Zebra Integrated RFID SDK for Xamarin Code Snippet

Date: 28-Apr-2023

Table of Contents

1. Setup RFID SDK

Following code segments provide the setup procedure for the RFID SDK.

```
IsrfidISdkApi apiInstance;

apiInstance = srfidSdkFactory.CreateRfidSdkApiInstance;
apiInstance.SrfidSetDelegate(instance);

apiInstance.SrfidSetOperationalMode((int)NativeRfidOpMode.OPMODE_MFI);

apiInstance.SrfidSubsribeForEvents((int)NotificationsRFID.READER_APPEARANCE
+ (int)NotificationsRFID.READER_DISAPPEARANCE +
(int)NotificationsRFID.SESSION_ESTABLISHMENT +
(int)NotificationsRFID.SESSION_TERMINATION);

apiInstance.SrfidSubsribeForEvents((int)NotificationsRFID.MASK_READ +
(int)NotificationsRFID.MASK_STATUS + (int)NotificationsRFID.MASK_PROXIMITY
+ (int)NotificationsRFID.MASK_TRIGGER);

apiInstance.SrfidSubsribeForEvents((int)NotificationsRFID.MASK_BATTERY +
(int)NotificationsRFID.MASK_STATUS_OPERENDSUMMARY +
(int)NotificationsRFID.MASK_TEMPERATURE +
(int)NotificationsRFID.MASK_POWER);

apiInstance.SrfidSubsribeForEvents((int)NotificationsRFID.MASK_DATABASE +
(int)NotificationsRFID.MASK_RADIOERROR);

                    apiInstance.SrfidEnableAvailableReadersDetection(true);

apiInstance.SrfidEnableAutomaticSessionReestablishment(true);
```

2. Get RFID SDK version

RFID SDK version information could be obtained as follows:

```
   apiInstance.GetSrfidGetSdkVersion();
```

3. Get RFID available scanners

Following code segment outputs the paired device list. Reader must be paired with the iOS device via Bluetooth before query action.

```
public void getNativeRfidSdkReaderList()
{
            //Get avilable readers
            NSMutableArray availableReaders = new NSMutableArray();

            IntPtr availableHandle = availableReaders.Handle;
            SrfidResult availableReaderResult =
apiInstance.SrfidGetAvailableReadersList(out availableHandle);
            availableReaders =
ObjCRuntime.Runtime.GetNSObject<NSMutableArray>(availableHandle);
```

```
            if (availableReaderResult == SrfidResult.Success)
            {
                System.Diagnostics.Debug.WriteLine("Native
SrfidGetAvailableReadersList : Success" + availableReaders);

            }
            else if (availableReaderResult == SrfidResult.ResponseError)
            {


System.Diagnostics.Debug.WriteLine("SrfidGetAvailableReadersList
ResponseError");
            }
            else if (availableReaderResult == SrfidResult.Failure ||
availableReaderResult == SrfidResult.ResponseTimeout)
            {


System.Diagnostics.Debug.WriteLine("SrfidGetAvailableReadersList reder
prob");
            }



            if (availableReaders != null)
            {


                foreach (srfidReaderInfo reader in
NSArray.FromArray<NSObject>(availableReaders))
                {


                    System.Diagnostics.Debug.WriteLine("Native Readers " +
reader.ReaderName);


                }
            }



}
```

4. RFID Operation

4.1. Inventory

4.1.1 Inventory Start

RFID tag reading can be started as follows. Once started, tags in the range will be r ead continuously.

```
public string RfidStartInventory(int readerID)
 {
            string statusMessage = null;
            srfidTagReportConfig tagReportConfiguration =
this.RfidGetTagReportConfiguration(readerID);
            srfidReportConfig reportConfigaration = new
srfidReportConfig();

reportConfigaration.SetIncFirstSeenTime(tagReportConfiguration.IncFirstSeen
Time);

reportConfigaration.SetIncLastSeenTime(tagReportConfiguration.IncLastSeenTi
me);
            reportConfigaration.SetIncPC(tagReportConfiguration.IncPC);
            reportConfigaration.SetIncRSSI(tagReportConfiguration.IncRSSI);

reportConfigaration.SetIncPhase(tagReportConfiguration.IncPhase);

reportConfigaration.SetIncChannelIndex(tagReportConfiguration.IncChannelIdx
);

reportConfigaration.SetIncTagSeenCount(tagReportConfiguration.IncTagSeenCou
nt);

            srfidAccessConfig accessConfig = new srfidAccessConfig();

            SrfidResult statusStartInventory =
apiInstance.SrfidStartInventory(readerID, SrfidMemorybank.None,
reportConfigaration, accessConfig, out statusMessage);

            if (statusMessage == "Inventory Started in Batch Mode")
            {
                return "Success";
            }
            else
            {
                return statusStartInventory.ToString();
            }
}
```

