    |
| <b>BT_SPP_PORT_NOT_FOUND</b>                   | The specified port may not be installed in the system or its installation may not be proper.<br>Value : 0x3090002  |
| <b>BT_SPP_SERVICE_IN_USE</b>                   | The specified service cannot be deleted as it is in use.<br>Value : 0x3090003                                      |
| <b>BT_SECURITY_ACCESS_DENIED</b>               | Request rejected due to security failure.<br>Value : 0x30C0001   |
| <b>BT_SECURITY_THREAD_CREATION_FAILED</b>      | Creating thread failed.<br>Value : 0x30C0002   |
| <b>BT_SECURITY_REGISTRY_OPERATION_FAILED</b>   | Registry operation failed.<br>Value : 0x30C0003  |
| <b>BT_SECURITY_CALLBACK_ALREADY_REGISTERED</b> | Tried to set callback multiple times.<br>Value : 0x30C0004   |
| <b>BT_GOEP_CONNECTION_TERMINATED</b>           | The transport connection has been terminated.<br>Value : 0x3200001   |
| <b>BT_GOEP_REQUEST_TIMEDOUT</b>                | The request has been timed out.<br>Value : 0x3200002   |
| <b>BT_GOEP_OPERATION_IN_PROGRESS</b>           | An operation is already in progress.<br>Value : 0x3200003  |
| <b>BT_GOEP_AUTHENTICATION_FAILED</b>           | Server could not authenticate the client.<br>Value : 0x3200004   |
| <b>BT_GOEP_INVALID_RESPONSE</b>                | Response from the server is invalid.<br>Value : 0x3200005  |
| <b>BT_GOEP_UNAUTHORIZED_SERVER</b>             | Client could not authenticate the server.<br>Value : 0x3200006   |
| <b>BT_GOEP_OPERATION_ABORTED</b>               | The operation is aborted.<br>Value : 0x3200007   |
| <b>BT_GOEP_INVALID_HEADER_ID</b>               | The header is invalid.<br>Value : 0x3200008  |
| <b>BT_GOEP_INVALID_HANDLE</b>                  | Given handle is invalid.<br>Handle can be sessionhandle, operation handle or headersethandle.<br>Value : 0x3200009 |
| <b>BT_GOEP_SERVICE_NOT_AVAILABLE</b>           | The specified service is not available at the server.<br>Value : 0x3200010   |
| <b>BT_GOEP_ATTRIBUTE_NOT_FOUND</b>             | The specified attribute is not available in the SDDb of the server.<br>Value : 0x3200011                           |
| <b>BT_GOEP_WORKING_DIRECTORY_REQUIRED</b>      | The working directory should be set prior to the function call.<br>Value : 0x3200012                               |

|   |  |
|---|--|
| <b>BT_GOEP_FILE_NOT_FOUND</b>             | The specified file not found in the given directory.<br>Value : 0x3200013          |
| <b>BT_GOEP_INVALID_PARAMETER</b>          | A parameter that was passed to the API was invalid.<br>Value : 0x3200014           |
| <b>BT_GOEP_OBEXCONNECTION_FAILED</b>      | GOEP connection failed to establish.<br>Value : 0x3200015                          |
| <b>BT_GOEP_INVALID_TYPE</b>               | Object is of an invalid type.<br>Value : 0x3200016                                 |
| <b>BT_GOEP_OPERATION_DISALLOWED</b>       | Operation is disallowed.<br>Value : 0x3200017                                      |
| <b>BT_GOEP_NO_OPERATION_IN_PROGRESS</b>   | Push or pull operation is not in progress.<br>Value : 0x3200018                    |
| <b>BT_OPP_BUSINESSCARD_PUSH_FAILED</b>    | Pushing business card failed.<br>Value : 0x3210001                                 |
| <b>BT_OPP_BUSINESSCARD_PULL_FAILED</b>    | Pulling business card failed.<br>Value : 0x3210002                                 |
| <b>BT_DUN_ALREADY_CONNECTED</b>           | The DUN connection is already established.<br>Value : 0x3260001                    |
| <b>BT_LAP_ALREADY_CONNECTED</b>           | The LAP connection is already established.<br>Value : 0x3270001                    |
| <b>BT_HP_NOT_INITIALIZED</b>              | HP has not been initialized<br>Value : 0x3280001                                   |
| <b>BT_HP_MAX_NO_OF_SESSIONS</b>           | Number of possible sessions Exceeds Maximum Limit<br>Value : 0x3280002             |
| <b>BT_HIDP_ALREADY_INITIALIZED</b>        | The HIDP component is already initialized<br>Value : 0x3310001                     |
| <b>BT_HIDP_NOT_INITIALIZED</b>            | The HIDP component is not initialized at all<br>Value : 0x3310002                  |
| <b>BT_HIDP_REQUEST_PENDING</b>            | An HIDp request is already pending<br>Value : 0x3310003                            |
| <b>BT_HIDP_COMMAND_TIMEOUT</b>            | HIDP command timed out<br>Value : 0x3310004  |
| <b>BT_HIDP_DEVICE_NOT_READY</b>           | HID device is not ready for responding<br>Value : 0x3310005                        |
| <b>BT_HIDP_INVALID_REPORT_ID</b>          | An invalid report Id for the HID device<br>Value : 0x3310006                       |
| <b>BT_HIDP_UNSUPPORTED_REQUEST</b>        | The request is not supported by HID<br>Value : 0x3310007                           |
| <b>BT_HIDP_INVALID_PARAMETER</b>          | An invalid HIDP parameter<br>Value : 0x3310008                                     |
| <b>BT_HIDP_ERR_UNKNOWN</b>                | An HIDP unknown error occurred<br>Value : 0x3310009                                |
| <b>BT_HIDP_ERR_FATAL</b>                  | A fatal error in HIDP component<br>Value : 331000A                                 |
| <b>BT_HIDP_CONNECT_CANCELLED</b>          | HIDP connection process cancelled<br>Value : 331000B                               |
| <b>BT_AVDTP_L2CAP_REGISTRATION_FAILED</b> | Failed to register the L2cap Callbacks<br>Value : 0x3320001                        |
| <b>BT_AVDTP_STREAM_CONTEXT_EMPTY</b>      | The stream Context is Empty<br>Value : 0x3320002                                   |
| <b>BT_AVDTP_INVALID_SEID</b>              | The given SEID is invalid<br>Value : 0x3320003                                     |
| <b>BT_AVDTP_INVALID_STREAM_STATE</b>      | The present State of the Stream is Invalid<br>Value : 0x3320004                    |
| <b>BT_AVDTP_NO_CHANNEL</b>                | Code Text :No L2cap channel exists<br>Value : 0x3320005                            |
| <b>BT_AVDTP_REQUEST_TIMEDOUT</b>          | Code Text :AVDTP request has timed out<br>Value : 0x3320006                        |
| <b>BT_GAVDP_LOCAL_SEID_MISMATCH</b>       | A mismatch has occurred between the Seids maintained locally.<br>Value : 0x3330001 |



|   |   |
|---|---|
| <b>BT_GAVDP_MEDIA_CODEC_NOT_SUPPORTED</b>         | Media Codec is not supported<br>Value : 0x3330002   |
| <b>BT_GAVDP_CAPABILITY_REGISTRATION_VIOLATION</b> | An attempt to register more than one capability of a particular category<br>Value : 0x3330003 |
| <b>BT_GAVDP_ACCEPT_TIMEOUT</b>                    | GAVDP_Accept request timed out.<br>Value : 0x3330004  |
| <b>BT_GAVDP_SIGNALING_CHANNEL_ALREADY_EXISTS</b>  | Presently supporting only connection between 2 devices<br>Value : 0x3330005                   |
| <b>BT_GAVDP_REQUEST_TIMEOUT</b>                   | Code Text :GAVDP request has timed out<br>Value : 0x3330006                                   |
| <b>BT_A2DP_NOT_INITIALIZED</b>                    | A2D profile not initialized properly<br>Value : 0x3340001                                     |
| <b>BT_A2DP_SEPTYPE_NOT_SET</b>                    | A2DP -Local SEP Type not set<br>Value : 0x3340002   |
| <b>BT_A2DP_INVALID_SEPTYPE</b>                    | A2DP -Local SEP Type is invalid<br>Value : 0x3340003  |
| <b>BT_A2DP_REQUEST_TIMEOUT</b>                    | A2DP request has timed out<br>Value : 0x3340004   |

## 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:

M3P\_Camera.h

Required lib:

M3P\_Camera.lib

Required DLL:

M3P\_Camera.dll

#### For C#

Required DLL:

M3P\_Camera.dll  
M3p\_cam\_net.dll

### Supported Product

M3 T with camera option.



