ZEBRA RFID

SDK for Windows

Developer Guide

MN-003515-01

Terms of Use

Proprietary Statement

This manual contains proprietary information of Zebra Technologies Corporation and its subsidiaries ("Zebra Technologies"). It is intended solely for the information and use of parties operating and maintaining the equipment described herein. Such proprietary information may not be used, reproduced, or disclosed to any other parties for any other purpose without the express, written permission of Zebra Technologies.

Product Improvements

Continuous improvement of products is a policy of Zebra Technologies. All specifications and designs are subject to change without notice.

Liability Disclaimer

Zebra Technologies takes steps to ensure that its published Engineering specifications and manuals are correct; however, errors do occur. Zebra Technologies reserves the right to correct any such errors and disclaims liability resulting therefrom.

Limitation of Liability

In no event shall Zebra Technologies or anyone else involved in the creation, production, or delivery of the accompanying product (including hardware and software) be liable for any damages whatsoever (including, without limitation, consequential damages including loss of business profits, business interruption, or loss of business information) arising out of the use of, the results of use of, or inability to use such product, even if Zebra Technologies has been advised of the possibility of such damages. Some jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you.

Revision History

Changes to the original guide are listed below:

Change	Date	Description
MN-003515-01 Rev. A	03-2019	Initial Release

Contents

List of Figures

List of Tables

About This Guide

Introduction

The RFD8500 RFID SDK Windows Developer Guide provides installation and programming information that allows RFID application development for Windows 7+ and MC55 (.Net Compact Framework).

Supported RFID Readers

The following RFID Readers are supported:

- RFD8500
- MC55

Chapter Descriptions

Topics covered in this guide are as follows:

- Zebra RFID SDK for Windows Overview provides detailed information about developing applications using the Windows RFID SDK.
- Creating, Building, and Running Projects provides step-by-step instructions to import the RFID SDK module and build Windows applications (with Microsoft .net 4.5/Compact framework) to work with the RFD8500 reader.
- Demo Applications provides information about the demonstration applications available for the Windows 7 RFID SDK and the Mobile RFID SDK.

Notational Conventions

The following conventions are used in this document:

- **Bold** text is used to highlight the following:
  - Key names on a keypad
  - Button names on a screen
- Bullets (•) indicate:
  - Action items
  - Lists of alternatives
  - Lists of required steps that are not necessarily sequential
- Sequential lists (e.g., those that describe step-by-step procedures) appear as numbered lists.

Related Documents and Software

The following documents provide more information about the readers.

- RFID Scanner SDK for Windows API Reference Guide
- RFD8500 User Guide, p/n MN002065Axx.
- RFD8500i User Guide, p/n MN-002761-XX.
- RFD8500 Quick Start Guide, p/n MN002225AXX.
- RFD8500i Quick Start Guide, p/n MN-002760-XX
- RFD8500 Regulatory Guide, p/n MN002062AXX.
- RFD8500i Regulatory Guide, p/n MN-002856-xx.
- RFD8500/i RFID Developer Guide, p/n MN002222AXX.

For the latest version of this guide and all guides, go to: www.zebra.com/support.

Service Information

If you have a problem using the equipment, contact your facility's technical or systems support. If there is a problem with the equipment, they will contact the Zebra Global Customer Support Center at: www.zebra.com/support.

When contacting Zebra support, please have the following information available:

- Serial number of the unit
- Model number or product name
- Software type and version number.

Zebra responds to calls by e-mail, telephone or fax within the time limits set forth in support agreements.

If your problem cannot be solved by Zebra support, you may need to return your equipment for servicing and will be given specific directions. Zebra is not responsible for any damages incurred during shipment if the approved shipping container is not used. Shipping the units improperly can possibly void the warranty.

If you purchased your business product from a Zebra business partner, contact that business partner for support.

Provide Documentation Feedback

If you have comments, questions, or suggestions about this guide, send an email to EVM-Techdocs@zebra.com.

Zebra RFID SDK for Windows Overview

Introduction

This chapter provides detailed basic through advanced information about developing applications using the Windows RFID SDK.

The Zebra RFID SDK for Windows provides an API that can be used by external applications to manage and control RFID specific functionality of an RFD8500 RFID reader connected over Bluetooth. The Zebra RFID SDK for Windows also allows .Net Compact Framework Smart Device Mobile applications to be developed for the MC55 mobile computer which communicates with the RFD8500 reader.

The Zebra RFID SDK for Windows provides the ability to manage RFID reader connections, perform various operations with connected RFID readers, configure connected RFID readers, and retrieve other information related to connected RFID readers.

All available APIs are defined under the Symbol.RFID.SDK namespace. The application uses the interface IRfidReader to interact with a reader.

Use available IRfidReader interface to register for events, connect with readers, and after successful connection, perform required operations such as inventory.

If method calls fail, the corresponding method throws an exception. The application should call all API methods in try-catch blocks for handling exceptions.

Connecting to an RFID Reader

Connection is the first step to communicate with an RFID reader. Import the namespace to use the RFID API as shown below.

```
using Symbol.RFID.SDK;
using Symbol.RFID.SDK.Domain.Reader;
```

Create an IRemoteReaderManagement interface instance by using the RfidSdk.RemoteReaderManagementServicesFactory class. Create the method as follows:

```
IRemoteReaderManagement readerManagement =
          RfidSdk.RemoteReaderManagementServicesFactory.Create();
```

Next call GetReaders method of the IRemoteReaderManagement interface instance object that gives a list of all available/paired RFID readers with a Windows device/PC. Readers list is in the form of IRfidReaderInfo interface instance collection.

```
IList<IRfidReaderInfo>allReaders =
          readerManagement.GetReaders(ReaderSearchOptions.AllReaders);
```

The Table 1 lists the reader search options that can be specified as a parameter to GetReaders method.

Table 1 Reader Search Options

Reader Search Options	Description
`ReaderSearchOptions.AllReaders`	Gives a list of all available/paired RFID readers with a Windows device/PC
`ReaderSearchOptions.Connected`	Gives a list of all connected RFID readers with a Windows device/PC
`ReaderSearchOptions.NotConnected`	Gives a list of all paired but not connected RFID readers with a Windows device/PC

Next call the RfidSdk.RFIDReaderFactory.Create method with the IRfidReaderInfo instance of the device to communicate with the device as follows.

```
    IRfidReader reader = RfidSdk.RFIDReaderFactory.Create(allReaders[0]);
```

The returned IRfidReader reader interface is used for performing all operations with RFID reader. To connect with the reader; use IRfidReader instance Connect() method.

```
    // Establish connection to the RFID Reader
    reader.Connect();
```

In addition, the application can register for IRemoteReaderWatcher instance events in the following way to get notified of RFID readers getting added (paired) / removed(unpaired), connected/disconnected.

```
    IRemoteReaderWatcher readerWatcher = RfidSdk.RemoteReaderWatcherServicesFactory.Create();
      readerWatcher.ReaderAppeared += ReaderWatcher_ReaderAppeared;
      readerWatcher.ReaderDisappeared += ReaderWatcher_ReaderDisappeared;
      readerWatcher.ReaderConnected += ReaderWatcher_ReaderConnected;
      readerWatcher.ReaderDisconnected += ReaderWatcher_ReaderDisconnected;
    }
    private void ReaderWatcher_ReaderAppeared(object sender, ReaderStatusChangedEventArgs e)
    {
    }

    private void ReaderWatcher_ReaderDisappeared(object sender, ReaderStatusChangedEventArgs e)
    {
    }

    private void ReaderWatcher_ReaderDisconnected(object sender, ReaderStatusChangedEventArgs
e)
    {
    }

    private void ReaderWatcher_ReaderConnected(object sender, ReaderStatusChangedEventArgs e)
    {
    }
```

Special Connection Handling Cases

In a normal scenario, the reader connects fine, but following are the cases which require special handling at the time of connection.