4.1.2 Inventory Stop

RFID tag reading cycle can be terminated as follows.

```
public void RfidStopInventory()
{
            string statusMessage = null;
            SrfidResult statusStopInventory =
apiInstance.SrfidStopInventory(connectedReaderID, out statusMessage);
            if (statusStopInventory == SrfidResult.Success)
            {

              System.Diagnostics.Debug.WriteLine("SrfidStopInventory
Success ");

            }
            else if (statusStopInventory == SrfidResult.ResponseError)
            {

                System.Diagnostics.Debug.WriteLine("SrfidStopInventory
ResponseError");
            }
            else if (statusStopInventory == SrfidResult.Failure ||
statusStopInventory == SrfidResult.ResponseTimeout)
            {

                System.Diagnostics.Debug.WriteLine("SrfidStopInventory
reder prob");
            }
        }
```

4.1.3 Tag Data Event

This event triggers when tag data is received.

```
 public override void SrfidEventReadNotify(int readerID, srfidTagData
tagData)
        {

            System.Diagnostics.Debug.WriteLine("Native SrfidEventReadNotify
MemoryBankData " + tagData.MemoryBankData);
            System.Diagnostics.Debug.WriteLine("Native SrfidEventReadNotify


        }
```

4.2. Locate Tag

Following two methods are used to locate tags.

4.2.1. Start Locate Tag

Tag locating can be started as follows.

```
public void RfidStartTagLocationing( string epcID)
        {
            string statusMessage = null;
            SrfidResult statusStartTagLocation =
apiInstance.SrfidStartTagLocationing(connectedReaderID, epcID, out
statusMessage);


            if (statusStartTagLocation == SrfidResult.Success)
            {
                System.Diagnostics.Debug.WriteLine("Native
SrfidStartTagLocationing : Success");


            }
            else if (statusStartTagLocation == SrfidResult.ResponseError)
            {


System.Diagnostics.Debug.WriteLine("SrfidStartTagLocationing
ResponseError");

            }
            else if (statusStartTagLocation == SrfidResult.Failure ||
statusStartTagLocation == SrfidResult.ResponseTimeout)
            {


System.Diagnostics.Debug.WriteLine("SrfidStartTagLocationing reder prob");

            }
        }
```

4.2.2. Stop Locate Tag

Stop locating tags.

```
public void RfidStopTagLocationing()
        {
            string statusMessage = null;
            SrfidResult statusStopTagLocation =
apiInstance.SrfidStopTagLocationing(connectedReaderID, out statusMessage);


            if (statusStopTagLocation == SrfidResult.Success)
            {
                System.Diagnostics.Debug.WriteLine("Native
SrfidStopTagLocationing : Success");


            }
            else if (statusStopTagLocation == SrfidResult.ResponseError)
            {

                System.Diagnostics.Debug.WriteLine("SrfidStopTagLocationing
ResponseError");

            }
            else if (statusStopTagLocation == SrfidResult.Failure ||
statusStopTagLocation == SrfidResult.ResponseTimeout)
            {

                System.Diagnostics.Debug.WriteLine("SrfidStopTagLocationing
reder prob");

            }
        }
```

4.2.3. Proximity Event

This event will trigger when reception of a proximity notification during on-going tag locating operation from a connected RFID reader.

```
public override void SrfidEventProximityNotify(int readerID, int
proximityPercent)
        {
            System.Diagnostics.Debug.WriteLine("Native
SrfidEventProximityNotify  :" + proximityPercent + " %");

      }
```

5. Battery

The SDK also provides an ability to cause a particular active RFID reader to immediately send information about current battery status. The following example demonstrates both requesting and processing of asynchronous battery status related notifications.

