One Cambridge Center, Cambridge, MA 02142 • Tel: 866-833-4069 • www.thingmagic.com 10/11

Determining Asset Directionality with UHF RFID Technology

By Kapil Asher

Introduction

UHF RFID technology is being widely adopted for tracking asset movement, theft prevention and lost asset recovery. Solutions are being deployed in a variety of industry verticals including healthcare, hospitality, manufacturing, IT data centers, construction and education. Monitoring only the last known location of an asset is often not sufficient in many applications as it does not give information about the direction in which the asset was moving in order to intelligently predict its next location. This is driving the requirement for RFID systems to report directionality in addition to the asset identity and location.

To address these challenges, this application note describes how one might design a solution to predict the direction of a moving asset. The solution includes off-the-shelf passive UHF RFID tags and antennas along with the ThingMagic Mercury6 (M6) reader and ThingMagic MercuryAPI platform. This application note assumes the reader to be generally familiar with RFID technology and the ThingMagic Mercury API.

Note: *Code samples are provided as examples only and will need to be modified to work in your solution environment.*

UHF RFID in Asset Tracking

The use of RFID in asset tracking applications can be divided into the following three parts –

(1) Attaching RFID tags to assets and associating their EPCs to the asset information:
When developing successful RFID applications, the selection of RFID tags and their placement on assets are extremely important factors. It is important to understand a few characteristics of the tag such as the antenna design, power up threshold, reflectivity and tuned frequency band. Tag packaging is also an important characteristic that determines if the tag can be attached to metal surfaces or liquid containers and if it can withstand harsh environments like temperature, humidity, shock and vibration and dust.

To help customers determine the ideal placement of RFID tags on assets, ThingMagic has formulated industry best practices to be followed as a guideline across a large number of asset types and industries. Observing these guidelines increases the performance of the RFID system with respect to tag reads and read range. For help with tag selection and placement, view our video: *RFID Tag Selection & Automated Placement Testing* at http://www.rfid.net/best-practices/43-best-practices/134-tag-it-right-passive-rfid-tag-placement.

Once the appropriate tags have been selected, they need to be commissioned with EPCs blocked for a particular organization and associated with asset information to build an inventory database. For large number of tags, it is generally cost effective to use RFID printers pre-loaded with commissioning and associating software. Printing industry leaders have partnered with ThingMagic to develop RFID enabled desktop and mobile printers for such applications. For a smaller volume of tags, ThingMagic provides a desktop UBS reader, described in the next section, which can be used to commission and associate tags one at a time. More information on RFID printers can be found at - http://www.thingmagic.com/applications-overview.

(2) Building RFID infrastructure to track inventory and asset movement:
Designing RFID infrastructure requires understanding the current workflow of the system and the expected behavior of asset movement. Enterprise level passive UHF RFID readers usually have 2 to 4 antenna ports that can be utilized for door portals and hallways to track assets within a building or a warehouse. ThingMagic's Mercury6 reader, described in the next section, provides 4 monostatic antenna ports and the ability to record tag events uniquely with respect to the antenna number that reads the tag. Strategic placement of antennas in a given workflow can enable seamless tracking of assets at a resolution as high as per-room and even per-rack in a room. Each antenna can be associated to a physical read zone and with the ability to identify a tag uniquely on each antenna, the movement and direction of the tag can be determined. The key is to design and optimize the system such that the read zones do not overlap and a tag is read only by the intended antenna(s) at any given time.

For example – if you need to determine whether an asset entered a room or left it, you will need to build a portal that is divided into 2 parts, one part being inside the entrance of the room and one being outside. When the asset moves into the room, it will be read by the antenna(s) outside the room first, recording the antenna number and timestamp and then by the antenna(s) inside the room. Knowing the antenna ID, their placements, and the timestamp, the software can determine that the asset moved into the room. The ThingMagic MercuryAPI enables developers to easily access this information associated with the tag, simplifying the application software development process.

(3) Uploading data to central database:
Asset tracking and inventory management data are stored on servers that are accessible via web interfaces for managers or other personnel who perform statistical analysis. RFID readers that are network-ready can easily stream data to servers via Ethernet and Wi-Fi. In case network connectivity cannot be achieved, the data can be stored on-board the RFID reader and transferred later. The ThingMagic Mercury6 has the ability to transfer data over Internet backhaul and also temporarily store data on-board the reader until it is queried by the middleware application.

