ILR 350 products, User's Guide

Reg. No. IM.0744.EN
Order Code:
Issue 0 / May 2009
– 05. November 2009 –

IDENTEC SOLUTIONS AG,
Millenium Park 2, 6890 Lustenau, Austria

Phone: +43 5577 87387- 0, Fax: +43 5577 87387-15
Email: info@identecsolutions.at
www.identecsolutions.com

Subject to alteration without prior notice.
© Copyright IDENTEC SOLUTIONS 2007
Printed in Germany

IDENTEC SOLUTIONS

1 Table of Contents

2 Getting started

The goals of this document are:

- Description of functionalities and parameters of ILR 350 products
- How to use, configure and access the products using IDENTEC ILR SDK for .NET

ILR 350 tags can only be used with ILR 350 readers, and the readers can only be interfaced using IDENTEC ILR SDK for .NET or specific applications (demonstration, configuration tools …).

To install and use the SDK, please refer to the SDK documentation.
For other specific software refer to its documentation.

3 Getting started with the SDK

IDENTEC Solutions Software Development Kit is provided as a Microsoft windows installation package.
The deployment package will install on the computer the following components:

- Sample code
- Help file format .chm
- Different dlls:
    - IDENTEC.dll and identec.desktop.dll, for the main functionalities, the first one is built for the compact .NET framework 2.0 and above and the second one for the full .NET framework 2.0 and above.
    - Identec.utilities.dll (2 different files) built for the compact and full .NET framework 2.0 and above

To start using the SDK, you only need to add a reference to the dlls based on the target application.

Note:
The SDK is using log4net for debugging support, so a reference to the log4net.dll must be added to the project.

4 ILR 350 Reader

The ILR 350 reader family primary two products are:

- The i-Port M350 can be interfaced via an RS232 converter or directly via TCP/IP (direct or wireless).The i-PORT M350 readers can be daisy chained, providing access to multiple readers via a unique interface (Serial or TCP/IP).
- The i-CARD CF350 can only be used via a serial interface.

4.1 Connection and enumeration

To connect to readers you need to know which interface is used (serial or TCP/IP) and the interface parameters.

```
IDENTEC.DataStream stream = null;
bool UseTCP = false;
if (UseTCP)
   stream = new TCPSocketStream("192.168.168.230", 2101);
else
   stream = new SerialPortStream("COM2");
try
{
   stream.Open();
}
catch (Exception ex)
{
   Debug.WriteLine("Failed to open interfac :" + ex.Message);
   return;
}
```

Once this information is known, enumeration of readers on the interface is done simply by calling the "EnumerateBusModules" method. This method will return an array of all devices found on the bus.

```
iBusAdapter myBus = new iBusAdapter(stream);
IDENTEC.IBusDevice[] devices = myBus.EnumerateBusModules();
```

Note:
For the i-Card CF 350 the port number selection is done by the host operating system, a special function allows discovery of the serial com port used.

```
int Port = IDENTEC.Readers.CFReaderSearch.FindReaderComPort();
stream = new SerialPortStream(Port);
```

4.2 Communication failure recovery process

During communication between host and reader, communication errors can occur. In this situation, the SDK will throw an exception. In order to recover from this exception, try to close and then reopen the communication interface. If after re-opening the interface the communication with the reader still does not work restart the enumeration process.

Note:
To avoid any potential issue, it is also recommended to reconfigure the reader after reconnecting the interface or a new reader enumeration.

4.3 Reader status event

When communicating with the reader, the SDK will generate an event when the reader has a condition such as a power cycle or "parameters have been reset to default to report".

An application can register this event and catch any error or information reported by the reader.

```
IDENTEC.ILRGen3.Readers.Gen3Reader.EventModuleStatusError += new
iBusModule.OnModuleStatusError(Gen3Reader_EventModuleStatusError);

void Gen3Reader_EventModuleStatusError(object module, iBusDeviceStatus status)
{
   IDENTEC.ILRGen3.Readers.Gen3Reader Gen3Reader;
   if (module is IDENTEC.ILRGen3.Readers.Gen3Reader)
   {
      Gen3Reader = module as IDENTEC.ILRGen3.Readers.Gen3Reader;
      Console.WriteLine("Error " + Gen3Reader.SerialNumber + status.ToString());
   }
}
```

Note:
This is a static delegate; this delegate will be called in case of an error reported by any reader object in the application.