5.1. Get battery status

```
public void GetBatteryStatus()
        {

            string statusMessage = "";
            NSMutableArray batteryStatusValueList = new NSMutableArray();
            IntPtr availableHandle = batteryStatusValueList.Handle;

            SrfidResult srfid_result = SrfidResult.Failure;
            //Retry for 2 times if we get any failure/timeout response
            for (int i = 0; i < 2; i++)
            {
                srfid_result =
apiInstance.SrfidGetBatteryStatus(connectedReaderID, out availableHandle,
out statusMessage);
                batteryStatusValueList =
ObjCRuntime.Runtime.GetNSObject<NSMutableArray>(availableHandle);

                if ((srfid_result != SrfidResult.ResponseTimeout) &&
(srfid_result != SrfidResult.Failure))
                {
                    break;
                }

            }


            if (srfid_result == SrfidResult.Success)
            {


                foreach (srfidRfidBatteryStatusInformation info in
batteryStatusValueList)
                {
                    System.Diagnostics.Debug.WriteLine("GetBatteryStatus
BatteryStatusTittle : " + info.BatteryStatusTittle);
                    System.Diagnostics.Debug.WriteLine("GetBatteryStatus
BatterStatusValue : " + info.BatterStatusValue);
                    logsString =  logsString + "\n" + "Battery Status Title
:" + info.BatteryStatusTittle + " value : " + info.BatterStatusValue;

                }

            }
            else if (srfid_result == SrfidResult.ResponseError)
            {

               System.Diagnostics.Debug.WriteLine("GetBatteryStatus
ResponseError");
            }
            else if (srfid_result == SrfidResult.Failure || srfid_result ==
SrfidResult.ResponseTimeout)
            {
                System.Diagnostics.Debug.WriteLine("GetBatteryStatus reder
prob");
            }

        }
```

5.2. Request battery status with the event

By using following method, we can get the battery status with the event.

```
public void requestBatteryStatus()
        {
            apiInstance.SrfidRequestBatteryStatus(connectedReaderID);

        }

       // Event
       public override void SrfidEventBatteryNotity(int readerID,
srfidBatteryEvent batteryEvent)
        {

            logsString =  "\n" + "SrfidEventBatteryNotity Power level  :" +
batteryEvent.PowerLevel + " %" + "\n"+   " Is charging : " +
batteryEvent.IsCharging;

        }
```

6. Trigger Mapping

6.1. Get Trigger Mapping

This “GetTriggerMapping” API will get the trigger key configuration.

```
public void GetTriggerMapping( SrfidNewEnumKeylayoutType upper,
SrfidNewEnumKeylayoutType lower)
        {

            upper = SrfidNewEnumKeylayoutType.NoAction;
            lower = SrfidNewEnumKeylayoutType.NoAction;

            SrfidResult srfid_result = SrfidResult.Failure;
            //Retry for 2 times if we get any failure/timeout response
            for (int i = 0; i < 2; i++)
            {
               srfid_result =
apiInstance.SrfidGetKeylayoutType(connectedReaderID,  out upper, out
lower);


                if ((srfid_result != SrfidResult.ResponseTimeout) &&
(srfid_result != SrfidResult.Failure))
                {
                    break;
                }
            }


            if (srfid_result == SrfidResult.Success)
            {

                System.Diagnostics.Debug.WriteLine("GetTriggerMapping
upperTriggerValue : " + upper);
                System.Diagnostics.Debug.WriteLine("GetTriggerMapping
lowerTriggerValue : " + lower);


            }
            else if (srfid_result == SrfidResult.ResponseError)
            {

                System.Diagnostics.Debug.WriteLine("GetTriggerMapping
ResponseError");
            }
            else if (srfid_result == SrfidResult.Failure ||
srfid_result == SrfidResult.ResponseTimeout)
            {

                System.Diagnostics.Debug.WriteLine("GetTriggerMapping
reder prob");
            }

        }
```

