

# Intermec RFID Sample Code User Guide Version 1.0

## 1.0 Introduction

The Simple RFID sample code is provided for educational purposes only. It is not supported and is not meant to function as a demonstration program. Its purpose is to show developers how to use various methods in the RFID Intermec Developer Library (IDL). The code is setup such that you can uncomment the sections you want to try. The GUI provides limited means for selecting RFID operations. It provides some simple controls for basic functionality and feedback. Feel free to add what you need.

## 2.0 Requirements

- Microsoft Visual Studio 2005 or 2008 with C#.
- Intermec Developer Library's for RFID and bar code scanning.
  - RFID resource kit
  - [www.intermec.com](http://www.intermec.com)
  - Or the direct link to the resource kits:

[http://home.intermec.com/eprise/main/GSS/Service/Content/Downloads/Show\\_DownloadSearchResults?Product=DEVRESOURCEKIT](http://home.intermec.com/eprise/main/GSS/Service/Content/Downloads/Show_DownloadSearchResults?Product=DEVRESOURCEKIT)

## 3.0 Connecting to the Reader: `OpenReaderConnection()`

The first step in an RFID application is to establish a connection to the reader. For fixed readers you must specify the connection URL. Typically the URL will be either the com port number or a TCPIP address of the reader:

- `SERIAL://COM1`
- `TCP://192.168.1.`

When you create the `BRIRReader` class you also open the connection to the reader. The sample code shows two common methods for creating the `BRIRReader` object. The first method is the simplest method:

```
brdr = new BRIRReader(this, sConnection);
```

The second method shows how to enable IDL debugging as well as allocating specific amount of memory for the reader and event buffers:

```
BRIRReader.LoggerOptionsAdv LogOp = new  
BRIRReader.LoggerOptionsAdv();  
LogOp.LogFilePath = ".\\IDLClassDebugLog.txt";  
LogOp.ShowNonPrintableChars = true;
```

```
this.brdr = new BRIRReader(this, sConnection, 1000, 1000,  
LogOp);
```

The primary reason for specifying buffers larger than default size is if you plan on reading a large number of tags. Tag IDs are stored in either the reader buffer or event buffer depending on the READ method. If you use REPORT=NO or

REPORT=DIRECT, the tags are stored in the reader buffer. If you use REPORT=EVENT, the tags are stored in the event buffer.

### 3.1 Connection Verification

To verify that you are talking to the reader you can send the PING command. It should respond with an OK>.

### 4.0 Configuring the Reader

Once you have established a connection you may want to configure the reader settings. The first option is to disable auto polling for GPIO trigger events. This is only required for level triggers which are almost never used. Most use cases require edge triggers. To disable the auto polling set the following IDL parameter:

```
this.brdr.SetAutoPollTriggerEvents(false);
```

You will also want to set some of the reader attributes, for example IDTRIES and TAGTYPE. The follow are examples of how to set various reader attributes:

```
this.brdr.Attributes.SetIDTRIES(1);
this.brdr.Attributes.SetANTTRIES(1);
this.brdr.Attributes.SetWRTRIES(3);      //set write tries
this.brdr.Attributes.SetTAGTYPE("EPCC1G2") //set tagtype
this.brdr.Attributes.SetANTS(new int[] { 1, 3, }); //set
//antennas
```

### 4.1 Adding Event Handlers

You will need to add what ever event handlers your application will use. The sample code shows the three most commonly used event handlers for fixed readers:

- Tag event handler
- Radio event handler
- GPIO event handler

The sample code also provides examples of how to use the event handlers, in particular the tag handler which will be used when you read tags with REPORT=EVENT.

### 5.0 Reading Tags

There are four methods for reading tags. You select the method using the REPORT parameter of DIRECT, NO, EVENT, or EVENTALL. When you choose to use DIRECT or NO methods the tag IDs will be stored in the reader buffer. If you choose to use either EVENT or EVENTALL the tags will be stored in the event buffer. Current the EVENTALL method is not supported by the IDL.

#### 5.1 ReadTagsReportDirect()

The simplest method for reading tags is to read only the EPC ID. The only data returned is the 12 byte (24 hex nibbles) EPC ID. The tags will be stored in the reader buffer and can be retrieved as shown in the sample code function LoadTagList(). You can use a WHERE clause to filter out unwanted tags. There is also the option to ask the reader to

return other information such as which antenna read the tag and how many times the tag was read. See the BRI Programmer's Guide for further information.