The following example shows a connection handled under try-catch block.

```
try
{
    // Establish connection to the RFID Reader
    reader.Connect();
}
catch (Exception e)
{
    Debug.Print(e.Message);
}
```

Region Is Not Configured

If the region is not configured an exception ERROR_REGION_NOT_CONFIGURED is given.

Then the caller chooses the operation regulatory region and sets the region with required configurations, as shown below:

```
private void GetRegionInfo()
{
    try
    {
        RegulatoryConfig regConfig = reader.Configurations.RegulatoryConfig;
        Debug.WriteLine("Config.RegulatoryConfig.Region : " + regConfig.Region);
    }
    catch (Exception ex)
    {
        if (ex.Message == reader.Configurations.ERROR_REGION_NOT_CONFIGURED)
            ConfigureDefaultRegion();
        else
            Debug.WriteLine(ex.Message + Environment.NewLine);
    }
}
private void ConfigureDefaultRegion()
{
    Debug.WriteLine("Region: Not Configured. Configuring as USA");
    RegulatoryConfig config = new RegulatoryConfig();
    config.Region = "USA";
    reader.Configurations.RegulatoryConfig = config;
}
```

Disconnect

When the application is done with the connection and operations on the RFID reader, call the following method to close the connection.

```
// Disconnects reader
reader.Disconnect();
```

Reader Capabilities

The capabilities (or Read-Only properties) of the reader are listed below.

General Capabilities

- Model Name.
- Serial Number.
- Manufacturer Name.
- Manufacture Date.
- Number of antennas supported.
- Is Tag Event Reporting Supported - indicates the reader's ability to report tag visibility state changes (New Tag, Tag Invisible, or Tag Visibility Changed).
- Is Tag Locationing Supported - indicates the reader's ability to locate a tag.
- Is Hopping Enabled - Not supported.

Gen2 Capabilities

- Block Erase - Supported
- Block Write - Supported
- State Aware Singulation - Supported
- Maximum Number of Operation in Access Sequence - Not supported
- Maximum Pre-filters allowable per antenna - Not supported
- RF Modes - Not supported.

Regulatory Capabilities

- Country Code
- Communication StandardRegion
- Hopping
- Enable Channels.

For setting/getting Region, see Region Is Not Configured on page 11.

Retrieving Reader Capabilities

```
    //Get Reader capabilities
    Console.WriteLine("ModelName: " + reader.Capabilities.ModelName);
    Console.WriteLine("SerialNumber: " + reader.Capabilities.SerialNumber);
    Console.WriteLine("Manufacture Name: " + reader.Capabilities.ManufactureName);
    Console.WriteLine("Manufacturing Date: " + reader.Capabilities.ManufacturingDate);
    Console.WriteLine("Tag Event Reporting 
Supported:"+reader.Capabilities.IsTagEventReportingSupported);
    Console.WriteLine("Tag Locationing Supported: " + 
reader.Capabilities.IsTagLocationingSupported);
    Console.WriteLine("Hopping Enabled: " + reader.Capabilities.IsHoppingEnabled);
```

Configuring the Reader

Antenna Specific Configuration

The reader.Configurations class contains the Antennas as object. The individual antenna can be accessed and configured using the index.

The reader.Configurations.Antennas[antennaID].Configuration Properties is used to set the antenna configuration to individual antenna.

The antenna configuration comprised of Transmit Power Index, Receive Sensitivity Index and Transmit Frequency Index.

Set/Get individual antenna configuration settings as follows:

```
    ushort PowerVal = 270;
    ushort curAntennaID = 0;
    AntennaConfiguration antConfig =
reader.Configurations.Antennas[curAntennaID].Configuration;
    antConfig.TransmitPowerIndex = PowerVal;
    reader.Configurations.Antennas[curAntennaID].Configuration = antConfig;
    Console.WriteLine("Set TransmitPowerIndex = " +
antConfig.TransmitPowerIndex.ToString());
```

Singulation Control

The property SingulationControl sets/gets the current settings of singulation control from the reader, for the given Antenna ID.

The following settings can be configured:

- Session - session number to use for inventory operation.
- Tag Population - an estimate of the tag population in view of the RF field of the antenna.
- Tag Transit Time - an estimate of the time a tag typically remains in the RF field.

```
    ushort curAntennaID = 0;
    // Get Singulation
    SingulationControl singulationControl =
reader.Configurations.Antennas[curAntennaID].SingulationControl;
    Console.WriteLine("Session : "+singulationControl.Session.ToString());
    Console.WriteLine("Population : "+singulationControl.TagPopulation.ToString());
    Console.WriteLine("TagTransitTime :"+singulationControl.TagTransitTime.ToString());

    // Set Singulation
    singulationControl.Session = SESSION.SESSION_S1;
    singulationControl.TagPopulation = 30;
    reader.Configurations.Antennas[curAntennaID].SingulationControl = singulationControl;
Console.WriteLine("SetSingulation :  Session  = [" + singulationControl.Session + "]");
    Console.WriteLine("SetSingulation : TagPopulation:"+
singulationControl.TagPopulation.ToString());
```

Tag Report Configuration

The SDK provides an ability to configure a set of fields to be reported in a response to an operation by a specific active RFID reader.

Supported fields that might be reported are listed below.

- First seen time
- Last seen time
- PC value
- RSSI value
- Phase value
- Channel index
- Tag seen count.

The reader.Configurations.ReportConfig class can be used to retrieve and sets the tag report parameters from the reader.

Dynamic Power Management Configuration

The SDK provides a way to configure the reader to operate in dynamic power mode. The dynamic power state can be switched to be either on or off.

```
// set Dynamic power state on
reader.Configurations.DynamicPowerConfig.setDPOState(DYNAMIC_POWER_OPTIMIZATION.ENABLE);
// set Dynamic power state off
reader.Configurations.DynamicPowerConfig.setDPOState(DYNAMIC_POWER_OPTIMIZATION.DISABLE);
```

Regulatory Configuration

The SDK supports managing of regulatory related parameters of a specific active RFID reader.

Regulatory configuration options are listed below.

- Code of selected region
- Hopping
- Set of enabled channels.

A set of enabled channels includes only such channels that are supported in the selected region. If hopping configuration is not allowed for the selected regions, a set of enabled channels is not specified.