6.2. Set Trigger Mapping

This “SetTriggerMapping” API will set the trigger key.

```
public void SetTriggerMapping(SrfidNewEnumKeylayoutType upperTrigger ,
SrfidNewEnumKeylayoutType lowerTrigger)
        {
            SrfidNewEnumKeylayoutType upperTriggerValue = upperTrigger;
            SrfidNewEnumKeylayoutType lowerTriggerValue = lowerTrigger;

            SrfidResult srfid_result = SrfidResult.Failure;
            //Retry for 2 times if we get any failure/timeout response
            for (int i = 0; i < 2; i++)
            {
               srfid_result =
apiInstance.SrfidSetKeylayoutType(connectedReaderID, upperTriggerValue,
lowerTriggerValue);

                if ((srfid_result != SrfidResult.ResponseTimeout) &&
(srfid_result != SrfidResult.Failure))
                {
                    break;
                }
            }

            if (srfid_result == SrfidResult.Success)
            {
                System.Diagnostics.Debug.WriteLine("GetTriggerMapping
SrfidResult.Success");

            }
            else if (srfid_result == SrfidResult.ResponseError)
            {

                System.Diagnostics.Debug.WriteLine("GetTriggerMapping
ResponseError");
            }
            else if (srfid_result == SrfidResult.Failure || srfid_result ==
SrfidResult.ResponseTimeout)
            {
                System.Diagnostics.Debug.WriteLine("GetTriggerMapping reder
prob"); }
}
```

7. Access Operation

7.1. Tag Read

Following values should be passed as arguments to *AccessOperationTagRead* API and it will return a TagData object.

tagId - string

tagAccessPassword - string

byteCount - short

offset - short

memoryBank – MemoryBank

- MEMORYBANK_EPC
- MEMORYBANK_TID
- MEMORYBANK_USER
- MEMORYBANK_RESV
- MEMORYBANK_NONE
- MEMORYBANK_ACCESS
- MEMORYBANK_KILL

```
public void AccessOperationTagRead( string tagId, SrfidMemorybank
memoryBank, short offset, short length, int password)
        {
            string statusMessage = null;
            SrfidResult tagReadResult = SrfidResult.Failure;

            srfidTagData tagData = new srfidTagData();
            IntPtr availableHandle = tagData.Handle;


            //Retry for 2 times if we get any failure/timeout response
            for (int i = 0; i < 2; i++)
            {
                tagReadResult = apiInstance.SrfidReadTag(connectedReaderID,
tagId, out availableHandle, memoryBank, offset, length, password, out
statusMessage);
                tagData =
ObjCRuntime.Runtime.GetNSObject<srfidTagData>(availableHandle);

                if ((tagReadResult != SrfidResult.ResponseTimeout) &&
(tagReadResult != SrfidResult.Failure))
                {
                    break;
                }

            }



            if (tagReadResult == SrfidResult.Success)
            {
                System.Diagnostics.Debug.WriteLine("Native SrfidReadTag
Memory Bank Data :" + tagData.MemoryBankData);


            }
            else if (tagReadResult == SrfidResult.ResponseError)
            {
                System.Diagnostics.Debug.WriteLine("SrfidReadTag
ResponseError");
            }
            else if (tagReadResult == SrfidResult.Failure || tagReadResult
== SrfidResult.ResponseTimeout)
            {

                System.Diagnostics.Debug.WriteLine("SrfidReadTag reder
prob");
            }

        }
```

7.2. Tag Write

Following values should be passed as arguments to AccessOperationTagWrite API and it will return a boolean value whether the write operation is successful or not.

tagId - string
tagAccessPassword - string
tagData - string
offset - short
memoryBank - MemoryBank

- MEMORYBANK_EPC
- MEMORYBANK_TID
- MEMORYBANK_USER
- MEMORYBANK_RESV
- MEMORYBANK_NONE
- MEMORYBANK_ACCESS
- MEMORYBANK_KILL