### 5.2 ReadTagsReportNo( )

When reading a large number of tags or for an extended period of time, it is easier and more efficient to use one of the two continuous read methods. When you set REPORT=NO, the reader stores the tag list and waits for you to poll for the tags. It will not stop reading until you send a READ STOP command. As with the direct mode, you can also use WHERE clauses or ask for other information such as the antenna which read the tag.

### 5.3 ReadTagsReportEvent( )

You can also enable continuous reading if you set REPORT=EVENT. The tags will be stored in the event buffer and returned to the application as events using the tag event handler. Each time the tag event handler fires it returns just one tag id. It will not stop reading until you send a READ STOP command. As with the direct mode, you can also use WHERE clauses or ask for other information such as the antenna which read the tag

## 6.0 Writing Tags

To write tags you must use the EXECUTE method of the IDL. There are numerous options for writing tags so please refer to the BRI Programmer's Guide for further details. The sample code shows four examples, simple write of hex data, write with a where clause, simple write for string data, and a string write with a where clause.

## 6.0 Triggers

Some RFID use cases require motion sensors to start or stop the reading process. Edge triggers and the best technique for using a sensor with the fixed reader. The sample code shows how to create and delete triggers. It also provides a sample GPIO trigger event handler. See the following functions in the sample code for further details:

```
private void SetTriggers()  
  
private void DeleteTriggers()
```

## Appendix A: Source Code For Sample Hand Held Application

The following is the actual sample code written in C# for Visual Studio 2005. The formatting is off because it was copied from the source project.

```
//*****  
//  
//    Simple RFID Sample Code For Win32  
//  
//    Author: Jim Peternel 2008  
//  
//    Copyright 2008 Intermec Technologies Corp.  
//
```

```
// The purpose of this code is to show how to use the various methods
// in the IDL.
// This sample code is not meant to be used as a demo program.
//
// To use this sample code you must download and install the RFID //
// resource kit from
// www.intermec.com
//
//*****
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Intermec.DataCollection.RFID;

namespace SimpleRFIDApp
{
    public partial class Form1 : Form
    {
        //define reader object
        public BRIDReader brdr = null;

        private string[] MyTagList = new string[100];
        private bool bReaderOffLine = true;
        private int iTagCount = 0;

        public Form1()
        {
            InitializeComponent();

            private bool OpenReaderConnection()
            {
                //Establish connection with reader.
                //Choose network or serial connection.

                bool bStatus = false;
                string sMsg = null;
                string sConnection = null;

                //define connection
                //string sConnection = "SERIAL://COM1";
                //string sConnection = "TCP://192.168.1.1";
                if (radioButton1.Checked == true)
                {
                    //serial
                    sConnection = textBox1.Text;
                }
                else
                {
                    //tcpip
                    sConnection = textBox2.Text;
                }
            }
        }
    }
}
```

```

        //optional: enable IDL debug logging
        *****
        BRIRReader.LoggerOptionsAdv LogOp = new
BRIRReader.LoggerOptionsAdv();
        LogOp.LogFilePath = ".\\IDLClassDebugLog.txt";
        LogOp.ShowNonPrintableChars = true;

//*****

        try
        {
            //option 1: open reader connection simple method, no
debugging
            brdr = new BRIRReader(this, sConnection);

            //option two -> set size of reader buffer, event
buffer, and enable IDL logging.
            //Reader Buffer is used for storing tags when you issue
a READ, or READ REPORT=NO
            //Event Buffer is used for storing tags when you issue
a READ REPORT=EVENT and all other events.
            //BRIRReader(this, sConnection, Reader Buffer, Event
Buffer, LogOp);
            //this.brdr = new BRIRReader(this, sConnection, 1000,
1000, LogOp);

            bStatus = true;
        }
        catch (BasicReaderException ex)
        {
            MessageBox.Show(ex.ToString());
            bStatus = false;
        }

        if (brdr == null || bStatus == false)
        {
            //failed to create reader connection
            bReaderOffLine = true;
            PostMessageToListBox1("Unable to connect to reader!");
            PostMessageToListBox1(sConnection);
            return false;
        }

        //autopoll is only needed for Level Triggers which are
almost never used.
        //We recommend using Edge triggers instead which do not
require this feature
        //to be enabled.
        this.brdr.SetAutoPollTriggerEvents(false);

        //Verify that we are actually talking to the RF module.
Should return OK>
        sMsg = this.brdr.Execute("PING");
        if (sMsg != null)
        {
            if (sMsg.Equals("OK>"))