## 4.2.1 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

#### CAM\_OPTION

```
typedef struct
{
    INT    nInFormat;        // IN Stream Format
    INT    nResolution;      // Resolution
    INT    nPrevRGBDepth;    // IN Stream RGB Depth
    INT    nSaveFormat;      // SaveFormat
    INT    nJpegQuality;     // Jpeg Quality
    INT    nImgWidth;        // Image Width
    INT    nImgHeight;       // Image Height
    INT    nNamePrefix;      // Image File Name Prefix

    INT    nImgLux;          // Image Lux
    INT    nImgExposure;     // Image exposure
    INT    nImgEffect;       // Image Effect
    INT    nImgbalance;      // Image Balance

    INT    nImgRotatesave;   //Image Rotate
    INT    nImgRotateview;   //Image Rotate
} CAM_OPTION, *LPCAM_OPTION;
```

### Parameters

#### Image Lux

|                                |   |
|--------------------------------|---|
| #define OPTION_IMAGE_NOMAL_LUX | 0 |
| #define OPTION_IMAGE_NIGHT_LUX | 1 |

#### Image Exposure

|                                     |   |
|-------------------------------------|---|
| #define OPTION_IMAGE_EXPOSURE_ZERO  | 0 |
| #define OPTION_IMAGE_EXPOSURE_ONE   | 1 |
| #define OPTION_IMAGE_EXPOSURE_TWO   | 2 |
| #define OPTION_IMAGE_EXPOSURE_THREE | 3 |
| #define OPTION_IMAGE_EXPOSURE_FOUR  | 4 |
| #define OPTION_IMAGE_EXPOSURE_FIVE  | 5 |

#### Image Effect

|  |   |
|--|---|
| #define OPTION_IMAGE_NOMAL_EFFECT      | 0 |
| #define OPTION_IMAGE_SEPIA_EFFECT      | 1 |
| #define OPTION_IMAGE_BLACKWHITE_EFFECT | 2 |
| #define OPTION_IMAGE_NEGATIVE_EFFECT   | 3 |
| #define OPTION_IMAGE_UVRED_EFFECT      | 4 |
| #define OPTION_IMAGE_UVBLUE_EFFECT     | 5 |

|                                     |   |
|-------------------------------------|---|
| #define OPTION_IMAGE_UVGREEN_EFFECT | 6 |
|-------------------------------------|---|

| Image Balance                             |   |
|---|---|
| #define OPTION_IMAGE_BALANCE_AUTO         | 0 |
| #define OPTION_IMAGE_BALANCE_SUNNY        | 1 |
| #define OPTION_IMAGE_BALANCE_DAY          | 2 |
| #define OPTION_IMAGE_BALANCE_CWF          | 3 |
| #define OPTION_IMGAE_BALANCE_INCANDESENCE | 4 |

| Image Resolution                    |   |
|-------------------------------------|---|
| #define OPTION_RESOLUTION_1600X1200 | 4 |
| #define OPTION_RESOLUTION_1280X1024 | 0 |
| #define OPTION_RESOLUTION_640X480   | 1 |
| #define OPTION_RESOLUTION_320X240   | 2 |

| Preview RGB Depth                   |   |
|-------------------------------------|---|
| #define OPTION_PREV_RGB_DEPTH_16BIT | 0 |
| #define OPTION_PREV_RGB_DEPTH_24BIT | 1 |

| Save Format                    |   |
|--------------------------------|---|
| #define OPTION_SAVE_FORMAT_BMP | 0 |
| #define OPTION_SAVE_FORMAT_JPG | 1 |

| Image File Name Prefix                    |   |
|---|---|
| #define OPTION_IMAGE_FILENAME_PREF_DATE   | 0 |
| #define OPTION_IMAGE_FILENAME_PREF_SERIAL | 1 |

| Image Rotate             |   |
|--------------------------|---|
| #define IMAGE_ROTATE_0   | 0 |
| #define IMAGE_ROTATE_90  | 1 |
| #define IMAGE_ROTATE_180 | 2 |
| #define IMAGE_ROTATE_270 | 3 |

#### 4.2.1.1 MC3P\_Init

This function performs initialization.

---

```
BOOL MC3P_Init (IntPtr hWnd, IntPtr p_hWnd);
```

---

##### Parameters

*hWnd*

Main window handle

*p\_hWnd*

Windows handle to show image.

##### Return Values

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

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

##### Remarks

MC3P\_Init is a basic initialization step to run Camera. Please register Window Handle focused on running camera at the first parameter. At second parameter, register Window Handle that will output image. If initialization is succeeded, preview screen will be seen on registered control.

---

C++

```
CStatic m_ctrlpreview;  
CM3P_Camera m_m3p_camera; // CM3P_Camera is the camera dll class.  
  
void CameraInit()  
{  
    // m_ctrlpreview is a static tool on camera dlg  
    m_m3p_camera.MC3P_Init(m_hWnd,m_ctrlpreview.m_hWnd)  
    //return device type  0: 6300, 1: 6400, 2:6500  
  
    if(!m_m3p_camera.MC3P_Open())  
    {  
        AfxMessageBox(L"Com Open Error");  
        return FALSE;  
    }  
}
```

#### 4.2.1.2 MC3P\_Open

This function opens a COM port to a camera module.

---

```
BOOL MC3P_Open ();
```

---

##### 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

MC3P\_Opne function opens a camera port and initializes module at the same time.

---

C++

```
CStatic m_ctrlpreview;  
CM3P_Camera m_m3p_camera; // CM3P_Camera is the camera dll class.  
  
void CameraInit()  
{  
    // m_ctrlpreview is a static tool on camera dlg
```

```

m_m3p_camera.MC3P_Init(m_hWnd,m_ctrlpreview.m_hWnd))
//return device type  0: 6300, 1: 6400, 2:6500

if(!m_m3p_camera.MC3P_Open())
{
    AfxMessageBox(L"Com Open Error");
    return FALSE;
}
}

```

#### 4.2.1.3 MC3P\_Close

This function closes a COM port to a camera module.