blockWrite - bool

```
public bool AccessOperationTagWrite(string tagId, SrfidMemorybank
memoryBank, short offset, string data, int password, bool blockWrite)
        {
            string statusMessage = null;
            bool status = false;
            SrfidResult tagWriteResult = SrfidResult.Failure;

            srfidTagData tagData = new srfidTagData();
            IntPtr availableHandle = tagData.Handle;

            //Retry for 2 times if we get any failure/timeout response
            for (int i = 0; i < 2; i++)
            {
                tagWriteResult =
apiInstance.SrfidWriteTag(connectedReaderID, tagId, out availableHandle,
memoryBank, offset, data, password, blockWrite, out statusMessage);
                tagData =
ObjCRuntime.Runtime.GetNSObject<srfidTagData>(availableHandle);

                if ((tagWriteResult != SrfidResult.ResponseTimeout) &&
(tagWriteResult != SrfidResult.Failure))
                {
                    break;
                }

            }



            if (tagWriteResult == SrfidResult.Success)
            {
                System.Diagnostics.Debug.WriteLine("Native SrfidWriteTag :"
+ tagData.TagId);
                status = true;

            }
            else if (tagWriteResult == SrfidResult.ResponseError)
            {
                status = false;
                System.Diagnostics.Debug.WriteLine("SrfidWriteTag
ResponseError");
            }
            else if (tagWriteResult == SrfidResult.Failure ||
tagWriteResult == SrfidResult.ResponseTimeout)
            {
                status = false;
                System.Diagnostics.Debug.WriteLine("SrfidWriteTag reder
prob");
            }

            return status;
        }
```

7.3. Tag Lock

Following values should be passed as arguments to AccessOperationTagLock API and it will return a boolean value whether the lock operation is successful or not.
tagId - string
tagAccessPassword - string
memoryBank - MemoryBank

- MEMORYBANK_EPC
- MEMORYBANK_TID
- MEMORYBANK_USER
- MEMORYBANK_RESV
- MEMORYBANK_NONE
- MEMORYBANK_ACCESS
- MEMORYBANK_KILL

lockPrivilege

- READ_WRITE
- PERMANENT_LOCK
- PERMANENT_UNLOCK
- UNLOCK

```
public bool AccessOperationTagLock( string tagId, SrfidMemorybank
memoryBank, SrfidAccesspermission accessPermission, int password)
        {
            string statusMessage = null;
            bool status = false;
            SrfidResult tagLockResult = SrfidResult.Failure;

            srfidTagData tagData = new srfidTagData();
            IntPtr availableHandle = tagData.Handle;

            //Retry for 2 times if we get any failure/timeout response
            for (int i = 0; i < 2; i++)
            {
                tagLockResult = apiInstance.SrfidLockTag(connectedReaderID,
tagId, out availableHandle, memoryBank, accessPermission, password, out
statusMessage);
                tagData =
ObjCRuntime.Runtime.GetNSObject<srfidTagData>(availableHandle);

                if ((tagLockResult != SrfidResult.ResponseTimeout) &&
(tagLockResult != SrfidResult.Failure))
                {
                    break;
                }

            }



            if (tagLockResult == SrfidResult.Success)
            {
                System.Diagnostics.Debug.WriteLine("Native SrfidLockTag :"
+ tagData.TagId);
                status = true;

            }
            else if (tagLockResult == SrfidResult.ResponseError)
            {

                System.Diagnostics.Debug.WriteLine("SrfidLockTag
ResponseError");
                status = false;
            }
            else if (tagLockResult == SrfidResult.Failure || tagLockResult
== SrfidResult.ResponseTimeout)
            {

                System.Diagnostics.Debug.WriteLine("SrfidLockTag reder
prob");
                status = false;
            }

            return status;
        }
```

7.4. Tag Kill

Following values should be passed as arguments to `AccessOperationTagKill` API and it will return a boolean value whether the kill operation is successful or not.

`readerID` - int

`tagId` – string

`password - int`

```
public bool AccessOperationTagKill(int readerID, string tagId, int
password)
{

            string statusMessage = null;
            bool status = false;

            srfidTagData tagData = new srfidTagData();
            IntPtr availableHandle = tagData.Handle;
            SrfidResult tagKillResult = apiInstance.SrfidKillTag(readerID,
tagId, out availableHandle, password, out statusMessage);
            tagData =
ObjCRuntime.Runtime.GetNSObject<srfidTagData>(availableHandle);


            if (tagKillResult == SrfidResult.Success)
            {
                System.Diagnostics.Debug.WriteLine("Native SrfidKillTag :
Success" );
                status = true;

            }
            else if (tagKillResult == SrfidResult.ResponseError)
            {

                System.Diagnostics.Debug.WriteLine("SrfidKillTag
ResponseError");
                status = false;
            }
            else if (tagKillResult == SrfidResult.Failure || tagKillResult
== SrfidResult.ResponseTimeout)
            {

                System.Diagnostics.Debug.WriteLine("SrfidKillTag reder
prob");
                status = false;
            }


            return status;


  }
```