Once the readers have been instantiated, it is highly recommended to configure the reader.

4.4 Reader configuration.

4.4.1 Mandatory

The following configuration is the minimum configuration required in order to detect and communicate with tags.

- Frequency
- RF Beacon baud rate
- Reader wake up duration

If any of the mandatory parameters are not correct, no tags will be detected.

```
 /// make sure that we use default parameters
 /// If we do not reset to factory default we have to make sure to set all parameters of the reader that may
have impact on the application
///Caution resetting the reader to default parameters will empty the reader internal beacon tag list. If you
need to keep it do not reset the reader
 reader.ResetToFactoryDefault();

 /// now we set the parameters we want to use
 /// the tag default beacon baudrate is 115200
 reader.SetRFBeaconBaudrate(RFBaudRate.RF_115200);

 reader.SetFrequency(Frequency.European);

 /// The wake up duration is duration of the wake up signal, the reader will send to wake up a tag
 ///
 /// the wake up duration is on most tags 2 seconds but on special tags it could be different
 /// so changing the wake up duration must be done only if you know you have a special tag
 /// otherwise this value is defined by default to 0.
 /// a wake up duration of 0 means the SDK will automatically select the correct wakeUpDuration
 /// reader.WakeUpDuration = new TimeSpan(0,0,2);
```

Caution:
The frequency shall be set in accordance to the country regulation. It shall not be possible to normal user to change the reader frequency.

4.4.2 Optional

The following parameters are only to be changed in order to improve or fine tune the system, by leaving the default parameters the system will work fine:

- Reader TX power
- RF communication baud rate

```
   /// the TX power must be set based on the antenna used and the expected range
   reader.TXPower = 5;

   /// the following parameters are optional, and should be changed only for fine tuning of the system
   /// we recommend using the lowest baud rate,
   /// increasing the baud rate will reduce the range and communication reliability.
   reader.RFBaudRate = RFBaudRate.RF_115200;
```

Caution:
The device shall not emit RF field in violation of the limit of the standard of country where the device will be used.
The maximum value to use shall be calculated based on the antenna used and potential RF path loss due to cable and connectors.

4.4.3 Beacon message list behavior

An ILR 350 reader will receive and save an internal list of all beacon messages received.
An entry is uniquely identified as same tag ID and same beacon bytes. If either the tag ID or the beacon bytes are different, this will create a new entry in the list.

The list management will be based on the following parameter.

Data Length:
This parameter defines the number of beaconed bytes the reader will keep in his list. The value can be from 0 to 50 (maximum number of bytes a tag can transmit). The number of beacon messages a reader can keep in internal memory will be dependent on this value.

```
   /// This is the maximum number of beaconed bytes saved in the list
   /// A tag can send a maximum of 50 bytes. Setting the datalen to 0 and only tag ID will be saved
  /// increasing the datalen will reduce the list capacity
   reader.SetDataLen(50);
```

List Behaviour:
This parameter defines how the reader will manage the list:
- Remove tag when reported, the reader will delete the tag from the list after it has been reported
- LeaveTag when reported

```
/// - What to do when the tag has been reported
reader.SetTagListBehavior(IDENTEC.Readers.BeaconReaders.TagListBehavior. LeaveTagWhenReported);
```

Inhibit Time:
This parameter defines the maximum time the tag must not be detected by the reader before being removed from the internal list.
If the list behaviour is set to `RemoveTagsWhenReported` then the tag is removed from the list as soon as it is reported so this parameter has no effect.

```
/// - When to remove it from the list
reader.SetTagListInhibitTime(new TimeSpan(0, 0, 60));
```

Re-Report interval:

If a tag has been reported to a host with the GetBeaconTags command, it will be reported only after the Re-report time interval and only if it has been detected after being reported.
If the list behaviour is set to `RemoveTagsWhenReported` then the tag is removed from the list as soon as it is reported so this parameter has no effect.

```
/// - When to report it again
reader.SetTagReReportingInterval(new TimeSpan(0, 0, 60));
```

When are tags removed from list?
Tags are only removed from the list after one of those conditions:

- Tag is reported and the inhibit time condition occurs.
- Tag is reported and list behaviour is set to "`RemoveTagsWhenReported`".
- After issuing a clear list command.
- If the list is full then the reader will go through all the reported beacon messages and remove the oldest one. If no beacon message has been reported then the it will take the oldest one. Note that the oldest message refers to the last time a beacon messages has been receive.

