Utrecht University of Applied Sciences,
Faculty of Natural Sciences & Technology,
Research centre for product development.

A remote sensor network using ZigBee and GPRS with data retrieval

Author:
Marten A. Janssen
1518673

Supervisor:
M.Eng E. Puik

February 12 2010

Abstract

A remote sensor network can be used for a great variety of purposes e.g. in our case, to monitor the compressive strength of concrete during its curing process and improve the concrete treatment [10, 18].

> *The method of weighted maturity through in-site temperature readings is valued highly by both academic as well as professional experts. Primarily for the non-destructive character, the accuracy, the ability to control the process with real-time measurements and simplicity as there is no need for (on-site) stored and unrepresentative sampling cubes, highly specialist workers, and equipment.* [18]

Our end-devices (equipped with a temperature sensor) can be embedded into concrete and take temperature samples from within. The data samples from the end-devices need to travel through several nodes, over different interfaces and protocols and are transformed many times before reaching its destination, the server (see figure 1). The WSN uses ZigBee® for its energy efficient features. A sensor device can work on a single button cell for several weeks, months, or even years, depending on the application. It can collect data samples and from these samples, assemble a message, called a sequence. Such a sequence stored locally and sent to the coordinator if it is reachable. The data might travel through several ZigBee® routers if the coordinator is out of range of the end-device.

The ZigBee® coordinator transfers the sequence over RS-232 to the GPRS module for RS-232 is widely used, reliable and supported by J2ME™, which runs on the GPRS module. The GPRS module keeps track of the sequence numbers and will request missing ones.

The GPRS module makes it possible to have a wireless sensor network at one location and the data-collecting server at another, as long as the GPRS module is in range of the GPRS network (and has sufficient credit). The server can build a model and graphs that as a result can be showed to the costumer at any location with an internet connection. From these results, the client can see the progress of the curing process and can react on it if necessary.

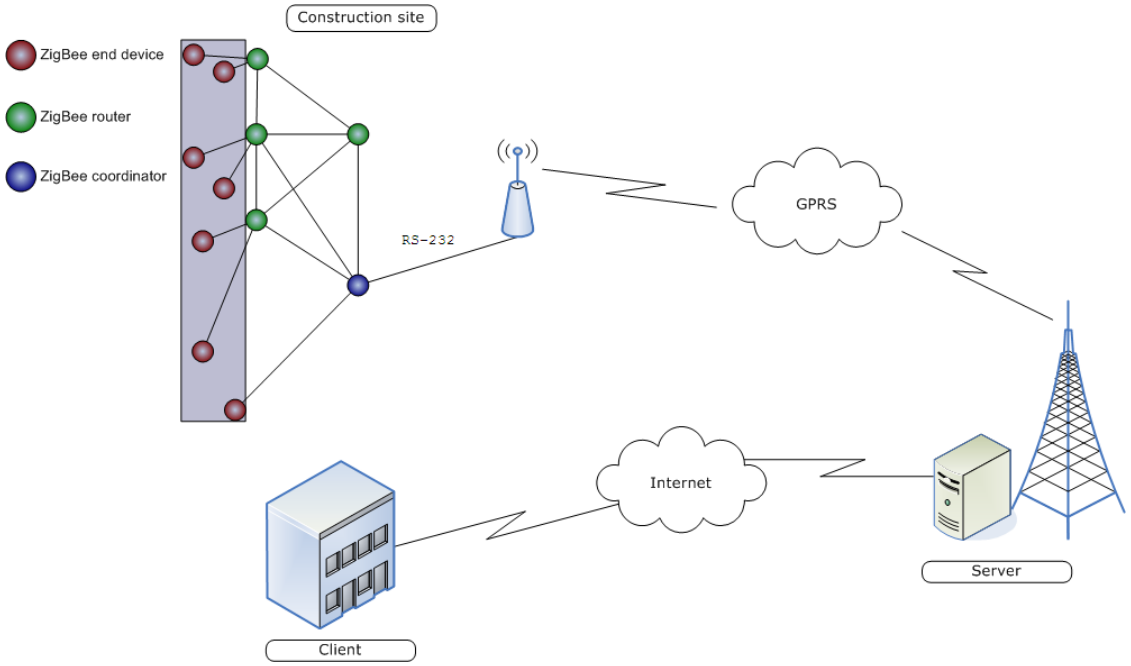


Figure 1: Overview of the situation.