Regulatory parameters could be retrieved and set via RegulatoryConfig property accordingly. The region information is retrieved using Region property. The following example demonstrates retrieving of current regulatory settings and configuring the RFID reader to operate in one of supported regions.

```
// Get Regulatory Config
RegulatoryConfig regConfig = reader.Configurations.RegulatoryConfig;
Console.WriteLine("Config.RegulatoryConfig.Region : " + regConfig.Region);
Console.WriteLine("Config.RegulatoryConfig.Hopping : " + regConfig.Hopping);
string[] enabledChannels = regConfig.EnabledChannels;

// Set Regulatory Config
RegulatoryConfig config = new RegulatoryConfig();
config.Region = "USA";
reader.Configurations.RegulatoryConfig = config;
```

Saving Configuration

Various parameters of a specific RFID reader configured via SDK are lost after the next power down. The SDK provides an ability to save a persistent configuration of RFID reader. The SaveConfig method can be used to make the current configuration persistent over power down and power up cycles. The following example demonstrates utilization of mentioned method.

```
// Saving the configuration
reader.Configurations.SaveConfig();
```

Reset Configuration to Factory Defaults

The SDK provides a way to reset the RFD8500 reader to the factory default settings. The ResetFactoryDefaults method can be used to attain this functionality. Once this method is called, all the reader settings like events, singulation control, etc. will revert to default values and the RFD8500 reboots. A connected application shall lose connectivity to the reader and must connect back again and is required to redo the basic steps for initializing the reader. For mobile device applications after reset to defaults, you will have to manually pair the scanner using BT Explorer. The following example demonstrates utilization of mentioned method.

```
// Resetting the configuration
reader.ResetFactoryDefaults();
```

Managing Events

The application can register for one or more events, to be notified when it occurs. There are several types of events. Table 2 lists the events supported.

Table 2 Supported Events

Event	Description
`readerWatcher.ReaderConnected`	Event notifying connection from the Reader.
`readerWatcher.ReaderDisconnected`	Event notifying disconnection from the Reader. The application can call connect method periodically to attempt reconnection or call disconnect method to cleanup and exit.
`readerWatcher.ReaderAppeared`	Event notified when reader paired.
`readerWatcher.ReaderDisappeared`	Event notified when reader unpaired.
`reader.Inventory.TagDataReceived`	Tag Data received event.
`reader.Inventory.InventoryStarted`	Inventory operation started. In case of periodic trigger, this event is triggered for each period.
`reader.Inventory.InventoryStopped`	Inventory operation has stopped. In case of periodic trigger this event is triggered for each period.
`reader.Inventory.InventorySessionSummary`	Event generated when operation end summary has been generated. The data associated with the event contains total rounds, total number of tags and total time in micro secs.
`reader.BatteryStatusNotification`	Events notifying different levels of battery, state of the battery, if charging or discharging.

Table 2 Supported Events (Continued)

Event	Description
`reader.PowerStatusNotification`	Events which notify the different power states of the reader device. The event data contains cause, voltage, current and power.
`reader.TemperatureStatusNotification`	When temperature reaches threshold level, this event is generated. The event data contains source name (PA/Ambient).
`reader.Inventory.BatchMode`	Event generated when batch tag read operation is in progress.

Registering for Tag Data Notification

```
    // registering for read tag data notification
    reader.Inventory.TagDataReceived += Inventory_TagDataReceived;

private void Inventory_TagDataReceived(object sender, TagDataReceivedEventArgs e)
    {
        Console.WriteLine("Events_ReadNotify --------");
        Console.WriteLine("Tag ID:" + e.EPCId);
        Console.WriteLine("Tag Seen Count:" + e.TagSeenCount);
        Console.WriteLine("RSSI" + e.RSSI);
    }
```

Device Status Related Events

Device status, like battery, power, and temperature, is obtained through events after initiating the reader.Configurations.GetDeviceStatus method.

Response to the above method comes as battery event, power event and temperature event according to the set boolean value in the respective parameters. The following is an example of how to get these events.

```
  reader.BatteryStatusNotification += Reader_BatteryStatusNotification;
  reader.TemperatureStatusNotification += Reader_TemperatureStatusNotification;
  reader.PowerStatusNotification += Reader_PowerStatusNotification;

  bool battery = true;
  bool power = true;
  bool temperature = true;
  reader.Configurations.GetDeviceStatus(battery, power, temperature);


  private void Reader_BatteryStatusNotification(object sender,
BatteryStatusNotificationReceivedEventArgs e)
  {
     //Handle battery event notification
  }
```

(continued on next page)

```
    private void Reader_PowerStatusNotification(object sender,
PowerStatusNotificationReceivedEventArgs e)
    {
        //Handle power event notification
    }

    private void Reader_TemperatureStatusNotification(object sender,
TemperatureStatusNotificationReceivedEventArgs e)
    {
        //Handle temperature event notification
    }
```

Basic Operations

Tag Storage Settings

This section covers the basic/simple operations that an application would need to be performed on an RFID reader which includes inventory and single tag access operations.

Each tag has a set of associated information along with it. During the Inventory operation the reader reports the EPC-ID of the tag where as during the Read-Access operation the requested Memory Bank Data is also reported apart from EPC-ID. In either case, there is additional information like PC-bits, RSSI, last time seen, tag seen count, etc. that is available for each tag. This information is reported to the application as TagData for each tag reported by the reader. Applications can also choose to enable/disable reporting certain fields in TAG_DATA. Disabling certain fields can sometimes improve the performance as the reader and the SDK are not processing that information.

Following are a few use-cases that get tags from the reader.

Reading Tag Data from Event

A simple continuous inventory operation reads all tags in the field of view of all antennas of the connected RFID reader. The start and stop trigger for the inventory is the default (i.e., start immediately when reader.Inventory.Perform is called, and stop immediately when reader.Inventory.Stop is called).

```
// registering for read tag data notification
reader.Inventory.TagDataReceived += Inventory_TagDataReceived;

// perform simple inventory reader
reader.Inventory.Perform();

// Keep getting tags in the TagDataReceived event if registered
Thread.Sleep(5000); // Wait for 5 seconds for tags to read

// stop the inventory
reader.Inventory.Stop();
private void Inventory_TagDataReceived(object sender, TagDataReceivedEventArgs e)
{
    Console.WriteLine("Events_ReadNotify --------");
    Console.WriteLine("Tag ID:" + e.EPCId);
    Console.WriteLine("Tag Seen Count:" + e.TagSeenCount);
    Console.WriteLine("RSSI" + e.RSSI);
}
```

Reading Tag Data from Queue

The GetNextTagDataReceived() method is used to read tag data from internal queue. This is a blocking method that retrieves oldest ITagData buffered in the internal queue. If no tag data is present, the method blocks and waits until tag data is received.

If a timeout is specified as a parameter the method blocks and waits for the specified amount of time, for tag data (ITagData) to appear in the internal queue and returns the corresponding value.

To enable tag data to be received from internal queue, update the App.Config xml <appSettings> section as follows:

```
    <!--Support tag data queuing for Win Mobile -->
    <add key="ZetiResponseDispatcherAssembly"
value="Symbol.RFID.SDK.Domain.Reader.Infrastructure.dll"/>
    <add key="ZetiTagDataDispatcher"
value="Symbol.RFID.SDK.Domain.Reader.Infrastructure.ZetiTagDataQueuingDispatcher"/>
```

NOTE: Tag data is not returned in the form of an event when you enable the setting above.