---

```

BOOL MC3P_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

MC3P\_Close function is called when the user which to close the camera.

---

C++

```

void CloseCam()
{
    MC3P_Close();
}

```

---

#### 4.2.1.4 MC3P\_PreviewStart

This function starts preview.

---

```

BOOL MC3P_PreviewStart( );

```

---

##### 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

MC3P\_PreviewStart function output the image from camera module through Window Handle registered in MC3P\_Init function.

---

C++

```

void PreviewStart(BOOL bOn)
{
    if(bOn == TRUE)
    {
        m_m3p_camera.MC3P_PreViewStart();
    }
    else
    {
        m_m3p_camera.MC3P_PreViewStop();
        m_ctrlpreview.Invalidate();
    }
}

```

---

```
}  
}
```

#### 4.2.1.5 MC3P\_PreviewStop

This function stops preview.

---

```
BOOL MC3P_PreviewStop();
```

---

##### 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

MC3P\_PreviewStop stops preview image.

---

C++

```
void PreviewStart(BOOL bOn)  
{  
    if(bOn == TRUE)  
    {  
        m_m3p_camera.MC3P_PreViewStart();  
    }  
    else  
    {  
        m_m3p_camera.MC3P_PreViewStop();  
        m_ctrpreview.Invalidate();  
    }  
}
```

#### 4.2.1.6 MC3P\_Capture

MC3P\_Capture outputs the preview image as JPG or BMP.

---

```
BOOL MC3P_Capture();
```

---

##### 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

The image captured is affected by the saved options.

---

C++

```
void CaptureImage()  
{  
    m_m3p_camera.MC3P_Capture();  
}
```

#### 4.2.1.7 MC3P\_ReceiveEvent

This function Receives event.

---

```
BOOL MC3P_ReceiveEvent();
```

---

### Parameters

None

### Return Values

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

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

#### 4.2.1.8 MC3P\_Set\_CAMERA\_OPTION

This function sets option values.

---

```
BOOL MC3P_Set_CAMERA_OPTION(ref CAM_OPTION option, ref string savefolder);
```

---

### Parameters

*option*

Option value

*savefolder*

Pointer to buffer to receive save folder name

### Return Values

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

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

### Remarks

LPCAM\_OPTION is a structure that contains option values of camera. Once the configurations of the structure are saved, it is used as the input parameter of this function. The captured or preview image is affected by the saved options.

---

C++

```
void SaveCamOption()
{
    INT    nIdx;
    UpdateData();

    //Image Format
    nIdx = m_cbImgFormat.GetCurSel();
    if( nIdx == 0)
        m_pCamOption->nInFormat =OPTION_XLLP_CAMERA_IMAGE_FORMAT_YCBCR422_PACKED;
    else if ( nIdx == 1)
        m_pCamOption->nInFormat = OPTION_XLLP_CAMERA_IMAGE_FORMAT_RGB565;

    //Resolution
    nIdx = m_cbResolution.GetCurSel();
    if (nIdx == 0)
    {
        m_pCamOption->nResolution = OPTION_RESOLUTION_1600X1200;
        m_pCamOption->nImgWidth = 1600;
        m_pCamOption->nImgHeight = 1200;
    }
    else if (nIdx == 1)
    {
        m_pCamOption->nResolution = OPTION_RESOLUTION_1280X1024;
        m_pCamOption->nImgWidth = 1280;
        m_pCamOption->nImgHeight = 1024;
    }
    else if (nIdx == 2)
    {
        m_pCamOption->nResolution = OPTION_RESOLUTION_640X480;
        m_pCamOption->nImgWidth = 640;
```

```

m_pCamOption->nImgHeight = 480;
}
else if (nIdx == 3)
{
m_pCamOption->nResolution = OPTION_RESOLUTION_320X240;
m_pCamOption->nImgWidth = 320;
m_pCamOption->nImgHeight = 240;
}

//Display RGB Depth
if (m_b16Bit == TRUE)
    m_pCamOption->nPrevRGBDepth = OPTION_PREV_RGB_DEPTH_16BIT;
else if (m_b24Bit == TRUE)
    m_pCamOption->nPrevRGBDepth = OPTION_PREV_RGB_DEPTH_24BIT;

//Save Format
nIdx = m_cbSaveFormat.GetCurSel();
if (nIdx == 0) m_pCamOption->nSaveFormat = OPTION_SAVE_FORMAT_BMP;
else if (nIdx == 1) m_pCamOption->nSaveFormat = OPTION_SAVE_FORMAT_JPG;

//Jpeg Quality
m_pCamOption->nJpegQuality = m_edJpegQuality;

//Image Effect
nIdx = m_cbEffect.GetCurSel();
if(nIdx == 0) m_pCamOption->nImgEffect = OPTION_IMAGE_NOMAL_EFFECT;
else if(nIdx == 1) m_pCamOption->nImgEffect = OPTION_IMAGE_SEPIA_EFFECT;
else if(nIdx == 2)
    m_pCamOption->nImgEffect = OPTION_IMAGE_BLACKWHITE_EFFECT;
else if(nIdx == 3) m_pCamOption->nImgEffect = OPTION_IMAGE_NEGATIVE_EFFECT;
else if(nIdx == 4) m_pCamOption->nImgEffect = OPTION_IMAGE_UVRED_EFFECT;
else if(nIdx == 5) m_pCamOption->nImgEffect = OPTION_IMAGE_UVBLUE_EFFECT;
else if(nIdx == 6)
    m_pCamOption->nImgEffect = OPTION_IMAGE_UVGREEN_EFFECT;

nIdx = m_cbPrefix.GetCurSel();
if(nIdx == 0) m_pCamOption->nNamePrefix = OPTION_IMAGE_FILENAME_PREF_DATE;
else if(nIdx == 1)
    m_pCamOption->nNamePrefix = OPTION_IMAGE_FILENAME_PREF_SERIAL;

nIdx = m_cbRotatesave.GetCurSel();
if(nIdx == 0) { m_pCamOption->nImgRotatesave = IMAGE_ROTATE_0; }
else if(nIdx == 1){ m_pCamOption->nImgRotatesave = IMAGE_ROTATE_90; }
else if(nIdx == 2){ m_pCamOption->nImgRotatesave = IMAGE_ROTATE_180; }
else if(nIdx == 3){ m_pCamOption->nImgRotatesave = IMAGE_ROTATE_270; }

nIdx = m_cbRotateview.GetCurSel();
if(nIdx == 0) { m_pCamOption->nImgRotateview = IMAGE_ROTATE_0; }
else if(nIdx == 1){ m_pCamOption->nImgRotateview = IMAGE_ROTATE_90; }
else if(nIdx == 2){ m_pCamOption->nImgRotateview = IMAGE_ROTATE_180; }
else if(nIdx == 3){ m_pCamOption->nImgRotateview = IMAGE_ROTATE_270; }
m_m3p_camera.MC3P_Set_CAMERA_OPTION(m_pCamOption,
m_edSaveFolder.GetBuffer(0));
}

```

#### 4.2.1.9 MC3P\_Get\_CAMERA\_OPTION

This function sets option value.

---

```

BOOL int MC3P_Get_CAMERA_OPTION(LPCAM_OPTION option, TCHAR *savefolder);

```

---

#### Parameters

*option*

Option value [out]

*\*savefolder*

Pointer to buffer to receive save folder name

### Return Values

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

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

### Remarks

LPCAM\_OPTION is a structure that contains option values of camera. Option values set up in current module can be brought through this structure.

---

C++

```
void GetOption()  
{  
    m_m3p_camera.MC3P_Get_CAMERA_OPTION(m_pCamOption,  
m_edSaveFolder.GetBuffer(0))  
}
```

#### 4.2.1.10 MC3P\_FlashOFF

This function turns off the flash.

---

```
int MC3P_FlashOFF();
```

---

### 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

When a flash is required to take a better quality image, this function can be used especially in dark environment.

---

C++

```
void CameraFlashOn(BOOL bOn)  
{  
    if(bOn == TRUE)  
    {  
        m_m3p_camera.MC3P_FlashON();  
    }  
    else  
    {  
        m_m3p_camera.MC3P_FlashOFF();  
    }  
}
```

#### 4.2.1.11 MC3P\_FlashON

This function turns on the flash.

---

```
int MC3P_FlashON();
```

---

### 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

When a flash is required to take a better quality image, this function can be used especially in dark environment.

---

```
C++
void CameraFlashOn(BOOL bOn)
{
    if(bOn == TRUE)
    {
        m_m3p_camera.MC3P_FlashON();
    }
    else
    {
        m_m3p_camera.MC3P_FlashOFF();
    }
}
```

---

#### 4.2.1.12 MC3P\_Bright

This function sets up the brightness of camera lense.

---