```

```

        {
            //get reader firmware version
            sMsg = this.brdr.Execute("VER");
            ParseResponseMessage(sMsg);

            SetReaderAttributes();
            bReaderOffLine = false;
            bStatus = true;
        }
    }

    if (bStatus == false)
    {
        //not connected to reader
        PostMessageToListBox1("Unable to connect to hand
held");

        PostMessageToListBox1(sConnection);
        bReaderOffLine = true;
    }

    return bStatus;
}

private void ParseResponseMessage(string sMsg)
{
    //parse the response from the reader.
    //Some responses have multiple lines, such as the reponse
    //to the VERSION command (VER). Lines separated by \r\n.

    int x = 0;
    string delimStr = null;
    string[] tList = null;
    char[] delimiter = null;
    int RspCount = 0;

    delimStr = "\n";
    delimiter = delimStr.ToCharArray();
    tList = sMsg.Replace("\r\n", "\n").Split(delimiter);
    RspCount = tList.Length;
    for (x = 0; x < RspCount; x++)
    {
        PostMessageToListBox1(tList[x]);
    }
}

private void SetReaderAttributes()
{
    //Optional Code
    //Some examples of how to set attributes
    string sRsp = null;

    try
    {
        this.brdr.Attributes.SetIDTRIES(1);
        this.brdr.Attributes.SetANTTRIES(1);
        this.brdr.Attributes.SetWRTRIES(3);
        this.brdr.Attributes.SetTAGTYPE("EPCC1G2");
    }
}

```

```

        //enable only antenna port 1
        this.brdr.Attributes.SetANTS(new int[] { 1, });
        //this.brdr.Attributes.SetANTS(new int[] { 1, 2, 3, 4
    });

    this.brdr.Attributes.SetANTS(new int[] { 1, 3, });

    //get the list of all attributes from the reader and
display them.
    sRsp = this.brdr.Execute("ATTRIB");
    ParseResponseMessage(sRsp);
    }
    catch (Intermec.DataCollection.RFID.BasicReaderException
ex)
    {
        MessageBox.Show("SetAttribute Exception : " +
ex.Message);
    }

    private void SetTriggers()
    {
        //Optional Code
        //You need to delete the triggers when you exit your app
otherwise they will
        //still be running after you disconnect.

        string sMsg = null;

        //delete any existing trigger definitions
        DeleteTriggers();

        //CREATE TRIGGERS
        sMsg = this.brdr.Execute("TRIGGER \"SensorON\" GPIOEDGE 1 1
FILTER 0");
        sMsg = this.brdr.Execute("TRIGGER \"SensorOFF\" GPIOEDGE 1
0 FILTER 0");
        PostMessageToListBox1("Triggers Created");
    }

    private void DeleteTriggers()
    {
        //Optional Code
        //You need to delete the triggers when you exit your app
otherwise they will
        //still be running after you disconnect.

        string sMsg = null;

        //CLEAR ALL MACROS AND TRIGGERS
        sMsg = this.brdr.Execute("TRIGGER RESET");
        PostMessageToListBox1("Triggers Delete");
    }

    private void PostMessageToListBox1(string sMsg)
    {
        //POST message to listbox1

```

```

        listBox1.Items.Add(sMsg);
        listBox1.SelectedIndex = listBox1.Items.Count - 1;
        listBox1.Refresh();
    }

    private void CloseReaderConnection()
    {
        //close reader connection by disposing of the object.
        if (brdr != null)
        {
            brdr.Dispose();
            brdr = null;
        }
    }

    private void AddRFIDEventHandlers()
    {
        //add rfid event handlers. There are a number of event
handlers, this
        //example app is only adding 3.
        brdr.EventHandlerTag += new
Tag_EventHandlerAdv(brdr_EventHandlerTag);
        brdr.EventHandlerGPIO += new
GPIO_EventHandlerAdv(brdr_EventHandlerGPIO);
        brdr.EventHandlerRadio += new
Radio_EventHandlerAdv(brdr_EventHandlerRadio);
    }

    void brdr_EventHandlerRadio(object sender,
EVTADV_Radio_EventArgs EvtArgs)
    {
        //Mainly used for Europe, Duty Cycle events. If reader
module overheats
        //you will get a thermal event.
        PostMessageToListBox1("Radio Event: " +
EvtArgs.ToString());
        PostMessageToListBox1("Radio Event: " +
EvtArgs.RadioDutyCycleTimeleft.ToString());
    }

    void brdr_EventHandlerGPIO(object sender, EVTADV_GPIO_EventArgs
EvtArgs)
    {
        //GPIO event has fired. This happens only if you have
created a trigger
        //and it has been activated.

        PostMessageToListBox1("GPIO Event: " +
EvtArgs.TriggerNameString);
        PostMessageToListBox1("GPIO State: " +
EvtArgs.GPIOState.ToString());

        //add your code here
        /*
        if (EvtArgs.TriggerName.Equals("myTrigNameA"))
        {
        }
        else if (EvtArgs.TriggerName.Equals("myTrigNameB"))