To enable tag data to be received as events use the following App.Config xml `<appSettings>` section:

```
    <!--Support tag data dispatching via events for Win 7 and above or powerful WM devices
-->
    <add key="ZetiResponseDispatcherAssembly"
value="Symbol.RFID.SDK.Domain.Reader.Infrastructure.dll"/>
    <add key="ZetiTagDataDispatcher"
value="Symbol.RFID.SDK.Domain.Reader.Infrastructure.ZetiTagDataDispatcher"/>
```

Simple Access Operations

Tag Access operations can be performed on a specific tag or can be applied on tags that match a specific Access-Filter. If no Access-Filter is specified the Access Operation is performed on all tags in the field of view of chosen antennas. This section covers the Simple Tag Access operation on a specific tag which could be in the field of view of any of the antennas of the connected RFID reader.

Dynamic power optimization should be disabled before any access operations.

```
// set Dynamic power state off
reader.Configurations.DynamicPowerConfig.setDPOState(DYNAMIC_POWER_OPTIMIZATION.DISABLE);
```

Read

The application can call method `reader.AccessOperations.TagRead.Read()` to read data from a specific memory bank.

Write

The application can call method `reader.AccessOperations.TagWrite.Write()` to write data to a specific memory bank. The response is returned as a Tagdata from where number of words can be retrieved.

Lock

The application can call method `reader.AccessOperations.TagLock.Lock()` to perform a lock operation on one or more memory banks with specific privileges.

Kill

The application can call method `reader.AccessOperations.TagKill.Kill()` to kill a tag.

Tag Locationing

This feature is supported only on hand-held readers and is useful to locate a specific tag in the field of view of the reader's antenna. The default locationing algorithm supported on the reader can perform locationing only on a single antenna. reader.TagLocate.Perform(string epc) can be used to start locating a tag, and reader.TagLocate.Stop() to stop the locationing operation. The result of locationing of a tag is reported as reader.TagLocate.ProximityPercentReceived event and ProximityPercent in ProximityPercentReceivedEventArgs gives the relative distance of the tag from the reader antenna.

Advance Operations

Using Triggers

Triggers are the conditions that should be satisfied to start or stop an operation (Inventory). This information can be specified using TriggerInfo class.

Use reader.Configurations.TriggerInfo.StartTrigger and reader.Configurations.TriggerInfo.StopTrigger methods to set triggers on the reader.

The following are some use-cases of using TRIGGER_INFO.

- Periodic Inventory: Start inventory at a specified time for a specified duration repeatedly.

```
TriggerInfo triggerInfo = reader.Configurations.TriggerInfo;
// start inventory every 3 seconds
triggerInfo.StartTrigger.Type=START_TRIGGER_TYPE.START_TRIGGER_TYPE_PERIODIC;
triggerInfo.StartTrigger.Periodic.Period=3000
// stop trigger
triggerInfo.StopTrigger.Type=STOP_TRIGGER_TYPE.STOP_TRIGGER_TYPE_DURATION;
triggerInfo.StopTrigger.Duration=5000; // stop after 5 seconds
reader.Configurations.TriggerInfo = triggerInfo;
```

- Perform 'n' Rounds of Inventory with a timeout: Start condition could be any; Stop condition is to perform 'n' rounds of inventory and then stop or stop inventory after the specified timeout.

```
    TriggerInfo triggerInfo = reader.Configurations.TriggerInfo;
    // start inventory immediate
    triggerInfo.StartTrigger.Type=START_TRIGGER_TYPE.START_TRIGGER_TYPE_IMMEDIATE;
    // stop trigger

triggerInfo.StopTrigger.Type=STOP_TRIGGER_TYPE.STOP_TRIGGER_TYPE_N_ATTEMPTS_WITH_TIMEOUT;
    triggerInfo.StopTrigger.NumAttempts.N=3; // perform 3 rounds of inventory
    triggerInfo.StopTrigger.NumAttempts.Timeout=3000; // timeout after 3 seconds
    reader.Configurations.TriggerInfo = triggerInfo;
```

- Read 'n' tags with a timeout: Start condition could be any; Stop condition is to stop after reading 'n' tags or stop inventory after the specified timeout.

```
    TriggerInfo triggerInfo = reader.Configurations.TriggerInfo;
    // start inventory immediate
    triggerInfo.StartTrigger.Type = START_TRIGGER_TYPE.START_TRIGGER_TYPE_IMMEDIATE;
    // stop trigger
    triggerInfo.StopTrigger.Type =
STOP_TRIGGER_TYPE.STOP_TRIGGER_TYPE_TAG_OBSERVATION_WITH_TIMEOUT;
    triggerInfo.StopTrigger.TagObservation.N= 5; // number of tag observations
    triggerInfo.StopTrigger.TagObservation.Timeout = 10000; // timeout after 10 seconds
    reader.Configurations.TriggerInfo = triggerInfo;
```

- Inventory based on hand-held trigger: Start inventory when the reader hand-held trigger button is pulled, and stop inventory when the hand-held trigger button is released or subject to timeout.

```
    TriggerInfo triggerInfo = reader.Configurations.TriggerInfo;
    // start inventory immediate
    triggerInfo.StartTrigger.Type = START_TRIGGER_TYPE.START_TRIGGER_TYPE_HANDHELD;
    triggerInfo.StartTrigger.Handheld.HandheldEvent =
          HANDHELD_TRIGGER_EVENT_TYPE.HANDHELD_TRIGGER_PRESSED; // number of tag
observations
    triggerInfo.StartTrigger.Handheld.Timeout = 10000; // timeout after 10 seconds
    // stop trigger
    triggerInfo.StopTrigger.Type =
STOP_TRIGGER_TYPE.STOP_TRIGGER_TYPE_HANDHELD_WITH_TIMEOUT;
    triggerInfo.StopTrigger.Handheld.HandheldEvent=
        HANDHELD_TRIGGER_EVENT_TYPE.HANDHELD_TRIGGER_RELEASED; // number of tag
observations
    triggerInfo.StopTrigger.Handheld.Timeout = 10000; // timeout after 10 seconds
reader.Configurations.TriggerInfo = triggerInfo;
```

Using Beeper

Use the reader.Configurations.BeeperVolume property to turn the beeper on/off, and set volume.

Get beeper setting example:

```
    BEEPER_VOLUME beeperVolume = reader.Configurations.BeeperVolume;
    string strBeeperVolume = "";
    switch (beeperVolume)
    {
        case BEEPER_VOLUME.HIGH_BEEP:
                strBeeperVolume = "HIGH_BEEP";
                break;
        case BEEPER_VOLUME.MEDIUM_BEEP:
                strBeeperVolume = "MEDIUM_BEEP";
                break;
        case BEEPER_VOLUME.LOW_BEEP:
                strBeeperVolume = "LOW_BEEP";
                break;
        case BEEPER_VOLUME.QUIET_BEEP:  // beeper sound off
                strBeeperVolume = "QUIET_BEEP";
                break;
    }
    Console.WriteLine("GetBeeperVolume = " + strBeeperVolume);
```

Set beeper example:

```
    //Set beeper volume high
    reader.Configurations.BeeperVolume = BEEPER_VOLUME.HIGH_BEEP;
```

Batch Mode

When the RFD8500 reader is configured to operate in batch mode, it is capable of reading RFID tag data without being connected to a host device. The reader.Configurations.BatchModeConfig property can be used to configure Batch Mode as follows:

```
    reader.Configurations.BatchModeConfig = BATCH_MODE.ENABLE
```

Batch Mode can be configured to one of the modes listed in Table 3.