8. Access Sequence

This API is used to execute multiple access operations (Read, Write, etc) at the same time.

```
public void AccessSequence(string fillterData ,string fillteMask)
        {

            // initialize access criteria
            srfidAccessCriteria accessCriteria = new srfidAccessCriteria();
            //// setup tag filter 1
            srfidTagFilter tagFilter1 = new srfidTagFilter();
            tagFilter1.SetFilterMaskBank(SrfidMemorybank.Epc);
            tagFilter1.SetFilterData(fillterData);
            tagFilter1.SetFilterDoMatch(true);
            tagFilter1.SetFilterMask(fillteMask);
            tagFilter1.SetFilterMaskStartPos(2);
            tagFilter1.SetFilterMatchLength(2);

            accessCriteria.TagFilter1 = tagFilter1;

            // Set access criteria pram for EPC read
            srfidAccessParameters accesParamsEPCRead = new
srfidAccessParameters();
            accesParamsEPCRead.AccessOperationCode =
SrfidAccessoperationcode.Read;
            accesParamsEPCRead.MemoryBank = SrfidMemorybank.Epc;
            accesParamsEPCRead.Offset = 2;
            accesParamsEPCRead.Length = 0;
            accesParamsEPCRead.Password = 00;

            // Set access criteria pram for TID read
            srfidAccessParameters accesParamsTIDRead = new
srfidAccessParameters();
            accesParamsTIDRead.AccessOperationCode =
SrfidAccessoperationcode.Read;
            accesParamsTIDRead.MemoryBank = SrfidMemorybank.Tid;
            accesParamsTIDRead.Offset = 0;
            accesParamsTIDRead.Length = 0;
            accesParamsTIDRead.Password = 00;



            NSMutableArray accessParametersArray = new NSMutableArray();
            accessParametersArray.Add(accesParamsEPCRead);
            accessParametersArray.Add(accesParamsTIDRead);

            SrfidResult resultPerformAccessInSequence;
            string status = null;
            resultPerformAccessInSequence =
apiInstance.SrfidPerformAccessInSequence(connectedReaderID, accessCriteria,
NSArray.FromArray<NSObject>(accessParametersArray), out status);

            if (resultPerformAccessInSequence == SrfidResult.Success)
            {
                System.Diagnostics.Debug.WriteLine("++====== Native result
PerformAccessInSequence : Success");


            }
            else if (resultPerformAccessInSequence ==
SrfidResult.ResponseError)
            {

                System.Diagnostics.Debug.WriteLine("result
PerformAccessInSequence ResponseError");

            }
            else if (resultPerformAccessInSequence == SrfidResult.Failure
|| resultPerformAccessInSequence == SrfidResult.ResponseTimeout)
            {

                System.Diagnostics.Debug.WriteLine("result
PerformAccessInSequence reder prob");

            }

        }

    }
}
```

9. Barcode SDK

9.1. Setup Barcode SDK

Following code segments provide the setup procedure for the barcode SDK.

```
ISbtSdkApi iosScannerApi;

iosScannerApi = SbtSdkFactory.CreateSbtSdkApiInstance;
                    iosScannerApi.SbtSetDelegate(instance);

iosScannerApi.SbtSubsribeForEvents((int)(NotificationsBarcodeSDK.EVENT_SCAN
NER_APPEARANCE | NotificationsBarcodeSDK.EVENT_SCANNER_DISAPPEARANCE |
NotificationsBarcodeSDK.EVENT_SESSION_ESTABLISHMENT |
NotificationsBarcodeSDK.EVENT_SESSION_TERMINATION |
NotificationsBarcodeSDK.EVENT_BARCODE ));

iosScannerApi.SbtEnableAvailableScannersDetection(true);
                    iosScannerApi.SbtSetOperationalMode(0x01);//MFI


public enum NotificationsBarcodeSDK
{
    EVENT_BARCODE = 1,
    EVENT_IMAGE = 2,
    EVENT_VIDEO = 4,
    EVENT_SCANNER_APPEARANCE = 8,
    EVENT_SCANNER_DISAPPEARANCE = 0x10,
    EVENT_SESSION_ESTABLISHMENT = 0x20,
    EVENT_SESSION_TERMINATION = 0x40,
    EVENT_RAW_DATA = 0x80
}
```