ThingMagic UHF RFID Readers

Variety of RFID readers –

ThingMagic offers one of the largest varieties of UHF RFID readers on the market, all with consistently superior performance. ThingMagic RFID readers can be interfaced via USB, RS232, Ethernet and Wi-Fi, providing a variety of options to connect to the rest of the system. ThingMagic RFID readers are designed for rugged environments, with some models providing up to 4 antenna ports and the option to use RF multiplexers for broader coverage.

Consider the following for your asset tracking and directionality reporting project needs:

The ThingMagic Mercury6 (M6) reader is a network-ready UHF RFID reader that is ideal for reading tags on assets in enterprise facilities, warehouses and many other areas. Out of the box, each M6 reader can be connected to 4 antennas, giving the users an ability to create up to 4 read zones with a single reader. Its continuous mode of transmission allows no down time in polling tags which ensures no missed reads during asset movement. This mode is asynchronous to other processes in the system, allowing uploading of data to servers at the same time as reading new RFID tags. It can be configured to operate at higher physical data rates by utilizing the Gen2 FM0 modulation scheme and a backscatter link frequency of 640KHz. Its high sensitivity and a read rate of more than 400 tags/sec ensures reads even in unfavorable tag orientations. The M6 reader can also read tags that have smaller transaction durations than Gen2 and may be favorable for some applications.

More info - http://www.thingmagic.com/fixed-rfid-readers/mercury6

The ThingMagic USB RFID reader is designed for desktop applications and is ideal for commissioning and associating tags to assets. It is powered via USB and has an integrated antenna with a read range of less than 12 inches to ensure programming only at a close range, thus avoiding association to an unintended tag. The reader has 2 LEDs for visual confirmation of association success and 2 buttons for triggering events.

More info - http://www.thingmagic.com/usb-rfid-reader

Seamless software integration –

ThingMagic readers are available with an SDK for the following platforms – C, C#/.Net and Java. The SDK is designed for cross-product development allowing software engineers to program middleware for all ThingMagic readers without special installation of drivers and DLL's required for specific readers. Function calls and parameters are the same for all readers with the exception of unsupported features on particular reader hardware. An example of a middleware that works with any ThingMagic reader can be found here (Universal Reader Assistant) - http://www.thingmagic.com/support-login?username=api&return=/download?func=startdown%26id=36

Embedded reader advantage –

The core of the ThingMagic technology lies in embedded UHF RFID modules that are integrated with other hardware to develop the finished readers, a path taken by several RFID portal manufacturers. These modules, built over the Atmel ARM microcontroller and RFID ASIC are powered with RFID read and write capabilities in a small form factor. The modules are designed to work for +5VDC (+3 to 5.5VDC for M5eC). A single off-the-shelf cable provides power and data to the modules, eliminating complex solder jobs for hardware integration. ThingMagic modules can be seamlessly integrated with existing telematics hardware allowing faster RFID data transfer to a previously designed system. A high level of control is available via the same MercuryAPI used for ThingMagic finished readers, allowing customers to program their power consumption scheme, boot-up configuration scripts and data transfer rate.

More info: http://www.thingmagic.com/embedded-rfid-readers

NOTE: *Antenna selection and placement are important elements of successful RFID solution performance. To ensure read zone performance and precise and uninterrupted tag-fixed reader communications, be sure the RFID antennas you select meet the performance specifications required for the environments and conditions in which you will be reading tags. ThingMagic offers a variety of RFID antennas for solution development and can assist with selecting the appropriate antennas for your deployment needs.*

Configuration guidelines with code sample

USB Reader –

A simple codelet below shows how to configure the USB reader to perform an association between the EPC tag and the asset.

Please note: this codelet is a guideline and will need to be modified to work in your environment.

Step 1: Configure the reader –

```
private Reader initializeSerialReaders(string comPort)
  {
      Reader rdr1 = null;
      try
      {
          //Define a new reader object
          rdr1 = new SerialReader(string.Concat("/", comPort));
          rdr1.Connect();
          //Set a timeout the host waits for transport messages to and
          from the reader.
          //Useful for slow processors.
          rdr1.ParamSet("/reader/transportTimeout", 10000);
          //Set the legal region allowed for the reader in a given area
          rdr1.ParamSet("/reader/region/id", Reader.Region.NA);
          //Set Gen2 session to 0 as there is only 1 tag in the field
          and continuous response from the tag
          //favours faster encoding
          rdr1.ParamSet("/reader/gen2/Session", Gen2.Session.S0);
          //Set a high baud rate
          rdr1.ParamSet("/reader/baudRate", 115200);
          return rdr1;
      }
      catch (System.IO.IOException)
      {
          MessageBox.Show("Reader not conencted on " + comPort,
          "Error!", MessageBoxButtons.OK);
          return null;
      }
      catch (ReaderCodeException ex)
      {
          MessageBox.Show("Error connecting to Reader: " +
          ex.Message.ToString(), "Error!", MessageBoxButtons.OK);
          return null;
      }
      catch (System.UnauthorizedAccessException)
      {
          MessageBox.Show("Access to " + comPort + " denied. Please
          check if another program is accessing this port", "Error!",
          MessageBoxButtons.OK);
          return null;
      }
 }
```