Table 3 Batch Modes

Mode	Description
`BATCH_MODE.DISABLE`	Tags are reported in real time as they are inventoried. No data is preserved if the application disconnects.
`BATCH_MODE.ENABLE`	Tags are stored in an internal database maintained in the reader, and are not returned to host in real time. While in batch mode, the reader will continue to perform inventory even if the reader is disconnected from the host. Upon re-connection, the ReadSessionBatchModeEventArgs" event will be raised indicating that the inventory is in progress. In order to retrieve the stored tags, inventory must be stopped by calling reader.Inventory.Stop(), and the reader.Inventory.GetBatchedTags() method must be called to get the stored tag Data.
`BATCH_MODE.AUTO`	Tags are reported in real time while the application that initiated performing inventory is still connected. If the reader is disconnected, the tag data is stored in an internal database maintained in the reader Upon re-connection, the "ReadSessionBatchModeEventArgs" event will be raised indicating that the inventory is in progress. In order to retrieve the stored tags, inventory must be stopped by calling reader.Inventory.Stop(), and the reader.Inventory.GetBatchedTags() method must be called to get the stored tag Data.

To clear stored batched tags in the reader's internal database, call the reader.Inventory.PurgeTags() method.

Using Pre-Filters

Pre-filters are the same as the Select command of C1G2 specification. Once applied, pre-filters are applied prior to Inventory and Access operations.

Singulation

Singulation refers to the method of identifying an individual Tag in a multiple-Tag environment.

In order to filter tags that match a specific condition, it is necessary to use the tag-sessions and their states (setting the tags to different states based on match criteria - reader.PreFilters.ConfiguredFilters) so that while performing inventory, tags can be instructed to participate (singulation - reader.Config.PreFilters.ConfiguredFilters[filterIndex].IsEnable) or not participate in the inventory based on their states.

Sessions and Inventoried Flags

Tags provide four sessions (denoted S0, S1, S2, and S3) and maintain an independent inventoried flag for each session. Each of the four inventoried flags has two values, denoted A and B. These inventoried flag of each session can be set to A or B based on match criteria using method reader.ConfiguredFilters[filterIndex].Action

Selected Flag

Tags provide a selected flag, SL, which can be asserted or deasserted based on match criteria using reader.ConfiguredFilters[filterIndex].Action

State-Aware Singulation

In state-aware singulation the application can specify detailed controls for singulation: Action and Target.

Action indicates whether matching Tags assert or deassert SL (Selected Flag), or set their inventoried flag to A or to B. Tags conforming to the match criteria specified using the reader.ConfiguredFilters[filterIndex].Action are considered matching and the remaining are non-matching.

Target indicates whether to modify a tag's SL flag or its inventoried flag, and in the case of inventoried it further specifies one of four sessions.

Applying Pre-Filters

Follow these steps to use pre-filters.

1. Add pre-filters.

   The application can update pre-filters using the reader.ConfiguredFilters list to add and remove pre-filters.

2. Set appropriate singulation controls.

   Now that the pre-filters are set (i.e., tags are classified into matching or non-matching criteria), the application needs to specify which tags should participate in the inventory using reader.Configurations.Antennas[curAntennaID].SingulationControl.

3. Perform Inventory or Access operation.

   Inventory or Access operation when performed after setting pre-filters, use the tags filtered out of pre-filters for their operation.

Exceptions

The Zebra RFID Windows SDK throws standard .Net exceptions. All API calls should be under try-catch block to catch exceptions thrown while performing API calls.

```
try
{
    // Establish connection to the RFID Reader
    reader.Connect();
}
catch (Exception e)
{
    Debug.Print(e.Message);
}
```

Creating, Building, and Running Projects

Introduction

This chapter provides step-by-step instructions to import the RFID SDK module and build Windows applications (with Microsoft .net 4.5/Compact framework) to work with the RFD8500 reader.

NOTE: To build a Microsoft .net 4.5 application for Window 7 use Visual Studio 2015.

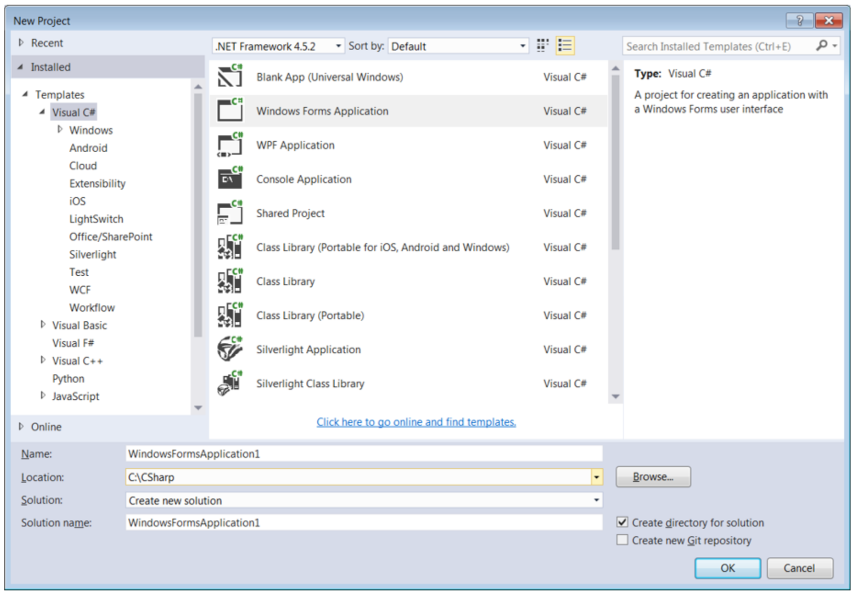
For .Net Compact Framework MC55 mobile applications use Visual Studio 2008 (and the Windows Mobile 6 SDK).

Creating a Windows Project

To create a C# Windows project in Visual Studio 2015:

1. Start Visual Studio 2015.
2. Select File > New > Project > Visual C#.
3. Create a new Windows Forms Application project and follow the on-screen steps in Visual Studio.

Figure 1 New Project Window



New Project

.NET Framework 4.5.2 Sort by: Default

Search Installed Templates (Ctrl+E)

Blank App (Universal Windows)	Visual C#
Windows Forms Application	Visual C#
WPF Application	Visual C#
Console Application	Visual C#
Shared Project	Visual C#
Class Library (Portable for iOS, Android and Windows)	Visual C#
Class Library	Visual C#
Class Library (Portable)	Visual C#
Silverlight Application	Visual C#
Silverlight Class Library	Visual C#

Type: Visual C#

A project for creating an application with a Windows Forms user interface

Click here to go online and find templates.