```
void MC3P_Bright(int nbright);
```

---

#### Parameters

*nbright*

Value ranges from 0 to 5

#### Return Values

None

#### Remarks

MC3P\_Bright adjusts the brightness of camera lense. It can only be applied to 2M camera.

---

```
C++
int m_nBright = 0;

void OnClickBrightUp()
{
    MC3P_Bright(++m_nBright);
}
```

---

#### 4.2.1.13 MC3P\_RowData

This function gets the preview raw image data.

---

```
void MC3P_RowData(byte* data);
```

---

#### Parameters

*data*

Raw data

#### Return Values

None

#### Remarks

MC3P\_RowData gets the raw data seen by preview screen.

---

C++

```
void CM3P_CameraTestDlg::OnButRowdata()
{
    byte    *m_data;
    DWORD   dsize;
    RECT rect;
    HDC  hPreviewDC = NULL;
    HBITMAP  hBitmap = NULL;
    HBITMAP  hOldBitmap = NULL;
    HDC  hMemoryDC = NULL;

    dsize = m_m3p_camera.MC3P_GetBuffSize();
    m_data = new byte[dsize];
    memset(m_data,0,dsize);
    m_m3p_camera.MC3P_RowData(m_data);

    hPreviewDC = ::GetDC(m_ctrltestview.m_hWnd);
    hMemoryDC = CreateCompatibleDC(hPreviewDC);
    ::GetClientRect(m_ctrltestview.m_hWnd, &rect);
    hBitmap = CreateBitmap ( 320,240, 1, 24, (char*)m_data);
    hOldBitmap = (HBITMAP)SelectObject (hMemoryDC, hBitmap);

    StretchBlt(hPreviewDC, 0, 0, rect.right-rect.left, rect.bottom-rect.top,
hMemoryDC, 0, 0, 320, 240,  SRCCOPY);

    SelectObject(hMemoryDC,hOldBitmap);
    DeleteObject(hBitmap);
    DeleteDC (hMemoryDC);
    ::ReleaseDC (m_ctrltestview.m_hWnd, hPreviewDC);
    delete [] m_data;
}
```

---

#### 4.2.1.14 MC3P\_GetBuffSize

This function gets size of the preview raw image data.

---

```
DWORD MC3P_GetBuffSize();
```

---

#### Parameters

None

#### Return Values

Size of the preview raw image data.

---

C++

```
void CM3P_CameraTestDlg::OnButRowdata()
{
    byte    *m_data;
    DWORD   dsize;
    RECT rect;
    HDC  hPreviewDC = NULL;
    HBITMAP  hBitmap = NULL;
    HBITMAP  hOldBitmap = NULL;
    HDC  hMemoryDC = NULL;

    dsize = m_m3p_camera.MC3P_GetBuffSize();
    m_data = new byte[dsize];
    memset(m_data,0,dsize);
    m_m3p_camera.MC3P_RowData(m_data);

    hPreviewDC = ::GetDC(m_ctrltestview.m_hWnd);
    hMemoryDC = CreateCompatibleDC(hPreviewDC);
```

---

```
        ::GetClientRect(m_ctrltestview.m_hWnd, &rect);
        hBitmap = CreateBitmap ( 320,240, 1, 24, (char*)m_data);
        hOldBitmap = (HBITMAP)SelectObject (hMemoryDC, hBitmap);

        StretchBlt(hPreviewDC, 0, 0, rect.right-rect.left, rect.bottom-rect.top,
hMemoryDC, 0, 0, 320, 240, SRCCOPY);

        SelectObject(hMemoryDC,hOldBitmap);
        DeleteObject(hBitmap);
        DeleteDC (hMemoryDC);
        ::ReleaseDC (m_ctrltestview.m_hWnd, hPreviewDC);
        delete [] m_data;
    }
```

## 4.2.2 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

#### CAM\_OPTION

```
public struct CAM_OPTION
{
    public int nImgbalance;
    public int nImgEffect;
    public int nImgExposure;
    public int nImgHeight;
    public int nImgLux;
    public int nImgRotatesave;
    public int nImgRotateview;
    public int nImgWidth;
    public int nInFormat;
    public int nJpegQuality;
    public int nNamePrefix;
    public int nPrevRGBDepth;
    public int nResolution;
    public int nSaveFormat;
}
```

#### 4.2.2.1 MC3P\_Init

This function performs initialization.

---

```
int MC3P_Init (HWND hWnd, HWND p_hWnd);
```

---

##### Parameters

*hWnd*

Main window handle.

*p\_hWnd*

Windows handle to show image.

##### Return Values

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

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

##### Remarks

MC3P\_Init is a basic initialization step to run Camera. Please register Window Handle focused on running camera at the first parameter. At second parameter, register Window Handle that will output image. If initialization is succeeded, preview screen will be seen on registered control.

---

C#

```
using M3p_cam_net;

private M3p_cam_net.CamCore camctrl;

public CameraInit()
{
    camctrl = new CamCore();
    camctrl.MC3P_Init(this.Handle, pictureBox1.Handle);

    if(!camctrl.MC3P_Open())
    {
        MessageBox.Show("open error");
    }
}
```

---

#### 4.2.2.2 MC3P\_Open

This function opens a COM port to a camera module.

---

```
int MC3P_Open ();
```

---

##### 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

MC3P\_Open function opens a camera port and initializes module at the same time.

---

C#

```
using M3p_cam_net;

private M3p_cam_net.CamCore camctrl;

public CameraInit()
{
    camctrl = new CamCore();
```

---

```

camctrl.MC3P_Init(this.Handle, pictureBox1.Handle);

if(!camctrl.MC3P_Open())
{
    MessageBox.Show("open error");
}
}

```

#### 4.2.2.3 MC3P\_Close

This function closes a COM port to a camera module.

---

```
int MC3P_Open ();
```

---

##### 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

MC3P\_Close function is called when the user which to close the camera.

---

C#

```

void CameraClose()
{
    camctrl.MC3P_Close();
}

```

#### 4.2.2.4 MC3P\_PreviewStart

This function starts preview.

---

```
int MC3P_PreviewStart();
```

---

##### 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

MC3P\_PreviewStart function output the image from camera module through Window Handle registered in MC3P\_Init function.

---

C#

```

void PreviewStart(BOOL bOn)
{
    if(bOn == TRUE)
    {
        camctrl.MC3P_PreViewStart();
    }
    else
    {
        camctrl.MC3P_PreViewStop();
        m_ctrpreview.Invalidate();
    }
}

```

#### 4.2.2.5 MC3P\_PreviewStop

This function stops preview.

---

```
int MC3P_PreviewStop();
```

---

##### 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

MC3P\_PreviewStop stops preview image.

---

C#

```
void PreviewStart(BOOL bOn)
{
    if(bOn == TRUE)
    {
        camctrl.MC3P_PreviewStart();
    }
    else
    {
        camctrl.MC3P_PreviewStop();
        m_ctrpreview.Invalidate();
    }
}
```

---

#### 4.2.2.6 MC3P\_Capture

MC3P\_Capture outputs the preview image as JPG or BMP.

---

```
int MC3P_Capture();
```

---

##### 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

The image captured is affected by the saved options.

---

C#

```
void CaptureImage()
{
    camctrl.MC3P_Capture();
}
```

---

#### 4.2.2.7 MC3P\_ReceiveEvent

This function Receives event.

---

```
int MC3P_ReceiveEvent();
```

---

##### Parameters

None

### Return Values

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

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

#### 4.2.2.8 MC3P\_Set\_CAMERA\_OPTION

This function sets option values.

---