Step 2: Define Tag Read Event Listener to store tags in the software buffer –

```
  StringDictionary associationDataBase = new StringDictionary();
  TagReadData currentRead = null;
  void PrintTagReadUsb(Object sender, TagReadDataEventArgs e)
  {
      TagReadData read = e.TagReadData;
      if (!associationDataBase.ContainsKey(read.EpcString))
      {
          currentRead = read;
      }

  }
```

Step 3: Read tags –

```
 private void Read()
  {
      rdr1.ReadException += delegate(Object senderException,
      ReaderExceptionEventArgs re)
      {
          MessageBox.Show("Error: " +
            re.ReaderException.Message.ToString());
      };
      rdr1.TagRead += PrintTagReadUsb;
      rdr1.StartReading();
  }
```

Step 4: Associate tag EPCs to assets –

```
private void associate(string currentTag, string assetInfo)
  {
      if ((assetInfo != "") && (currentTag != ""))
      {
          if (associationDataBase.ContainsKey(currentTag))
          {
              associationDataBase.Remove(currentTag);
              associationDataBase.Add(currentTag, assetInfo);
          }
          else
          {
              associationDataBase.Add(currentTag, assetInfo);
          }
      writeToFile(makeDictionaryIntoXML(associationDataBase),
      “output.xml", false);
     }
 }

private string
makeDictionaryIntoXML(System.Collections.Specialized.StringDictionary
poDict)
  {
      string sReturnXML = "";
```

```
            sReturnXML = "";
            foreach (System.Collections.DictionaryEntry oDictEntry in poDict)
            {
                  sReturnXML = sReturnXML.ToString() + "\r\n<item key='" +
                  CheckXMLValue(oDictEntry.Key.ToString()) + "'><![CDATA[" +
                  oDictEntry.Value.ToString() + "]]></item>";
            }

            if (sReturnXML.Trim().ToString() != "")
            {
                  sReturnXML = "<?xml version='1.0'?>\r\n<root>" +
                  sReturnXML.ToString() + "\r\n</root>";
            }
            return sReturnXML;
        }
```

Mercury 6 Readers –

The codelet below shows how to configure the reader to read tags uniquely on different antennas and determine the direction with the help of Timestamps provided as tag metadata by the Mercury API.

Please note: this codelet is a guideline and will need to be modified to work in your environment.

Step 1: Configure the reader –

```
private Reader initializeReaders(string hostName)
  {
     Reader rdr1 = null;
     try
     {
        //Define the antenna port list to use for the reads
        int[] antennas = new int[] { 1, 2, 3, 4 };
        TagProtocol tagProto = TagProtocol.GEN2;
        //Define a read plan that will be executed when the reader is reading
        SimpleReadPlan readPlan = new SimpleReadPlan(antennas, tagProto);
        //Create the reader object
        rdr1 = Reader.Create("tmr://" + hostName);
        rdr1.Connect();
        //Set a timeout the host waits for transport messages to and from the
        reader
        //Useful for slow processors
        rdr1.ParamSet("/reader/transportTimeout", 5000);
        //Set asynchOffTime to a non-zero timeout (ms) to enable pseudo
        asynch mode of operation
        rdr1.ParamSet("/reader/read/asyncOffTime", 10);
        //Set the appropriate regulatory region setting for the region of
        operation
        Reader.Region regionToSet = Reader.Region.NA;
```

```
            rdr1.ParamSet("/reader/region/id", regionToSet);
            RqlReader rqlR = (RqlReader)rdr1;
            //Set the reader to report tags as unique on different antennas
            rdr1.ParamSet("/reader/tagReadData/uniqueByAntenna", true);
            //Set the gen2 Session to 1. Allows reading of more than a single tag
            in the field and does not put the tag to sleep for a long time so
            that the next read point can read the tag.
            rqlR.ParamSet("/reader/gen2/Session", Gen2.Session.S1);
            //Set the gen2 tag encoding scheme to the fastest data rate
            rqlR.ParamSet("/reader/gen2/tagencoding", Gen2.TagEncoding.FM0);
            //Set the gen2 backscatter link frequency to the fastest link rate
            rqlR.ParamSet("/reader/gen2/blf", Gen2.LinkFrequency.LINK640KHZ);
            rqlR.ParamSet("/reader/read/plan", readPlan);
            //Set the read power upto 3150cdBm depending on choice of antenna and
            area RF regulations
            rqlR.ParamSet("/reader/radio/readPower", 3000);
            return rdr1;
         }
         catch (System.IO.IOException)
         {
            MessageBox.Show("Reader not connected on " + hostName, "Error!",
            MessageBoxButtons.OK);
            return null;
         }
         catch (ReaderCodeException ex)
         {
            MessageBox.Show("Error connecting to Reader: " +
            ex.Message.ToString(), "Error!", MessageBoxButtons.OK);
            return null;
         }
         catch (System.UnauthorizedAccessException)
         {
            MessageBox.Show("Access to " + hostName + " denied. Please check if
            another program is accessing this port", "Error!",
            MessageBoxButtons.OK);
            return null;
         }
         catch (ReaderCommException)
         {
            MessageBox.Show("Reader not connected on " + hostName, "Error!",
            MessageBoxButtons.OK);
            return null;
         }
 }
```