Name: WindowsFormsApplication1

Location: C:\CSharp

Solution: Create new solution

Solution name: WindowsFormsApplication1

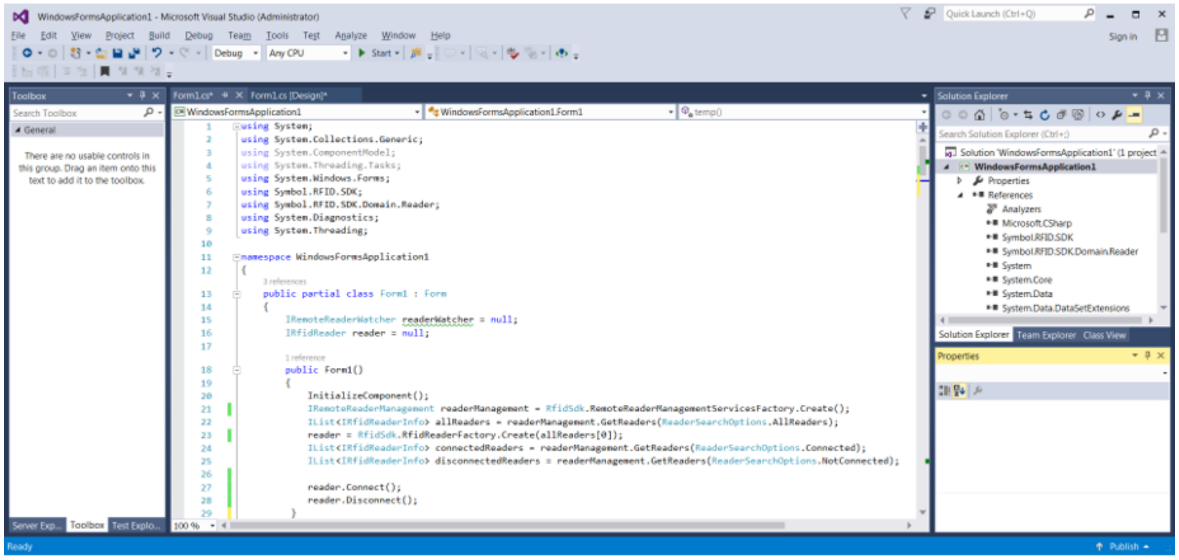
[✓] Create directory for solution

[ ] Create new Git repository

4. Add a reference to the Symbol.RFID.SDK and Symbol.RFID.SDK.Domain.Reader assemblies/DLLs from RFID SDK binaries.

5. Import the Symbol.RFID.SDK and Symbol.RFID.SDK.Domain.Reader namespace/classes.

Figure 2 Import Window



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Threading.Tasks;
using System.Windows.Forms;
using Symbol.RFID.SDK;
using Symbol.RFID.SDK.Domain.Reader;
using System.Diagnostics;
using System.Threading;

namespace WindowsFormsApplication1
{
    3 references
    public partial class Form1 : Form
    {
        IRemoteReaderWatcher readerWatcher = null;
        IRfidReader reader = null;

        1 reference
        public Form1()
        {
            InitializeComponent();
            IRemoteReaderManagement readerManagement = RfidSdk.RemoteReaderManagementServicesFactory.Create();
            IList<IRfidReaderInfo> allReaders = readerManagement.GetReaders(ReaderSearchOptions.AllReaders);
            reader = RfidSdk.RfidReaderFactory.Create(allReaders[0]);
            IList<IRfidReaderInfo> connectedReaders = readerManagement.GetReaders(ReaderSearchOptions.Connected);
            IList<IRfidReaderInfo> disconnectedReaders = readerManagement.GetReaders(ReaderSearchOptions.NotConnected);

            reader.Connect();
            reader.Disconnect();
        }
```

Building and Running a Windows Project

To build and run a project:

1. Before building/running a project, ensure the assemblies listed below are in the target application folder. The target application folder is the location of the compiled application (for example, \bin\Debug\).
   - InTheHand.Net.Personal.dll
   - RFIDCommandLib.dll
   - Symbol.Extensions.Compatibility.dll
   - Symbol.RFID.SDK.Connectivity.Windows.dll
   - Symbol.RFID.SDK.Discovery.Windows.dll
   - Symbol.RFID.SDK.dll
   - Symbol.RFID.SDK.Domain.Reader.dll
   - Symbol.RFID.SDK.Domain.Reader.Infrastructure.dll
   - Symbol.RFID.SDK.Domain.Reader.Infrastructure.Management.dll
   - TaskParallel.dll
2. Add the following settings to the App.Config file (ensure the correct paths are specified for the target assemblies).

```
    <configuration>
        <appSettings>
          <add key="RemoteReaderAssembly" value="Symbol.RFID.SDK.Domain.Reader.dll"/>
          <add key="RemoteReaderService"
value="Symbol.RFID.SDK.Domain.Reader.ZetiRfidReader"/>
```

```
          <add key="RemoteReaderConnectionAssembly"
value="Symbol.RFID.SDK.Connectivity.Windows.dll"/>
          <add key="RemoteReaderConnectionService"
value="Symbol.RFID.SDK.Connectivity.Windows.SocketDeviceConnection"/>

          <add key="RemoteReaderDiscoveryServiceAssembly"
value="Symbol.RFID.SDK.Discovery.Windows.dll"/>
          <add key="RemoteReaderDiscoveryService"
value="Symbol.RFID.SDK.Discovery.Windows.ReaderWatcher"/>

          <add key="RemoteReaderInfrastructureServiceAssembly"
value="Symbol.RFID.SDK.Domain.Reader.Infrastructure.dll"/>
          <add key="RemoteReaderInfrastructureService"
              value="Symbol.RFID.SDK.Domain.Reader.Infrastructure.ZetiRfidReaderAdapter"/>

          <add key="RemoteReaderManagementServiceAssembly"
              value="Symbol.RFID.SDK.Domain.Reader.Infrastructure.Management.dll"/>
          <add key="RemoteReaderManagementService"

value="Symbol.RFID.SDK.Domain.Reader.Infrastructure.Management.RemoteReaderManagement"/>
        </appSettings>
    </configuration>
```

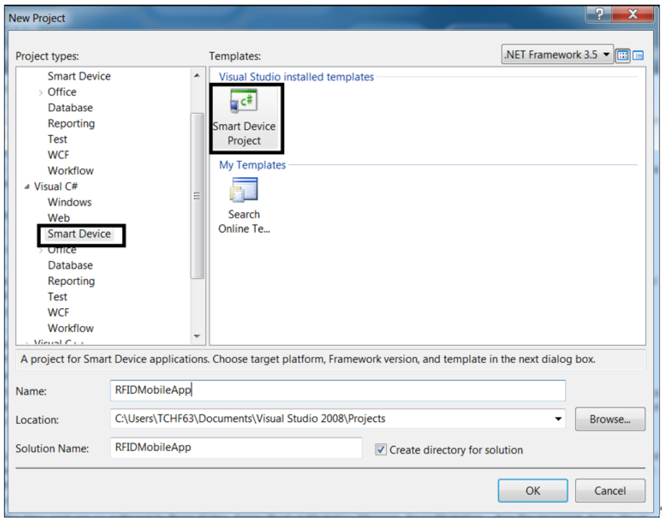
NOTE: Ensure the specified DLLs are present in the target application folder and add the <appSettings> section above to the App.Config xml configuration file of the project. The settings above are for Desktop Windows RFID reader connections using Bluetooth sockets.

Creating a Windows Mobile Project

To create a C# Windows mobile project (.Net Compact Framework) in Visual Studio 2008:

1. Start Visual Studio 2008.
2. Select File > New > Project > Visual C#.
3. Create a new Smart Device project and follow the on-screen steps in Visual Studio.

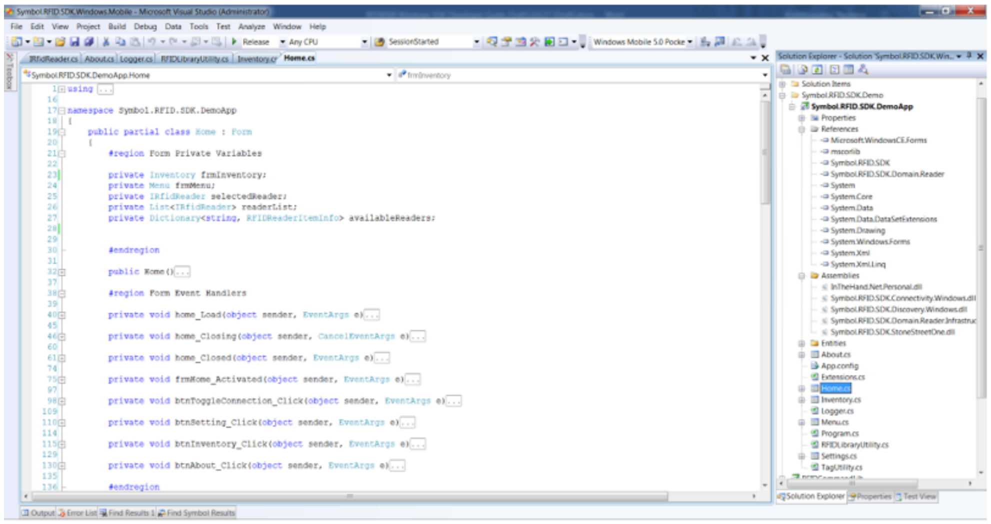
Figure 3 Smart Device Project Window



A project for Smart Device applications. Choose target platform, Framework version, and template in the next dialog box.

Name: RFIDMobileApp
Location: C:\Users\TCHF63\Documents\Visual Studio 2008\Projects
Solution Name: RFIDMobileApp
☑ Create directory for solution

Figure 4 Assembly Configuration for RFID SDK Demo Project


Symbol.RFID.SDK.Windows.Mobile - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Data Tools Test Analyze Window Help
Release
Any CPU
SessionStarted
Windows Mobile 5.0 Pocke
IRfidReader.cs About.cs Logger.cs RFIDLibraryUtility.cs Inventory.cs Home.cs
Symbol.RFID.SDK.DemoApp.Home
frmInventory
Output Error List Find Results 1 Find Symbol Results
Solution Explorer - Solution 'Symbol.RFID.SDK.Win...
Solution Explorer Properties Test View