```
int MC3P_Set_CAMERA_OPTION(LPCAM_OPTION option, TCHAR *savefolder);
```

---

### Parameters

*option*

Option value

*savefolder*

Pointer to buffer to receive save folder name

### Return Values

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

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

### Remarks

LPCAM\_OPTION is a structure that contains option values of camera. Once the configurations of the structure are saved, it is used as the input parameter of this function. The captured or preview image is affected by the saved options.

---

C#

```
void SetOptionClick()
{
    if (comboBox_Format.SelectedIndex == 1)
        m_camoption.nInFormat = 5;
    else
        m_camoption.nInFormat = 15;
    m_camoption.nResolution = comboBox_Resolution.SelectedIndex;
    if (true == m_b2MCAm)
    {
        if (m_camoption.nResolution == 0)
        {
            m_camoption.nImgWidth = 1600;
            m_camoption.nImgHeight = 1200;
        }
        if (m_camoption.nResolution == 1)
        {
            m_camoption.nImgWidth = 1280;
            m_camoption.nImgHeight = 1024;
        }
        else if (m_camoption.nResolution == 2)
        {
            m_camoption.nImgWidth = 640;
            m_camoption.nImgHeight = 480;
        }
        else if (m_camoption.nResolution == 3)
        {
            m_camoption.nImgWidth = 320;
            m_camoption.nImgHeight = 240;
        }
    }
    else
    {
        if (m_camoption.nResolution == 0)
```

---



```

    {
        m_camoption.nImgWidth = 1280;
        m_camoption.nImgHeight = 1024;
    }
    if (m_camoption.nResolution == 1)
    {
        m_camoption.nImgWidth = 640;
        m_camoption.nImgHeight = 480;
    }
    else if (m_camoption.nResolution == 2)
    {
        m_camoption.nImgWidth = 320;
        m_camoption.nImgHeight = 240;
    }
}
m_camoption.nSaveFormat = comboBox_saveformat.SelectedIndex;
m_camoption.nNamePrefix = comboBox_picturename.SelectedIndex;
m_camoption.nJpegQuality = (int)numericUpDown_jpec.Value;
m_camoption.nImgRotatesave = 1;
m_camoption.nImgRotateview = 1;
m_szfoldername = textBox_savefolder.Text;
m_pcmcore.MC3P_Set_CAMERA_OPTION(ref m_camoption, ref m_szfoldername);
}

```

#### 4.2.2.9 MC3P\_Get\_CAMERA\_OPTION

This function sets option value.

---

```
int MC3P_Get_CAMERA_OPTION(LPCAM_OPTION option, TCHAR *savefolder);
```

---

##### Parameters

*option*

Option value [out]

*\*savefolder*

Pointer to buffer to receive save folder name

##### Return Values

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

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

##### Remarks

LPCAM\_OPTION is a structure that contains option values of camera. Option values set up in current module can be brought through this structure.

---

C#

```

void GetOption()
{
    CAM_OPTION pcm_option;
    string szfolder;

    camctrl.MC3P_Get_CAMERA_OPTION(out pcm_option,out szfolder);
}

```

---

#### 4.2.2.10 MC3P\_FlashOFF

This function turns off the flash.

---

```
int MC3P_FlashOFF();
```

---

##### 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

When a flash is required to take a better quality image, this function can be used especially in dark environment.

---

C#

```
void CameraFlashOn(BOOL bOn)
{
    if(bOn == TRUE)
    {
        m_m3p_camera.MC3P_FlashON();
    }
    else
    {
        m_m3p_camera.MC3P_FlashOFF();
    }
}
```

---

#### 4.2.2.11 MC3P\_FlashON

This function turns on the flash.

---

int MC3P\_FlashON();

---

#### 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

When a flash is required to take a better quality image, this function can be used especially in dark environment.

---

C#

```
void CameraFlashOn(BOOL bOn)
{
    if(bOn == TRUE){
        camctrl.MC3P_FlashON();
    }
    else{
        camctrl.MC3P_FlashOFF();
    }
}
```

---

#### 4.2.2.12 MC3P\_Bright

This function sets up the brightness of camera lense.

---

void MC3P\_Bright(int nbright);

---

#### Parameters

*nbright*

Value ranges from 0 to 5.

**Return Values**

None

**Remarks**

MC3P\_Bright adjusts the brightness of camera lense. It can only be applied to 2M camera.

---

C++

```
int m_nBright = 0;

void OnClickBrightUp()
{
    camctrl.MC3P_Bright(++m_nBright);
}
```

---

**4.2.1.13 MC3P\_RowData**

This function gets the preview raw image data.

---

```
void MC3P_RowData(byte* data);
```

---

**Parameters**

*data*

Raw data

**Return Values**

None

**Remarks**

MC3P\_RowData gets the raw data seen by preview screen.

**4.2.2.14 MC3P\_GetBuffSize**

This function gets size of the preview raw image data.

---

```
DWORD MC3P_GetBuffSize();
```

---

**Parameters**

None

**Return Values**

Size of the preview raw image data.

## 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 T MC-6700S

### 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 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 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 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);  
}
```

---



### 4.3.2 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 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.

*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#

```
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 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 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 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:

```
MCSSLib.dll  
MCSSLibNet.dll
```

### Supported Product

M3 T with 1D scanner that uses software decoder.

#### 4.4.1 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;

#ifdef 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;
ABLE           XmitAIMID;    //2009.09.30

} KSCANREADEX2;
typedef KSCANREADEX2* PKSCANREADEX2;

```

## **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_INDUSTRIY = 0x08,    // 8,
    CODE25KIND_IATA = 0x10,         // 16,
    CODE25KIND_ITF14 = 0x20,        // 32,
}CODE25_KIND;

```



```
typedef enum {  
    DIGIT0=0,  
    DIGIT1,  
    DIGIT2  
}CHECK_DIGIT;
```

#### 4.4.1.1 CKScan::Open

The Open function opens a COM port to a Scan Module, and performs initialization.

---

```
BOOL Open(  
    int CommNumber = 0,  
    BOOL CommDetect = FALSE,  
    DWORD BaudRate = CBR_115200,  
    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: *Number*, 1...16. The default is FALSE.

###### *BaudRate*

Specifies baud rate for Com port. The default is CBR\_115200.

###### *BaudDetect*

Specifies automatic scanning of baud rate when the Scan Module was not found using *BaudRate* specified. The order of scanning is: *BaudRate*, CBR\_115200, CBR\_38400, CBR\_57600. The default is FALSE.

###### *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 CKScan::Close 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 = m_KScan.Open(6, FALSE, CBR_115200, FALSE, NULL);
```

---

```
If(bRet == FALSE)
    ::MessageBox(NULL, L"Error : Scanner Open Failed", NULL, MB_TOPMOST);
```

#### 4.4.1.2 CKScan::Close

The **Close** function closes the COM port to a Scan Module, and may put the Scan Module into sleep mode.

```
BOOL Close();
```

##### 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 CKScan::Open for proper operation and resource management.

It is advised to fully utilize sleep mode capability of the Scan Module. When battery preservation is critical, users may want to implement CKScan::Sleep and CKScan::Wakeup for every operation.

C++

```
BOOL bRet = m_KScan.Close();
If(bRet == FALSE)
    ::MessageBox(NULL, L"Error : Scanner Close Failed", NULL, MB_TOPMOST);
```

#### 4.4.1.3 CKScan::Read

The **Read** function reads a barcode from the Scan Module. This function call can be either blocking or non-blocking depending on parameters in the KSCANREAD structure.