Signal Filter RSSI Level

This is the minimum RSSI level the reader will use to add the tag to the internal list, all tags received with an RSSI level below this limit will not be added to the list.
If a tag is already in the list then all beacon messages will be proceseed regardless of the RSSI level.
If no filter is required, a minimum value of -128 dB must be set.

```
/// minimum RSSI level the reader can detect is approx -100 so setting a value to -128 means the reader will not discard any beacon messages.
reader.SetTagSignalFilterLevel(-128);
```

4.5 Broadcast Write

Tags support broadcast write command.
This allows writing data to all tags or to a special tag type in the field. This command is useful only in specific situations.

Note:
Since this command does not generate an acknowledgement message from the tag, there is no guarantee that the tag received the command.

5 ILR 350 Tags

5.1 Fast/slow sniff and awake time

For battery saving reasons tags are in sleep mode most of the time. The tag wakes up on a regular interval in order to check if a reader needs to communicate with it. This process is called sniffing. Standard slow sniff interval is 2 seconds.

While sniffing, if a reader has been detected, the tag will stay awake to wait for a command from the reader. If the tag does not receive a command, it will go back to slow sniff mode. If it receives a command it will process it and stay awake for 500 milliseconds. After this period it will go to fast sniff mode (100 ms interval). As long as the tag receives valid commands it will retrigger the awake time and stay in fast sniff mode.   After 5 seconds without any valid command, the tag will go back to slow sniff mode.

The wake up is managed automatically by the SDK but can be forced by the application if required.

5.2 Scanning

The scan command will query all the tags in the field. The scan command has an anti collision methodology that is based on a slotted aloha mechanism. When the reader sends the scan command it also sends the number of slots in which the tags have to respond. Increasing the number of slots decreases the collision probability while increasing the time required executing the command. Wake Up duration (it looks like the idea was not fully written here)
Before sending the scan command, the reader can send a wake up command to wake up all tags in the field.

Recommendation:
For better efficiency it is recommended to send 1 scan command with the wake up, followed by multiple scan commands without wake up. This process should be repeated more than 1 time to make sure all tags in range are detected.

```
// demo on how to scan for tags with a 2 seconds wake up
TagList = readerToUse.ScanForTags(10, true, new TimeSpan(0,0,0,2,0));
// this command is also sending a scan command with the wake up duration of the
reader object or 2 seconds if the reader wakeup duration is 0.
TagList = readerToUse.ScanForTags(20, true);
// this is how to send a scan command with no wake up
TagList = readerToUse.ScanForTags(20, false);
Console.WriteLine("Scanned " + TagList.Count.ToString());
```

5.3 Searching for a single tag

Tag objects are created when tags are discovered during a get Beacon message command or when scanning for tags.
If you do not want to scan for tags and the tag is not beaconing or you only want to search for a single tag you can query the reader to ping this single tag.

If the reader finds the tag it will automatically create the tag object from the correct type of tag you are searching for.

```
IDENTEC.ILRGen3.Tags.Gen3Tag tag = readerToUse.PingTag(ID);
if (tag != null)
{
    Console.WriteLine("Found tag " + tag.SerialLabel);
    if (tag.description != null)
        Console.WriteLine("type of tag " + tag.description.Name);
}
else
    Console.WriteLine("tag not found");
```

5.4 Getting and processing Beacon tag messages

To retrieve the list of beacon message the reader has saved internally, use the "GetBeaconTag" method.

```
IDENTEC.ILRGen3.Tags.Gen3TagCollection TagList = null;
 /// now query the reader for the list of beacon tag messages
 TagList = readerToUse.GetBeaconTags();

 Console.WriteLine("Beacon message  " + TagList.Count.ToString());

 /// now process the beacon messages
 foreach (IDENTEC.ILRGen3.Tags.Gen3Tag tag in TagList)
 {
     this.ProcessTagBeaconMessage(tag);
 }
```