```



```

        { }
        */
    }

    void brdr_EventHandlerTag(object sender, EVTADV_Tag_EventArgs
EvtArgs)
    {
        //Used with ReadTagsReportEvent() with
        BRIReader.TagReportOptions.EVENT
        //When using REPORT=EVENT, tags are returned only one time
        by the reader, the
        //first time that they are read. If you stop the read and
        restart it, then the tags
        //we be reported again.

        string sTagData = EvtArgs.DataString.ToString();

        PostMessageToListBox1(sTagData);

        label1.Text = "Tag Count = " + iTagCount;
        label1.Refresh();

        //you must issues a READ STOP to turn off the RF when you
        are done reading.
        //normally you would not stop at this point but would
        continue collecting tags
        //but for this sample we are going to stop reading tags
        here.
        brdr.StopReadingTags();
    }

    private void ReadTagsReportDirect()
    {
        //Simple read

        bool bStatus = false;

        if (bReaderOffLine == true)
        {
            PostMessageToListBox1("Reader is offline");
            return;
        }

        iTagCount = 0;

        //Here are various read options
        //Pick one and comment out the other two

        //1. Simple read of epc id's
        bStatus = brdr.Read();

        //2. Read only tags who's epc id starts with hex 0102
        //bStatus = brdr.Read("HEX(1:4,2)=H0102");

        //3. Return the antenna that read the tag and the number of
        times each tag was read
        //bStatus = brdr.Read(null, "ANT COUNT");
    }

```

```

        //get the tag ids
        LoadTagList();
    }

private void ReadTagsReportNo()
{
    //Continuous read using polling to retrieve tag list

    bool bStatus = false;

    if (bReaderOffLine == true)
    {
        PostMessageToListBox1("Reader is offline");
        return;
    }

    iTagCount = 0;

    //determine how much time you want before polling for tag
list.
    //here it is set to 1 second
    timer1.Interval = 1000;

    //Here are various read options
    //Pick one and comment out the other two

    //1. Read epc id's
    bStatus = brdr.StartReadingTags(null, null,
BRIReader.TagReportOptions.POLL);

    //2. Use a filter to read only tags who's epc id starts
with hex 0102
    //bStatus = brdr.StartReadingTags("HEX(1:4,2)=H0102", null,
BRIReader.TagReportOptions.POLL);

    //3. Return the antenna that read the tag and the number of
times each tag was read
    //bStatus = brdr.StartReadingTags(null, "ANT COUNT",
BRIReader.TagReportOptions.POLL);

    //enable timer to poll for tags
    timer1.Enabled = true;
}

private void ReadTagsReportEvent()
{
    //Continuous Read which returns tags as events

    bool bStatus = false;

    if (bReaderOffLine == true)
    {
        PostMessageToListBox1("Reader is offline");
        return;
    }
}

```

```

        iTagCount = 0;

        //Here are various read options
        //Pick one and comment out the other two

        //1. Read epc id's
        bStatus = brdr.StartReadingTags(null, null,
BRIRReader.TagReportOptions.EVENT);

        //2. Use a filter to read only tags who's epc id starts
with hex 0102
        //bStatus = brdr.StartReadingTags("HEX(1:4,2)=H0102", null,
BRIRReader.TagReportOptions.EVENT);

        //3. Return the antenna that read the tag and the number of
times each tag was read
        //bStatus = brdr.StartReadingTags(null, "ANT COUNT",
BRIRReader.TagReportOptions.EVENT);
    }

    private void WriteTag()
    {
        string sRSP = null;

        if (bReaderOffLine == true)
        {
            PostMessageToListBox1("Reader is offline");
            return;
        }

        //Pick a method to use for writing tags. Below are FOUR
examples of different writing methods.
        //1. Issue simple write command to program the EPC code in
a tag. This write command will write to
        //all tags found.
        sRSP = this.brdr.Execute("W
EPCID=H010203040506070809101112");

        //2. Program an EPC code using a where clause to select a
subgroup of tags.
        //sRSP = this.brdr.Execute("W
EPCID=H010203040506070809101112 WHERE HEX(1:4,2)=H1122");

        //3. Issue simple write command which writes the ascii
string "HI" to the tag. This write command
        //will write to all tags found.
        //sRSP = this.brdr.Execute("W STRING(1:4,2)=\"HI\"");

        //4. Issue write command which writes the ascii string "HI"
to the all tags using a where clause to select
        //all tags that do not have "HI" already programmed into
them. This write command
        //will write to all matching the where clause.
        //sRSP = this.brdr.Execute("W STRING(1:4,2)=\"HI\" WHERE
STRING(1:4,2)!\= \"HI\"");

        ParseResponseMessage(sRSP);