```
BOOL Read(
    PKSCANREAD    pRead
);
```

##### Parameters

*pRead*

Specifies pointer to the KSCANREAD Structure.

##### 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 CKScan::Open.

Reading a barcode is delicate and complicated process. It is highly recommended that users read Reading a barcode with proper options section for proper understanding of these options and flags.

Please refer to included sample program for details of blocking and 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.

If the scan engine is in sleep mode, it is necessary to issue CKScan::Wakeup first, otherwise the function call will result in fail.

C++

```
void CMainSheet::M3_ScanRead( )
```

```

{
    CString      Str;
    BOOL         bRet;
    m_bReading = TRUE;

    if (m_bReading)
    {
        // Reading already in progress, now cancel it.
        bRet = m_KScan.ReadCancel();

        if (bRet)
        {
            m_bReading = FALSE;
        }
        else
        {
            Str = _T("Error ReadCancel\r\n");
            Str += KScanGetLastErrorMsg();
        }
    }

    m_bReading = TRUE;
    bRet = m_KScan.Read(&kRead);
    if (!bRet)
    {
        Str.Format(_T("Error Read \r\n%d\r\n%s"), kRead.out_Status,
        KScanGetLastErrorMsg());
        m_bReading = FALSE;
    }
}

```

#### 4.4.1.4 CKScan::ReadCancel

The **ReadCancel** function cancels CKScan::ReadForever or CKScan::Read that was initiated as non-blocking operation. Its use is strongly discouraged.

---

```

BOOL ReadCancel ()

```

---

##### 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 CKScan::Read. 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 take up to 1 second to complete its tasks.

## See Also

CKScan::Read, KSCANREAD Structure

---

C++

---

```
BOOL bRet = m_KScan.ReadCancel();
```

---

### 4.4.1.5 CKScan::ReadForever

The **ReadForever** function is a specialized form of the CKScan::Read function. The limitations are:

- It must always be in non-blocking mode.
- It only looks for 1D barcodes.
- It does not terminate itself whether a barcode is found or not.

The function terminates when the call back function returns 0 (see Callback Function for Non-blocking Read).

---

```
BOOL ReadForever(  
    PKSCANREAD    pRead  
) ;
```

---

#### Parameters

*pRead*

Specifies pointer to the KSCANREAD Structure. The parameter nTimeInSeconds is ignored.

#### Return Values

If the function succeeds, the return value is TRUE.

If the function fails or times out, the return value is FALSE.

#### Remarks

All the remarks for CKScan::Read are applicable here.

**ReadForever** is an extremely powerful function, albeit a very fragile process that provides applications with even more control of how the scan module reads barcodes. 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 CKScan::ReadCancel function should be avoided to terminate the read. Using the CKScan::ReadCancel function multiple times to terminate the read operation may lead to unstable application.

### 4.4.1.6 CKScan::GetCommName

The **GetCommName** function returns a pointer to the name of current COM port with CKScan::Open.

---

```
LPCTSTR GetCommName();
```

---

#### Parameters

None

#### Return Values

The function returns a pointer to the name of current COM port such as "COM1:". If there is no COM port open, the return string is "None".

#### Remarks

None

### 4.4.1.7 CKScan::GetCommNumber

The **GetCommNumber** function returns the number of current COM port with CKScan::Open.

---

```
int GetCommNumber();
```

---

#### Parameters

None

### Return Values

If the function succeeds, the return value is the COM port number.  
If the function fails or there is no open port, the return value is zero.

### Remarks

None

#### 4.4.1.8 CKScan::GetLastErrorMsg

The **GetLastErrorMsg** function returns a pointer to a string with description of the last error occurred in KScanBar.dll.

---

```
LPCTSTR GetLastErrorMsg();
```

---

### Parameters

None

### Return Values

The function returns a pointer to a string with description of the last error occurred in KScanBar.dll.

### Remarks

None

#### 4.4.1.9 CKScan::GetVersionInfo

The **GetVersionInfo** function returns a pointer to a string with version information associated with the current port.

---

```
LPCTSTR GetVersionInfo();
```

---

### 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

#### 4.4.1.10 CKScan::Sleep

The Sleep function puts the Scan Module into sleep mode.

---

```
BOOL Sleep();
```

---

### 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 advised to fully utilize sleep mode capability of the Scan Module. KSCAN\_SLEEP\_NOW puts the Scan Module into power saving sleep mode allowing only minimum current drain. It should be noted that waking up from sleep mode will incur additional time of up to 10 msec.

#### 4.4.1.11 CKScan::Wakeup

The Wakeup function wakes up the Scan Module that was previously put in sleep mode.

---

```
BOOL KScanWakeup( ) ;
```

---

##### **Parameters**

None

##### **Return Values**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

##### **Remarks**

Typical time delay for waking up from sleeping mode is measured to be less than 10 msec.

#### 4.4.1.12 CKScan::Reset

The **Reset** function resets the Scan Module.

---

```
BOOL Reset( ) ;
```

---

##### **Parameters**

None

##### **Return Values**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

##### **Remarks**

This function is valid only after a successful call to CKScan::Open.

If the scan engine is in sleep mode, it is necessary to issue CKScan::Wakeup first, otherwise the function call will result in fail.

#### 4.4.2 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

```
typedef struct _MCBarCodeType
{
    // BARCODE 28
    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;
}MCBarCodeType,* PMCBarCodeType;

typedef struct _MCReadOption
{
    BYTE bMC_WIDESCANGANGLE;
    BYTE bMC_RETURNCHECK;
    BYTE bMC_ERRORCHECK;
    BYTE bMC_HIGHFILTERMODE;
}MCReadOption,* PMCReadOption;

typedef struct _MCModuleOption
{
    int nMC_TimeOutSec;
    int nMC_MinLen;
    int nMC_SecurityLevel;
```

```

}MCModuleOption,* PMCModuleOption;

////////////////////////////////////
// BARCODE OPTION STRUCT

typedef struct _MCBarOption_UPCA{
    BYTE    bMC_UPCA_Enable;
    BYTE    bMC_UPCA_XNum;
    BYTE    bMC_UPCA_XCD;
    BYTE    bMC_UPCA_AS_EAN13;
    BYTE    bMC_UPCA_AddOn;
}BarOption_UPCA, *PBarOption_UPCA;

typedef struct _MCBarOption_UPCE{
    BYTE    bMC_UPCE_Enable;
    BYTE    bMC_UPCE_XNum;
    BYTE    bMC_UPCE_XCD;
    int     nMC_UPCE_Convert;
}BarOption_UPCE, *PBarOption_UPCE;

typedef struct _MCBarOption_EAN13{
    BYTE    bMC_EAN13_Enable;
    BYTE    bMC_BOOKLAND_Enable;
    BYTE    bMC_EAN13_XCD;
    BYTE    bMC_EAN13_AddOn;
}BarOption_EAN13, *PBarOption_EAN13;

typedef struct _MCBarOption_EAN8{
    BYTE    bMC_EAN8_Enable;
    BYTE    bMC_EAN8_XCD;
    BYTE    bMC_EAN8_AS_EAN13;
}BarOption_EAN8, *PBarOption_EAN8;

typedef struct _MCBarOption_CODE39{
    BYTE    bMC_CODE39_Enable;
    BYTE    bMC_CODE32_Enable;
    BYTE    bMC_PZN_Enable;
    BYTE    bMC_CODE39_CDV;
    BYTE    bMC_CODE39_XCD;
    BYTE    bMC_CODE39_FullASCII;
    int     nMC_CODE39_MinLen;
    int     nMC_CODE39_MaxLen;
}BarOption_CODE39, *PBarOption_CODE39;

typedef struct _MCBarOption_CODE128{
    BYTE    bMC_CODE128_Enable;
    BYTE    bMC_UCCEAN128_Enable;
    int     nMC_CODE128_MinLen;
    int     nMC_CODE128_MaxLen;
}BarOption_CODE128, *PBarOption_CODE128;

typedef struct _MCBarOption_CODE93{
    BYTE    bMC_CODE93_Enable;
    int     nMC_CODE93_MinLen;
    int     nMC_CODE93_MaxLen;
}BarOption_CODE93, *PBarOption_CODE93;