This method will return the list of beacon message including all information on the message. The message is an array of bytes, to convert the message to valid info, we do provide utilities.
This utility will parse all information sent by the tag and return a list of information as described below.

```
/// This method demonstrate how to process th ebeacon mesage received
public void ProcessTagBeaconMessage(IDENTEC.ILRGen3.Tags.Gen3Tag tag)
{
   if (tag == null)
      return;
   Console.WriteLine(tag.SerialLabel + " " + tag.BeaconMessageType.ToString() + " Received :" +
tag.TimeLastSeen.ToString());
   List<IDENTEC.Utilities.BeaconData.BeaconInfo> infos =
IDENTEC.Utilities.BeaconData.BeaconDataConverter.Convert(tag);
   if (infos == null)
      return;

   foreach (IDENTEC.Utilities.BeaconData.BeaconInfo info in infos)
   {
      /// check if we have marker information
      IDENTEC.Utilities.BeaconData.LFMarker marker = info as IDENTEC.Utilities.BeaconData.LFMarker;
      if (marker != null)
      {
            Console.WriteLine("loop ID: " +  marker.NewerPosition.LoopID.ToString() + " time :" +
marker.NewerPosition.PositionTime.ToString());
      }
      // check if we have temperature
      IDENTEC.Utilities.BeaconData.Temperature temperature = info as
IDENTEC.Utilities.BeaconData.Temperature;
      if (temperature != null)
      {
         Console.WriteLine("pos: " + temperature.Position.ToString() + " temperature :" +
temperature.Celsius.ToString("F2"));
      }
      // check if user data has been sent
      IDENTEC.Utilities.BeaconData.UserData userData = info as IDENTEC.Utilities.BeaconData.UserData;
      if (userData != null)
      {
         Console.WriteLine("User Data: " + userData.ToString());
      }
      // check if we have a DIgital IO info
      IDENTEC.Utilities.BeaconData.DigitalIO IO = info as IDENTEC.Utilities.BeaconData.DigitalIO;
      if (IO != null)
      {
         Console.WriteLine("DIgitalIO: " + IO.ToString());
      }
   }
}
```

5.5 Configuring Beacon data and interval

ILR 350 tags can beacon configurable data on a configurable interval.

The data transmitted on a regular interval can be composed of the following information:

- User data, up to 50 bytes of user configurable data
- Marker loop information. (only for "L" tags)
- The measured temperature. (only for temperature tag)
- The last 2 Digital input changes detected

The tag beacon interval is defined in 500 milliseconds step, the maximum value is 5 minutes. A value of 0 will disable beacon.

```
BeaconInformation info;
TimeSpan interval = TimeSpan.MinValue;
byte[] UserData;
/// the following call will read the full reader beacon configuration
tag.ReadBeaconConfiguration(readerToUse, out info, out interval, out UserData);
Console.WriteLine("Tag :" + tag.SerialLabel + " beacon interval "
                + interval.ToString() + " beacon data " + info.ToString());

/// this allows to read only the beacon interval
interval = tag.ReadBeaconInterval(readerToUse);
/// this method allows to read the user data
UserData = tag.ReadBeaconUserData(readerToUse);
```

```
/// the following calls will set the beacon configuration
/// Here the tag will beacon every 20 seconds
tag.WriteBeaconInterval(readerToUse, new TimeSpan(0, 0, 20));
byte[] data = System.Text.Encoding.ASCII.GetBytes("Beacon info");
/// If the tag is set to beacon user data it will beacon "Beacon info"
/// Caution the tag can beacon only 50 bytes in total so if the tag is also
beaconing other information
/// like marker or temperature, and the user data length is too long
/// the tag will truncate the user data to the maximum size it can transmit
tag.WriteBeaconUserData(readerToUse, data, data.Length);
/// The tag is set to beacon the marker info and the user data
tag.WriteBeaconConfiguration(readerToUse, BeaconInformation.Marker |
BeaconInformation.UserData);
/// The tag is set to beacon the marker info and the user data and the
temperature (if we have a temperature tag)
tag.WriteBeaconConfiguration(readerToUse, BeaconInformation.Marker |
BeaconInformation.UserData | BeaconInformation.Temperature |
BeaconInformation.DigitalIO);
/// the following call will disable the beacon messages
tag.WriteBeaconConfiguration(readerToUse, BeaconInformation.None);
```

The tag battery lifetime is dependent on the beacon interval and data length to beacon.