Step 2: Define Tag Read Event Listener to store tags in the software buffer and determine direction.

```
/// <summary>
/// Class derived from TagReadData to accomodate tag direction property
/// as MetaData
/// </summary>
class TagReadDataDirection : TagReadData
        {
            private string _direction = "Unknown";
            public string Direction
            {
                get
                {
                    return _direction;
                }
                set
                {
                    _direction = value;
                }
            }
        }

private string getAntennaPosition(int antenna)
        {
            string returnString;
            switch (antenna)
            {
                case 1:
                    {
                        returnString = "Position A";
                        break;
                    }
                case 2:
                    {
                        returnString = "Position B";
                        break;
                    }
                default:
                    {
                        returnString = "Unknown Position";
                        break;
                    }
            }
            return returnString;
        }
        Dictionary<string, TagReadDataDirection> tagDataBaseDirection = new
Dictionary<string, TagReadDataDirection>();
        void PrintTagReadM6(Object sender, TagReadDataEventArgs e, bool flag)
        {
            TagReadDataDirection read = (TagReadDataDirection)e.TagReadData;
            TagReadDataDirection forDataBase;
            string singulationString = "";
            singulationString = read.EpcString;
            lock (tagDataBaseDirection)
            {
                if (tagDataBaseDirection.Count == 0)
                {
                    tagDataBaseDirection.Add(singulationString, read);
                }
                else
                {
                    if (tagDataBaseDirection.ContainsKey(singulationString))
                    {
                        if (tagDataBaseDirection[singulationString].Antenna
                        != read.Antenna)
                        {
                            forDataBase =
                            tagDataBaseDirection[singulationString];
                            tagDataBaseDirection.Remove(singulationString);
                            read.ReadCount += forDataBase.ReadCount;
                            read.Direction = "The asset moved from " +
getAntennaPosition(tagDataBaseDirection[singulationString].Antenna) + " to "
+ getAntennaPosition(read.Antenna);
                            tagDataBaseDirection.Add(singulationString,
read);
                        }
                        else
                        {
                            forDataBase =
                            tagDataBaseDirection[singulationString];
                            tagDataBaseDirection.Remove(singulationString);
                            read.ReadCount += forDataBase.ReadCount;
                            tagDataBaseDirection.Add(singulationString,
                            read);
                        }
                    }
                    else
                    {
                        tagDataBaseDirection.Add(singulationString, read);
                    }
                }
            }
        }
```

Summary

Whether you are working in a distribution center, manufacturing facility, hospital, or on the retail sales floor, monitoring the location, movement and status of your goods, assets, and potentially even your staff, is critical to success. Many RFID portal solutions can report the location of tagged assets, but make you assume directionality. While this assumption-based model may be useful for some applications, knowing directionality with a higher degree of certainty is critical for others.

Although determining directionality is not as difficult as it may first appear, there are some challenges to overcome. As described in this note, one can address these challenges and achieve success with off-the-shelf passive UHF RFID tags, Mercury6 readers and the Mercury API platform from ThingMagic.

For more information, visit www.thingmagic.com.