typedef struct _MCBarOption_CODE35{
    BYTE    bMC_CODE35_Enable;
}BarOption_CODE35, *PBarOption_CODE35;

typedef struct _MCBarOption_CODE11{
    BYTE    bMC_CODE11_Enable;

```



```

    BYTE    bMC_CODE11_XCD;
    int     nMC_CODE11_CDV;
    int     nMC_CODE11_MinLen;
    int     nMC_CODE11_MaxLen;
}BarOption_CODE11, *PBarOption_CODE11;

typedef struct _MCBarOption_I2OF5{
    BYTE    bMC_I2OF5_Enable;
    BYTE    bMC_ITF14_Enable;
    BYTE    bMC_MATRIX2OF5_Enable;
    BYTE    bMC_DLOGIG_Enable;
    BYTE    bMC_INDUSTRY_Enable;
    BYTE    bMC_IATA_Enable;
    BYTE    bMC_I2OF5_CDV;
    BYTE    bMC_I2OF5_XCD;
    int     nMC_I2OF5_MinLen;
    int     nMC_I2OF5_MaxLen;
}BarOption_I2OF5, *PBarOption_I2OF5;

typedef struct _MCBarOption_CODABAR{
    BYTE    bMC_CODABAR_Enable;
    BYTE    bMC_CODABAR_XSS;
    int     nMC_CODABAR_MinLen;
    int     nMC_CODABAR_MaxLen;
}BarOption_CODABAR, *PBarOption_CODABAR;

typedef struct _MCBarOption_MSI{
    BYTE    bMC_MSI_Enable;
    BYTE    bMC_MSI_CDV;
    BYTE    bMC_MSI_XCD;
    int     nMC_MSI_MinLen;
    int     nMC_MSI_MaxLen;
}BarOption_MSI, *PBarOption_MSI;

typedef struct _MCBarOption_PLESSEY{
    BYTE    bMC_PLESSEY_Enable;
    BYTE    bMC_PLESSEY_CDV;
    BYTE    bMC_PLESSEY_XCD;
    int     nMC_PLESSEY_MinLen;
    int     nMC_PLESSEY_MaxLen;
}BarOption_PLESSEY, *PBarOption_PLESSEY;

typedef struct _MCBarOption_GS1{
    BYTE    bMC_GS1_Enable;
    BYTE    bMC_GS1LIM_Enable;
    BYTE    bMC_GS1EXP_Enable;
}BarOption_GS1, *PBarOption_GS1;

typedef struct _MCBarOption_TELEPEN{
    BYTE    bMC_TELEPEN_Enable;
    BYTE    bMC_TELEPEN_OldStyle;
}BarOption_TELEPEN, *PBarOption_TELEPEN;

```

#### 4.4.2.1 ScannerControl.ScanInit

The **ScanInit** function opens a COM port to Scan Module, and performs initialization.

---

```
int ScanInit();
```

---

##### 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#

```
Bool m_nResult = ScanCtrl.ScanInit();
if(m_nResult != 0)
{
    MessageBox.Show("Scanner Init Failed");
}
```

---

#### 4.4.2.2 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#

```
ScanCtrl.ScanClose();
```

---

#### 4.4.2.3 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;
    }

    m_bReading = false;
    ScanCtrl.ScanRead();
}
```

---

### 4.4.2.4 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 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();
```

---

### 4.4.2.5 ScannerControl.SetBarcodeType

The SetBarcodeType function sets Structure of Symbology.

---

```
void SetBarcodeType(ref MCBarCodeType pBCT)
```

---

## Parameters

*MCBarCodeType*

Specifies pointer to the MCSSLibNet Structure

## Return Values

None

## Remarks

Use after successful initialization of the scanner.

---

C#

```
private MCSSLibNet.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);
```

---

#### 4.4.2.6 ScannerControl.GetBarcodeType

The GetBarCodeType function gets Structure of Symbology.

---

```
void GetBarCodeType(out MCBarCodeType pBCT)
```

---

##### Parameters

*MCBarCodeType*

Specifies pointer to the MCSSLibNet Structure

##### Return Values

None

##### Remarks

Use after successful initialization of the scanner.

---

C#

```
private MCSSLibNet.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_GSL;
m_bGslLim = M3BarCodeType.bMC_GSL_LIMITED;
m_bGslExp = M3BarCodeType.bMC_GSL_EXPANDED;

```

#### 4.4.2.7 ScannerControl.SetModuleOption

The SetModuleOption function sets Structure of ModuleOption.

---

```
void SetModuleOption(ref MCModuleOption pMDO)
```

---

##### Parameters

*MCModuleOption*

Specifies pointer to the MCSSLibNet Structure

##### Return Values

None

##### Remarks

Use after successful initialization of the scanner.

---

C#

```

private MCSSLibNet.MCModuleOption M3ModuleOption;

M3ModuleOption = new MCModuleOption();

M3ModuleOption.nMC_TimeOutSec = 10;
M3ModuleOption.nMC_SecurityLevel = 1;
M3ModuleOption.nMC_MinLen = 3

ScanCtrl.SetModuleOption(ref M3ModuleOption);

```

---

#### 4.4.2.8 ScannerControl.GetModuleOption

The GetModuleOption function gets Structure of ModuleOption.

---

```
void GetModuleOption(out MCMModuleOption pMDO)
```

---

##### Parameters

*MCMModuleOption*

Specifies pointer to the MCSSLibNet 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 MCSSLibNet.MCMModuleOption M3ModuleOption;

M3ModuleOption = new MCMModuleOption();

ScanCtrl.GetModuleOption(out M3ModuleOption);

m_nSecurityLevel = M3ModuleOption.nMC_TimeOutSec;
m_nTimeOut = M3ModuleOption.nMC_SecurityLevel;
m_nMinLen = M3ModuleOption.nMC_MinLen;
```

---

#### 4.4.2.9 ScannerControl.SetReadOption

The SetReadOption function sets Structure of ReadOption.

---

```
void SetReadOption(ref MCReadOption pRDO)
```

---

##### Parameters

*MCReadOption*

Specifies pointer to the MCSSLibNet Structure

##### Return Values

None

##### Remarks

Use after successful initialization of the scanner.

---

C#

```
private MCSSLibNet.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);
```

---

#### 4.4.2.10 ScannerControl.GetReadOption

The SetReadOption function sets Structure of ReadOption.

---

```
void SetReadOption(ref MCReadOption pRDO)
```

---

##### Parameters

*MCReadOption*

Specifies pointer to the MCSSLibNet 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 MCSSLibNet.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;
```

#### 4.4.2.11 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.

#### 4.4.2.12 ScannerControl.RegisterReceiveForm

The RegisterRecieveForm This function registers a form as a receiver of scan message.

---

```
void RegisterRecieveForm()
```

---

##### Parameters

None

##### Return Values

None

##### Remarks

If a form is intended to process scan message, this function should be called when a form is initialized or activated.

#### 4.4.2.13 ScannerControl.UseDefaultSound

The **UseDefaultSound** function sound come after finishing ScanRead Success.

---

```
void UseDefaultSound(bool bSound, string sndPath)
```

---

##### Parameters

*bSound*

Determines whether Sound is used or not. True if default sound is used and false if default sound is not used.

*sndPath*

Assign the path where Wav files are located.

In the case of NULL, the default wav is played.

##### Return Values

None

##### Remarks

Default sound is j°\Windows\alarm4.wav.

#### 4.4.2.14 ScannerControl.UseResumeMsg

The **UseResumeMsg** function is Default Display Scanner Initialize.message window when resuming.

---