5.6 Events Beacon messages

Tags have the possibility to beacon a burst of beacon message upon detection of specific events.
The list of event could be:

- Detection of an LF marker (only for "L" tags)
- Detection of alarm condition for measured value out of range (only tag with sensor)
- Detection of Digital input change

When an event condition is detected by the tag, the tag will beacon configurable data with:

- The beacon information related to the event, for example for a marker event, the tag will always transmit the marker message.
- Extra data requested by the user, for example user data or marker or digital input or temperature.

The additional parameters of the event beacon messages are used to improve detection of event beacon message collision when multiple tags send an event at the exact same time. This can be common when tags are entering a marker field at the same time:

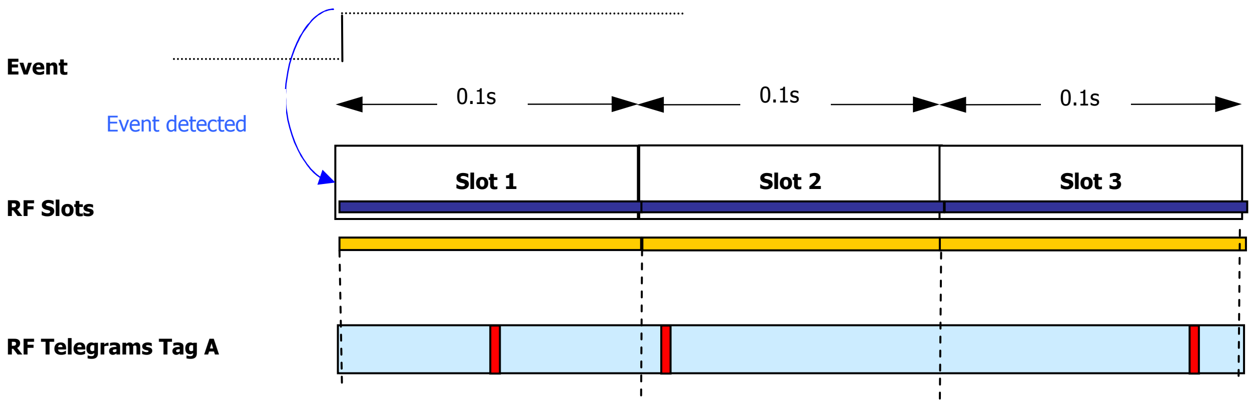
- The number of time the beacon message will be transmitted (maximum 15).
- The slot size. The slot size defines a time interval when it can transmit randomly within this interval one beacon message. The next beacon message will be transmitted at a random time within the next slot time.

```
BeaconInformation info;
TimeSpan slot = TimeSpan.MinValue;
int burstNumber;
```

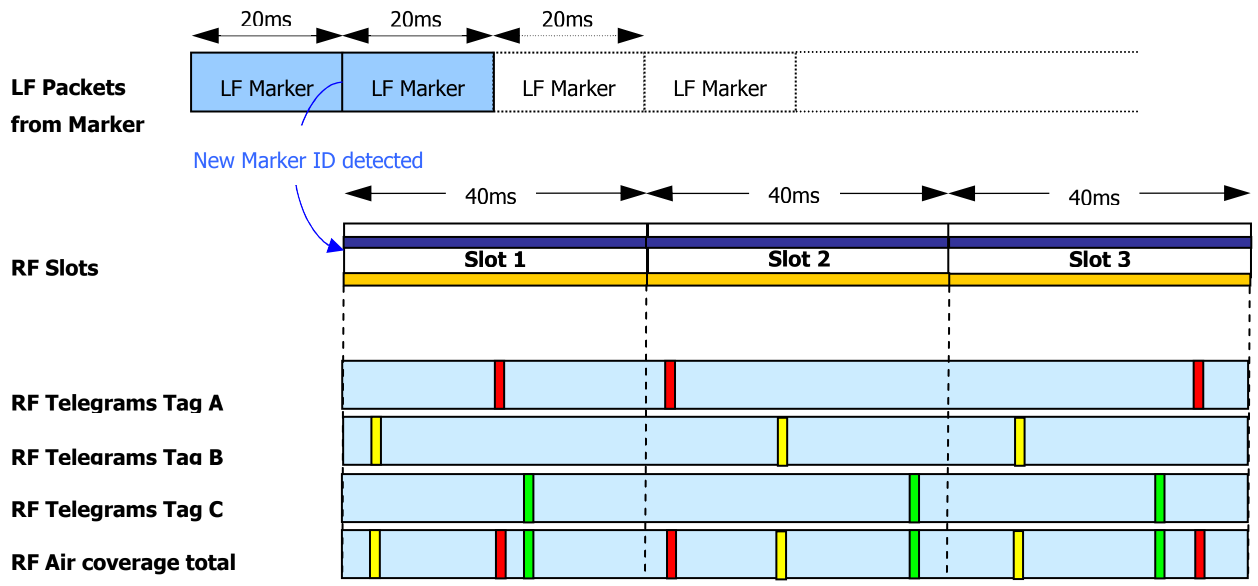
```
/// first we read teh current information
tag.ReadEventConfiguration(readerToUse, EventType.Marker, out info, out slot, out
burstNumber);
Console.WriteLine("Tag :" + tag.SerialLabel + " marker event "
                + info.ToString() + " interval  " + slot.ToString() + " burst " +
burstNumber);

/// the following call will request the tag to beacon the user data
/// 5 times randomly in interval slot of 2 seconds
/// Note that the marker information will always be sent even if not requested
tag.WriteEventConfiguration(readerToUse, EventType.Marker,
BeaconInformation.UserData, new TimeSpan(0,0,0,2), 5);
```