```
using ...

namespace Symbol.RFID.SDK.DemoApp
{
    public partial class Home : Form
    {
        #region Form Private Variables

        private Inventory frmInventory;
        private Menu frmMenu;
        private IRfidReader selectedReader;
        private List<IRfidReader> readerList;
        private Dictionary<string, RFIDReaderItemInfo> availableReaders;


        #endregion

        public Home()...

        #region Form Event Handlers

        private void home_Load(object sender, EventArgs e)...

        private void home_Closing(object sender, CancelEventArgs e)...

        private void home_Closed(object sender, EventArgs e)...

        private void frmHome_Activated(object sender, EventArgs e)...

        private void btnToggleConnection_Click(object sender, EventArgs e)...

        private void btnSetting_Click(object sender, EventArgs e)...

        private void btnInventory_Click(object sender, EventArgs e)...

        private void btnAbout_Click(object sender, EventArgs e)...

        #endregion
```

```
Solution Items
Symbol.RFID.SDK.Demo
  Symbol.RFID.SDK.DemoApp
    Properties
    References
      Microsoft.WindowsCE.Forms
      mscorlib
      Symbol.RFID.SDK
      Symbol.RFID.SDK.Domain.Reader
      System
      System.Core
      System.Data
      System.Data.DataSetExtensions
      System.Drawing
      System.Windows.Forms
      System.Xml
      System.Xml.Linq
    Assemblies
      InTheHand.Net.Personal.dll
      Symbol.RFID.SDK.Connectivity.Windows.dll
      Symbol.RFID.SDK.Discovery.Windows.dll
      Symbol.RFID.SDK.Domain.Reader.Infrastruc
      Symbol.RFID.SDK.StoneStreetOne.dll
    Entities
    About.cs
    App.config
    Extensions.cs
    Home.cs
    Inventory.cs
    Logger.cs
    Menu.cs
    Program.cs
    RFIDLibraryUtility.cs
    Settings.cs
    TagUtility.cs
```

Building and Running a Windows Mobile Project

To build and run a mobile project:

1. Before building/running a Windows mobile project, ensure the .Net CF specific RFID SDK assemblies listed below are in the target application folder. The target application folder is the location of the compiled application (for example, \bin\Debug\Assemblies).

   - InTheHand.Net.Personal.dll
   - Symbol.RFID.SDK.Connectivity.Windows.dll
   - Symbol.RFID.SDK.Discovery.Windows.dll
   - Symbol.RFID.SDK.Domain.Reader.Infrastructure.Management.dll
   - Symbol.RFID.SDK.StoneStreetOne.dll

   In addition, manually add references to the following DLLs in the target Visual Studio project:

- Symbol.RFID.SDK.dll
- Symbol.RFID.SDK.Domain.Reader.dll

For Compact Framework applications built with Visual Studio 2008 (for the MC55) use the serial connection with StoneStreet One Bluetooth stack. Only StoneStreet One Bluetooth stack is supported for Mobile Compact Framework applications. Add the following settings to the App.Config file and ensure the correct paths are specified for the target assemblies.

```
    <configuration>
      <appSettings>
        <add key="RemoteReaderAssembly" value="Symbol.RFID.SDK.Domain.Reader.dll"/>
        <add key="RemoteReaderService" value="Symbol.RFID.SDK.Domain.Reader.ZetiRfidReader"/>

        <add key="RemoteReaderConnectionAssembly"
value="Assemblies\Symbol.RFID.SDK.Connectivity.Windows.dll"/>
        <add key="RemoteReaderConnectionService"
value="Symbol.RFID.SDK.Connectivity.Windows.SerialPortDeviceConnection"/>

        <add key="RemoteReaderDiscoveryServiceAssembly"
value="Assemblies\Symbol.RFID.SDK.Discovery.Windows.dll"/>
        <add key="RemoteReaderDiscoveryService"
value="Symbol.RFID.SDK.Discovery.Windows.ReaderWatcherSS"/>

        <add key="RemoteReaderInfrastructureServiceAssembly"
value="Symbol.RFID.SDK.Domain.Reader.Infrastructure.dll"/>
        <add key="RemoteReaderInfrastructureService"
value="Symbol.RFID.SDK.Domain.Reader.Infrastructure.ZetiRfidReaderAdapter"/>

        <add key="RemoteReaderManagementServiceAssembly"
    value="Assemblies\Symbol.RFID.SDK.Domain.Reader.Infrastructure.Management.dll"/>
        <add key="RemoteReaderManagementService"
value="Symbol.RFID.SDK.Domain.Reader.Infrastructure.Management.RemoteReaderManagementSS"/>