9.2. Get Barcode SDK Version

Barcode SDK version information can be obtained as follows:

```
  iosScannerApi.SbtGetVersion;
```

9.3. Get Available Barcode SDK Scanner List

Code segments to get the available scanner list as follows.

```
public List<SbtScannerInfo> GetAvailableScannerList()
{

            NSMutableArray availableScanners = new NSMutableArray();

            IntPtr availableHandle = availableScanners.Handle;
            SbtResult availableScannerResult =
iosScannerApi.SbtGetAvailableScannersList(out availableHandle);
            availableScanners =
ObjCRuntime.Runtime.GetNSObject<NSMutableArray>(availableHandle);

            if (availableScannerResult == SbtResult.Success)
            {
                System.Diagnostics.Debug.WriteLine("Native Barcode SDK
opModeStatus : Success");

            }
            else if (availableScannerResult == SbtResult.Failure)
            {
                System.Diagnostics.Debug.WriteLine("Native Barcode SDK
opModeStatus : Failure");
            }

            scannerList.Clear();
            if (availableScanners != null)
            {

                foreach (SbtScannerInfo scanner in
NSArray.FromArray<NSObject>(availableScanners))
                {
                    System.Diagnostics.Debug.WriteLine("");
                    scannerList.Add(scanner);
                }
            }

            return scannerList;
}
```

9.4. Connect to Reader in Barcode SDK

Following method is used to connect to the connect to the scanner at *scannerID*.

```
public void ConnectScanner(int scannerID)
{
            SbtResult scannerConnectedResult =
iosScannerApi.SbtEstablishCommunicationSession(scannerID);
            if (scannerConnectedResult == SbtResult.Success)
            {
                System.Diagnostics.Debug.WriteLine("Native Barcode SDK
SbtEstablishCommunicationSession : Success");

            }
            else if (scannerConnectedResult == SbtResult.Failure)
            {
                System.Diagnostics.Debug.WriteLine("Native Barcode SDK
SbtEstablishCommunicationSession : Failure");
            }
}
```

9.5. Barcode Event

Initializes the Barcode Event.

```
public override void SbtEventBarcodeData(NSData barcodeData, int
barcodeType, int scannerID)
        {
            System.Diagnostics.Debug.WriteLine("Native Barcode SDK
SbtEventBarcodeData barcodeType: " + barcodeType);

        }
```

10. Switch Mode into RFID or Scanner for RFD8500

Changes the mode type of the RFD8500 device programmatically.

```
//Set device mode to RFID or Scanner
public void SwitchModeIntoRfidOrBarcode(DeviceMode deviceMode)
{

            int attributeModeSwitch = 1664;
            string attributeTypeModeSwitch = "B";
            int rfidMode = 0;
            int scannerMode = 1;



            srfidAttribute attribute = new srfidAttribute();
            attribute.SetAttrNum(attributeModeSwitch);
            attribute.SetAttrType(attributeTypeModeSwitch);
            if (deviceMode == DeviceMode.RFID)
            {
                attribute.SetAttrVal(rfidMode.ToString());
            }
            else
            {
                attribute.SetAttrVal(scannerMode.ToString());
            }


            string statusMessage = null;
            SrfidResult setAttributeResult =
apiInstance.SrfidSetAttribute(connectedReaderID, attribute, out
statusMessage);

            if (setAttributeResult == SrfidResult.Success)
            {
                System.Diagnostics.Debug.WriteLine("Native
SrfidSetAttribute : Success" );

            }
            else if (setAttributeResult == SrfidResult.ResponseError)
            {

                System.Diagnostics.Debug.WriteLine("SrfidSetAttribute
ResponseError");
            }
            else if (setAttributeResult == SrfidResult.Failure ||
setAttributeResult == SrfidResult.ResponseTimeout)
            {


System.Diagnostics.Debug.WriteLine("SrfidGetAvailableReadersList reder
prob");
            }




}
```