Below is an example of event detection with a 100 ms slot size and 3 beacon messages sent.



Below is an example of tags entering a marker field at the same time.


20ms
20ms
20ms
LF Packets from Marker
LF Marker
LF Marker
LF Marker
LF Marker
New Marker ID detected
40ms
40ms
40ms
RF Slots
Slot 1
Slot 2
Slot 3
RF Telegrams Tag A
RF Telegrams Tag B
RF Telegrams Tag C
RF Air coverage total

5.7 Logging

5.7.1 Measurement interval

The logging process uses the measurement interval. By default the tag will make sure that the measurement interval is at least equal to the logging interval to make sure that each log sample is a new value. If you set the measurement interval 2 times faster than the logging interval, the logger will log the average of all the values measured after the previous log sample.

```
/// We suppose the tag will be logging every 10 minutes
/// we set the measurement interval to the same value as logging interval
/// so each sample is a new value instead of an average of multiple samples
tag.WriteMeasurementInterval(readerToUse, new TimeSpan(0, 10, 0));
/// in this call we set the measurement interval to twice the logging interval
/// meaning each log sample will be the average of 2 samples
tag.WriteMeasurementInterval(readerToUse, new TimeSpan(0, 5, 0));
```

5.7.2 Start, stop logging and read log

Starting the log requires only 1 parameter (Log interval) additional to the reader to use:

```
tag.StartLogging(readerToUse, new TimeSpan(0, 10, 0));  // start logging with a
10 minutes interval
```

Stopping the log requires the reader to use the following:

```
tag.StopLogging(readerToUse);
```

Reading the log information can be done even when the tag is logging.

```
/// for information we read the measurement interval
TimeSpan interval = tag.ReadMeasurementInterval(readerToUse);
Console.WriteLine("Tag "  + tag.SerialLabel + " measurement interval " +
interval.ToString());

/// reading the log info will allow to check all the tag logging status
IDENTEC.Tags.Logging.LogInfoData info =
LoggerTag.ReadLogInformation(readerToUse);
if (info != null)
{
     Console.WriteLine("tag " + tag.SerialLabel + " logging state: " +
info.IsLogging.ToString() + " wrapped " + info.Wrapped.ToString());
     Console.WriteLine("Interval :" + info.LoggingInterval.Minutes.ToString() +
":" + info.LoggingInterval.Seconds.ToString());
     Console.WriteLine("Start Time:" + info.LoggerStarted.ToString());
     if (!info.IsLogging)
            Console.WriteLine("Stop Time :" + info.LoggerStopped.ToString());
}
```

Reading the logged data is done using the following method:

```
IDENTEC.Tags.Logging.RawLogData log = null;
log = tag.ReadLastnLogSamples(readerToUse, 0, 20); /// we request to read all
data with maximum 20 retries.

/// display the temperature data
IDENTEC.Tags.Logging.TemperatureLogData temLog = log as
IDENTEC.Tags.Logging.TemperatureLogData;
if (temLog != null)
{
    IDENTEC.Tags.Logging.TemperatureLogSampleCollection samples = temLog.Samples;
    int SamplePerLine = 6;
    foreach (IDENTEC.Tags.Logging.TemperatureLogSample sample in samples)
    {
        Console.Write(sample.SampleTime.ToString() + " " +
sample.DegreesCelsius.ToString("F1") + "  ");
        if (--SamplePerLine == 0)
        {
            SamplePerLine = 6;
            Console.WriteLine();
        }
    }
    Console.WriteLine();
    Console.WriteLine("Number of sample read :" + log.SampleCount + " out of " +
log.LogInfo.TagSampleCount.ToString() + " stored in tag");
    Console.WriteLine("lowest : " + temLog.LowestTemperatureRecord.ToString() + "
highest : " + temLog.HighestTemperatureRecord.ToString());
}
```