```

```

        //you need to check the response to determine if the write
was successful
        if (sRSP.IndexOf("WROK") > 0)
        {
            //write succeeded. Add your code here.
            iTagCount++;
            label2.Text = "RFID Tag Write Count = " + iTagCount;
        }
        else if (sRSP.IndexOf("WRERR") > 0)
        {
            //Write failed. Add your code here.

        }
        else if (sRSP.IndexOf("ERR") > 0)
        {
            //catch any other type of error that could occur during
the write operation. Add your code here.

        }
        else if (sRSP.IndexOf("ERR") == 0)
        {
            //syntax error in command. Add your code here.

        }
    }

    private void LoadTagList()
    {
        //load epc ids into a list
        //used with REPORT=N0... -> private void timer1_Tick(object
sender, EventArgs e)
        //used with REPORT=DIRECT...default read method -> private
int ReadTagsReportDirect(){}

        if (brdr.TagCount > 0)
        {
            foreach (Tag tt in brdr.Tags)
            {
                MyTagList[++iTagCount] = tt.ToString();

                if (tt.TagFields.ItemCount > 0)
                {
                    foreach (TagField tf in
tt.TagFields.FieldArray)
                    {
                        if (tf.Status < 0)
                        {
                            //code to handle read or write error on
this field

                        }
                        else
                        {
                            //get field data such as ANT, COUNT,
TIME, AFI, etc.

                            MyTagList[iTagCount] += " " +
tf.ToString();

```

```

        }
    }
    }
    PostMessageToListBox1(MyTagList[iTagCount]);
    label1.Text = "Tag Count = " + iTagCount;
    label1.Refresh();
}
}
else
{
    PostMessageToListBox1("NO TAGS");
    label1.Text = "Tag Count = " + iTagCount;
    label1.Refresh();
}
}

private void timer1_Tick(object sender, EventArgs e)
{
    //enabled by ReadTagsReportNo() function
    //timer used to poll for tags

    bool bStatus = false;

    if (bReaderOffLine == true) { return; }

    timer1.Enabled = false;

    bStatus = brdr.PollTags();

    //get the tag ids
    LoadTagList();

    //you must issues a READ STOP to turn off the RF when you
are done reading.
    //normally you would not stop at this point but would
continue to polling for tags
    //but for this sample we are going to stop reading tags
here.

    brdr.StopReadingTags();

    //normally you would re-enable timer and continue polling
for tags
    //timer1.Enabled = true;
}

private void button1_Click(object sender, EventArgs e)
{
    //open reader connection

    bool bStatus = false;

    button1.Enabled = false;

    bStatus = OpenReaderConnection();

    if (bStatus == false)
    {

```

```

        //failed to connect to reader
        MessageBox.Show("Unable to connect to reader!");
        button1.Enabled = true;
    }
    else
    {
        //connected and all is ok

        //create IDL event handlers
        AddRFIDEventHandlers();

        PostMessageToListBox1("Reader online and ready");

        //enable READ button.
        button4.Enabled = true;
    }
}

private void button3_Click(object sender, EventArgs e)
{
    //close reader connection and exit application
    CloseReaderConnection();
    Dispose();
    Application.Exit();
}

private void button2_Click(object sender, EventArgs e)
{
    //clear listbox1
    listBox1.Items.Clear();
    listBox1.Refresh();
    iTagCount = 0;
    label1.Text = "Tag Count = " + iTagCount;
}

private void button4_Click(object sender, EventArgs e)
{
    //Read Button
    if (bReaderOffLine == true)
    {
        PostMessageToListBox1("Reader is offline");
        return;
    }

    ReadTagsReportDirect();
    //ReadTagsReportEvent();
    //ReadTagsReportNo();
    //WriteTag();
}
}
}

```