```
void UseResumeMsg(bool bResume)
```

---

##### Parameters

*bResume*

Determines whether message is used or not. True if message is used and false if message is not used.

##### Return Values

None

##### Remarks

Default sound is\Windows\alarm4.wav.

#### 4.4.2.15 ScannerControl.RegHotKey

The **RegHotKey** function registers Scanner Button as a hot key.

---

```
bool RegHotKey(int id,uint vk,bool SyncMode);
```

---

##### Parameters

*Id*

Identifier of the hot key. No other hot key in the calling thread should have the same identifier. An application must specify a value in the range 0x0000 through 0xBFFF

*vk*

Vitrual Key value

*SyncMode*

In the case of If true?-> SyncMode

In the case of If false?-> AsyncMode

##### 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 can be used in CF 1.x because of it's complexity to get the event regarding KeyDown and KeyUp.

If RegHotkey is used as hotkey, it should be freed before the application ends.

#### 4.6.2.16 ScannerControl.UnRegHotKey

The UnRegHotKey function Free Scanner Button as a hot key.

---

```
bool UnRegHotKey(int id);
```

---

### Parameters

*Id*

Identifier of the hot key to be freed

### 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 can be used in CF 1.x because of it's complexity to get the event regarding KeyDown and KeyUp.

If RegHotkey is used as hotkey, it should be freed before the application ends.

#### 4.4.2.17 ScannerControl.SetBarOptionUPCA

The **SetBarOptionUPCA** function sets 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);
```

---

#### 4.4.2.18 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;
```

---

#### 4.4.2.19 ScannerControl.SetBarOptionUPCE

The **SetBarOptionUPCE** function set 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);
```

---

#### 4.4.2.20 ScannerControl.GetBarOptionUPCE

The **GetBarOptionUPCA** 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;
```

---

#### 4.6.2.21 ScannerControl.SetBarOptionEAN13

The SetBarOptionEAN13 function set 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);
```

---

#### 4.4.2.22 ScannerControl.GetBarOptionEAN13

The SetBarOptionEAN13 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;
```

---

#### 4.4.2.23 ScannerControl.SetBarOptionEAN8

The SetBarOptionEAN8 function sets 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);
```

#### 4.4.2.24 ScannerControl.GetBarOptionEAN8

The GetBarOptionEAN8 function gets 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;
```

#### 4.4.2.25 ScannerControl.SetBarOptionCODE39

The SetBarOptionCODE39 function sets 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);
```

---

#### **4.4.2.26 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;
```

---

#### **4.4.2.27 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);
```

---

#### **4.4.2.28 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;
```

---

#### **4.4.2.29 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#

```
MCTBarOption_CODE93 pCode93 = new MCTBarOption_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);
```

---

#### 4.4.2.30 ScannerControl.GetBarOptionCODE93

The GetBarOptionCODE93 function gets the option of CODE93 Barcode.

---

```
void GetBarOptionCODE93(out MCTBarOption_CODE93 pCode93);
```

---

##### Parameters

*MCTBarOption\_CODE93*

Specifies pointer to the MCSSLibNet Structure

##### Return Values

None

##### Remarks

Use after successful initialization of the scanner.

---

C#

```
MCTBarOption_CODE93 pCode93 = new MCTBarOption_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;
```

---

#### 4.4.2.31 ScannerControl.SetBarOptionCODE35

The SetBarOptionCODE35 function sets the option of CODE35 Barcode.

---

```
void SetBarOptionCODE35(ref MCTBarOption_CODE35 pCode35);
```

---

##### Parameters

*MCTBarOption\_CODE35*

Specifies pointer to the MCSSLibNet Structure

##### Return Values

None

##### Remarks

Use after successful initialization of the scanner.

---

C#

```
MCTBarOption_CODE35 pCode35 = new MCTBarOption_CODE35();

pCode35.bMC_CODE35_Enable = m_bCode35 = Code35Dlg.m_bEnable;

ScanCtrl.SetBarOptionCODE35(ref pCode35);
```

---

#### 4.4.2.32 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;
```

---

#### 4.4.2.33 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);
```

---

#### 4.4.2.34 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;
```

---

### 4.4.2.35 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_INDUSTRY_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);
```

---

### 4.4.2.36 ScannerControl.GetBarOptionI2OF5

The SetBarOptionCODEI2OF5 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#

```
MCSBarOption_I2OF5 pI2of5 = new MCSBarOption_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;
```

---

#### 4.6.2.37 ScannerControl.SetBarOptionCODABAR

The SetBarOptionCODABAR function sets the option of CODABAR Barcode.

---

```
void SetBarOptionCODABAR(ref MCSBarOption_CODABAR pCodabar);
```

---

##### Parameters

*MCSBarOption\_CODABAR*

Specifies pointer to the MCSLibNet Structure

##### Return Values

None

##### Remarks

Use after successful initialization of the scanner.

---

C#

```
MCSBarOption_CODABAR pCodabar = new MCSBarOption_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);
```

---

#### 4.4.2.38 ScannerControl.GetBarOptionCODABAR

The SetBarOptionCODABAR function gets the option of CODABAR Barcode.

---

```
void GetBarOptionCODABAR(out MCSBarOption_CODABAR pCodabar);
```

---

##### Parameters

*MCSBarOption\_CODABAR*

Specifies pointer to the MCSLibNet 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;

```

#### 4.4.2.39 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);

```

#### 4.4.2.40 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;
```

#### 4.4.2.41 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);
```

#### 4.4.2.42 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;
```

#### 4.4.2.43 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);
```

#### 4.4.2.44 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;
Gs1Dlg.m_bGs1Lim = pGs1.bMC_GS1LIM_Enable;
Gs1Dlg.m_bGs1Exp = pGs1.bMC_GS1EXP_Enable;
```

#### 4.4.2.45 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);
```

---

#### 4.4.2.46 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;
```

---

## 4.5 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 T with 2D scanner that uses software decoder.

## 4.5.1 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 dwFirmwareChecksum;
    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

```

#### 4.5.1.1 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.

#### 4.5.1.2 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.

#### 4.5.1.3 CancelIO

This function cancels the current bar code capture.

---

```
BOOL CancelIO( )
```

---

##### Return Value

true indicates success.

false indicates failure.

#### 4.5.1.4 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.

#### 4.5.1.5 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.

#### 4.5.1.6 SetScanningLightsMode

This function gives the user the ability to select what the illumination and aimer do during imaging.

---

```
BOOL SetScanningLightsMode(  
    ScanIlluminat nIllumMode  
)
```

---

#### Parameters

*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.

#### 4.5.1.7 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.

#### **4.5.1.8 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.

#### **4.5.1.9 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.

#### **4.5.1.10 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.

#### 4.5.1.11 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

#### 4.5.1.12 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.

#### 4.5.1.13 ScanLed

This function turns the terminal's bacode 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.

**4.5.1.14 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.

**4.5.1.15 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.

**4.5.1.16 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.

#### 4.5.1.17 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.

#### 4.5.1.18 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.

#### 4.5.1.19 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.

## 4.5.2 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

**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 dwFirmwareChecksum;
    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;
}

```

#### 4.5.2.1 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.

#### 4.5.2.2 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.

#### 4.5.2.3 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.

#### 4.5.2.4 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.

#### 4.5.2.5 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.

#### 4.5.2.6 Scanner.CancelIo Method

This function cancels the current bar code capture.

---

```
public bool CancelIo( )
```

---

##### Parameters

None.

##### Return Value

true indicates success.

false indicates failure.

#### 4.5.2.7 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.

#### 4.5.2.8 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.

#### 4.5.2.9 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.

#### 4.5.2.10 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.

#### 4.5.2.11 Scanner.ScanLed Method

This function turns the terminal's bacode 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.

**4.5.2.12 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.

**4.5.2.13 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 disable.

**Return Value**

true indicates success.

false indicates failure.

**4.5.2.14 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.