Note: The logger reference time can be invalid, if the log was started from a device with an invalid time or if the tag has been reset.

5.7.3 Measured temperatures extremes

Temperature tag saves the maximum and minimum temperature measured.
When the logger is started, the tag reinitialized those extremes values and after every sampled temperature it updates the values if needed.
Those extreme temperatures do represent the maximum and minimum temperature measured since the logger was started.
The values may not be present in the logged data for 2 reasons:

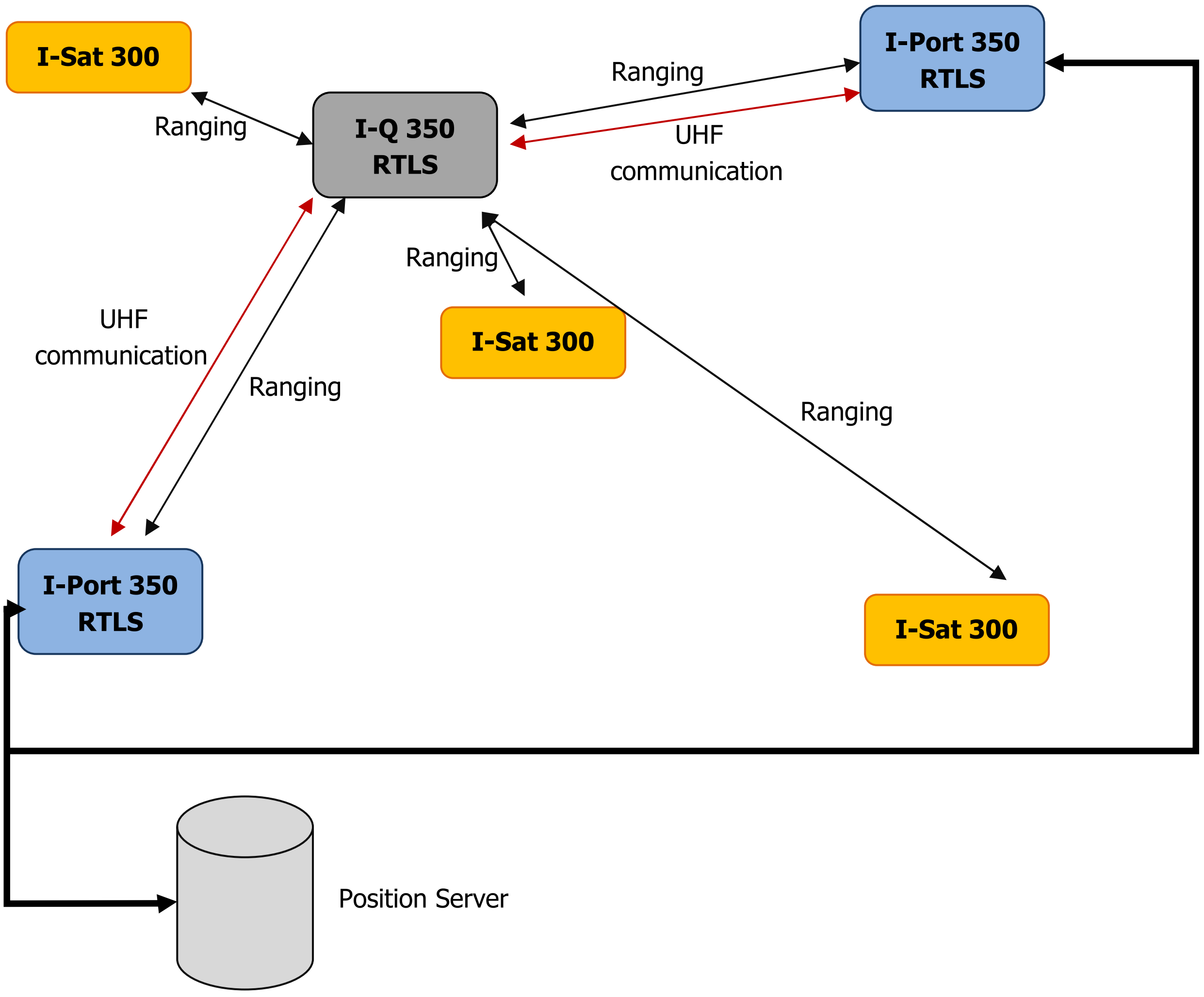
- The logger is wrapped and the values have been erased by new values in the log
- The tag is logging average measured values.