        <add key="ZetiResponseDispatcherAssembly"
value="Symbol.RFID.SDK.Domain.Reader.Infrastructure.dll"/>
        <add key="ZetiTagDataDispatcher"
value="Symbol.RFID.SDK.Domain.Reader.Infrastructure.ZetiTagDataQueuingDispatcher"/>
      </appSettings>
    </configuration>
```

Pairing with Bluetooth

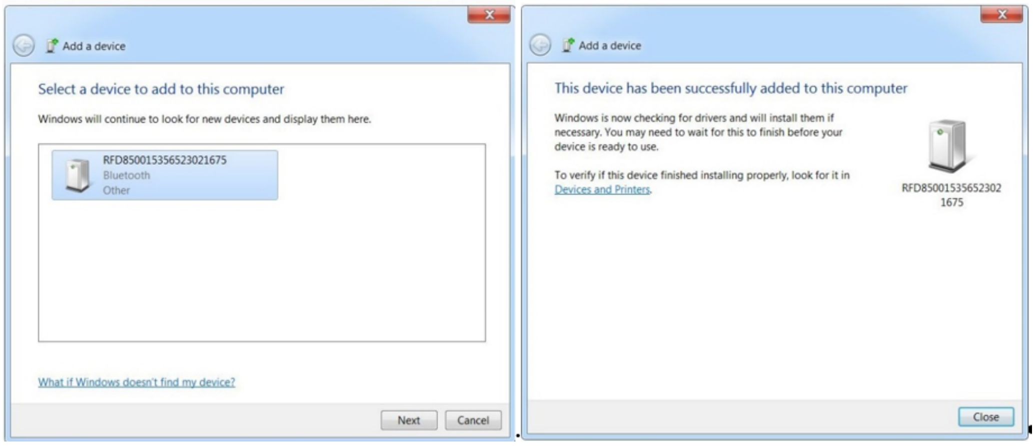
Pairing with a Personal Computer

1. If the BT LED is not blinking, press the **BT** button for one second to make the RFD8500 discoverable (the BT LED starts blinking when in discoverable mode).
2. From the Start menu, select Device and Printers.
3. Select Add a device.
4. Select the device and click **Next**. When the BT LED starts blinking rapidly press the trigger within 25 seconds to acknowledge pairing.
5. Select **Close** to complete the pairing process.

Pairing with a MC55 mobile device

1. Go to BTExplorer and select Menu > New Connection.
2. Select **Next**.
3. Select Menu > Discover Devices.
4. If the BT LED is not blinking, press the **BT** button for one second to make the RFD8500 discoverable (the BT LED starts blinking when in discoverable mode).
5. When the device appears in the list, tap the device name and select **Next**.
6. Select RFID Serial Port and then select **Next**.
7. Select **Next**.
8. Select **Connect**.
9. Select **OK**.
10. Select **Yes** to confirm connection. When the BT LED starts to blink rapidly, press the RFD8500 trigger within 25 seconds to accept the pairing request.
11. When your device is in the connected state, select it and disconnect. This retains only the paired state.

Figure 5 Pairing Bluetooth Device on Windows



Select a device to add to this computer

Windows will continue to look for new devices and display them here.

RFD850015356523021675
Bluetooth
Other

This device has been successfully added to this computer

Windows is now checking for drivers and will install them if necessary. You may need to wait for this to finish before your device is ready to use.

To verify if this device finished installing properly, look for it in Devices and Printers.

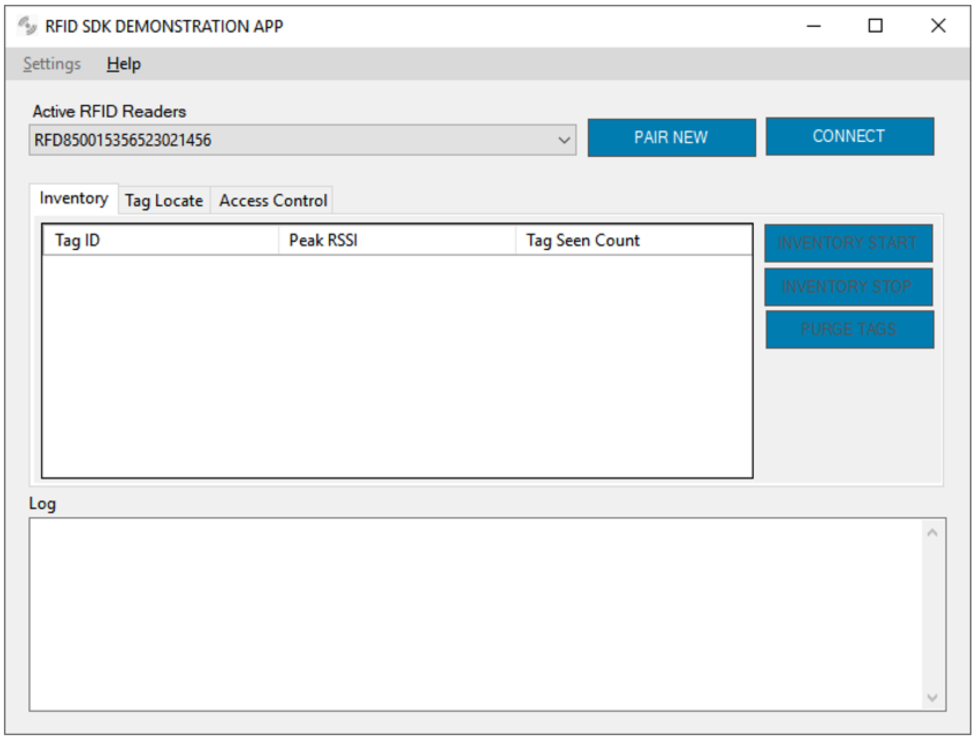
RFD85001535652302
1675

Demo Applications

Windows 7 RFID SDK Demo Application

The Desktop Windows RFID SDK Sample Application shows how to call the RFID Windows API to communicate/configure the RFD8500 reader and receive tag data.

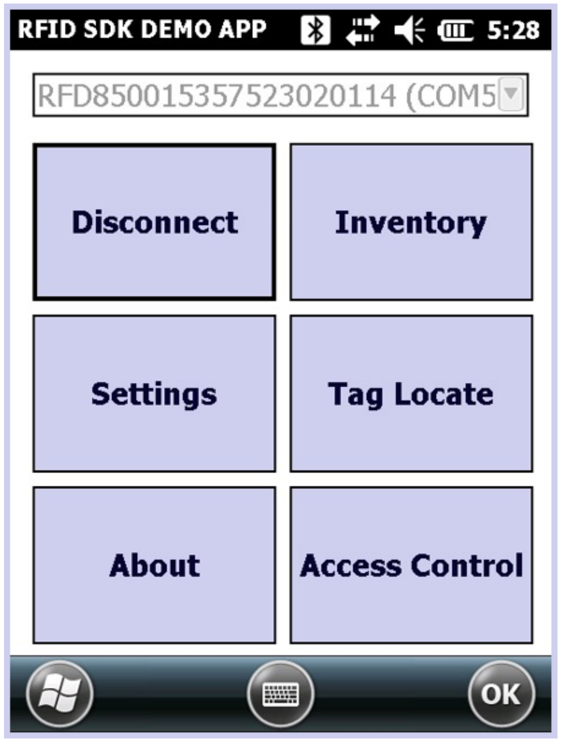
Figure 1 Windows 7 RFID SDK Demo App



Windows Mobile RFID SDK Demo Application

The Windows Mobile Compact Framework RFID SDK Sample Application shows how to call the RFID Windows API to communicate/configure the RFD8500 reader and receive tag data from Windows Mobile Device.

Figure 2 Windows Mobile RFID SDK Demo App


RFID SDK DEMO APP
5:28
RFD85001535752302011 4 (COM5
Disconnect
Inventory
Settings
Tag Locate
About
Access Control
OK