Note: The maximum and minimum temperatures are always updated even when the tag is not logging.

```
/// <summary>
/// This method demonstrates how to read the maximum and minimum temperature
measured by the tag
/// </summary>
/// <param name="tag"></param>
public void ReadTemperatureExtremes(IDENTEC.ILRGen3.Tags.iQ350Logger tag)
{
      /// for information we read the measurement interval
       IDENTEC.Tags.Logging.TemperatureExtremes data =
tag.ReadTTemperatureExtremes(readerToUse);
      Console.WriteLine("Tag "  + tag.SerialLabel + " max temp " +
data.MaximumDegreesCelsius.ToString() + " min temp " +
data.MinimumDegreesCelsius.ToString());
      Console.WriteLine("logger started " + data.LogStart.ToString() + " logger
end time" + data.LogEnd.ToString());
}
```

6 ILR 350 RTLS Tag

6.1 RTLS System overview



The RTLS system has been designed to provide a very light infrastructure with a minimum of cabling required. The system is composed of the following components:

- **(TAG) :** Model: I-Q 350 RTLS: the tag which performs measurement with readers
- **(iSAT) :** Model: I-Sat 300: a reference point which performs measurement with tags and only requires power to operate.
- **(Reader) :** Model: I-port 350 RTLS: this reader act both as an I-sat 300(reference point) and I-port 350(reader). It is able to perform measurement with tags(Ranging), and as well to communicate over UHF frequencies(UHF). This reader has an Ethernet or RS422 communication to the position server.

During standard operation, i-Q 350 RTLS tags will periodically perform, or try to perform, ranging with the readers (i-Sat 300 or i-Port 350 RTLS) in their surroundings. The tag can perform measurement with a maximum of 15 readers at once. The interval between two measurements is called the ranging interval. After

their measurements are finished, tags will broadcast their results over UHF frequencies to all the I-Port 350 or i-Port 350 RTLS in range.
When a tag is in motion it will automatically switch to a "in motion" mode. Two option are possible and are described hereafter.

6.2 Parameters

6.2.1 2.4GHz Output power

It is possible to change the output power of the i-Q 350 RTLS(Tag) with values described hereafter

- Value: from 0dBm to 20 dBm
- Unit: 1 dBm
- Default value: 20 dBm

The output power will influence the achievable range of the tag. 20dBm will give the longest range

6.2.2 "In motion" behavior

The i-Q 350 RTLS(Tag) is able to identify if it is in motion or stand still. In the case of movement, the tag will switch to the "in motion" behavior. Two behaviors are available and described hereafter:

- **Continuous**: While in motion, the i-Q 350 RTLS(Tag) will perform ranging using the interval specified in "ranging interval in motion".
- **On-Stop**: The i-Q350 RTLS(Tag) will keep its standard ranging interval while moving. As soon as the i-Q350 RTLS(Tag) detects that it stops, it will perform 10 measurements with the interval specified in "ranging interval in motion"

6.2.3 Ranging interval

The ranging interval defines the interval between 2 consecutive measurements. A measurement consists of several distance estimation between an i-Q350 RTLS(Tag) and readers. A maximum of 15 readers can be used at once.
The ranging interval parameter is specified as described hereafter:

- Value: from 1 second to 4 hours and 39 minutes
- Unit : 1 ms (millisecond)
- Default value: 5 minutes

6.2.4 Ranging interval in motion

The Ranging interval in motion is an alternative interval which is used depending on the "in motion" behavior specified.
The ranging interval in motion parameter is specified as described hereafter:

- Value: from 1 second to 1 minute
- Unit : 1 ms (millisecond)
- Default value: 30 seconds