Document information

Date:
February 12 2010

Author:
M.A. Janssen, student Computer Science
Utrecht University of Applied Sciences, Faculty of Natural Sciences & Technology.
Oudenoord 700
Room E.0.14
3513 EX Utrecht
marten.janssen@student.hu.nl
www.martenjanssen.com
+31 (0)6 - 4852 4936

Originating principal:
Professor M.Eng E. Puik, professor Chair of Micro Engineering
Utrecht University of Applied Sciences, Faculty of Natural Sciences & Technology,
Chair of Micro Engineering.
Oudenoord 700
Room E.0.14
3513 EX Utrecht
erik.puik@hu.nl
+31 (0)651 - 55 40 41

Study:
Utrecht University of Applied Sciences, Faculty of Natural Sciences & Technology,
Bachelor Computer Science.
Nijenoord 1
3552 AS Utrecht, the Netherlands
www.hu.nl
+31 (0)30 - 230 81 08

Acknowledgements

This project was not possible without the assistance and contribution of many. First, I would like to thank originating principal professor M.Eng E. Puik for setting up this project, his insightful comments, and for giving me the opportunity to work on this project. I would like to pose a sincere thanks to research assistant Dennis van Wijk of Utrecht University of Applied Sciences for his work on the sensor device prototype, sharing his vision on setting up the different data representation protocols, pondering with me about several complexities, his incredibly useful input, and pleasant collaboration. I would like to thank trainee Wouter Oet for sharing his vision on setting up the data structure presentation between the GPRS module and the server in XML and trainee Raymond Siudak for providing me a useful USB to serial converter.

Furthermore, I would like to thank M.Eng. Mark D. Tammer for his research in wireless monitoring of the concrete curing process, which led to this project. Last but not least, I would like to thank Utrecht University of Applied Sciences lecturer M. Wensink for his insight on ZigBee® and the Z-Stack™ and his critical look on my documentation.

Table of contents

List of Figures

List of Examples

1

Introduction

Recent advances in micro-electro-mechanical systems, microprocessors, and low power radio technologies offer low-cost, low power, but multi-functional miniature sensor devices, which can observe changes in their surrounding environments [1,9,33]. These sensor devices can also be used as an innovative tool to measure the quality of concrete [18] or to monitor the vibration of a highway bridge [33]. Wireless sensor networks (WSN) are composed of a set of deployed (sensor) devices, capable of communicating with each other wirelessly.

Since sensor devices typically have a limited power supply, it is essential to make them as energy efficient as possible. [31] indicates that the ZigBee® protocol stack (Z-Stack™) in combination with Texas Instruments' CC243x micro controller has a potential energy consumption as low as 0.3μA when in sleeping mode (see section 2.5 Sleep modes), which allows it to run for years on a single button cell.

1.1 Previous and related work

We have worked with ZigBee® and the CC243x before to implement a method by which the CC243x can be reprogrammed over the air (OTA) [6, 30], which can also be implemented if the hardware meets the minimum hardware requirements, defined in [6].

M.Eng. Mark D. Tammer has done a research in the potential of wireless monitoring of the concrete curing process using microchips embedded into the concrete [18]. He discovered that the embedded microchips have a great potential to monitor the curing process of concrete and that is an innovative and accurate technique. He stated "The stated concatenation of execution process related incidents are triggering a cautious emergence of insurance party initiated inspections and initiatives like a structural 'birth certificate' to close the gap between prescribing regulations and actual achieved structural asset characteristics.".

1.2 Research subject

Under the guidance of professor M.Eng Erik Puik, research assistant Dennis van Wijk has built a PCB, based on the CC2430 (the microchip mentioned above), which is capable of measuring, storing and wirelessly transmitting temperature and vibration levels. With these capabilities in mind, software has been built for the WSN, based on ZigBee® 's Z-Stack™ , that retrieves sample data from sensor devices in an energy efficient way and sends this data over GPRS to a centralized server for analyses and a generic frame format has been created for the data over several devices. Older data samples can also be retrieved from the end-devices when the server did not receive them, e.g. when the WSN was unreachable for an extended period, to be able to create accurate analyses. The end-device is (by default) capable of storing 320 sequences of a single data sample, allowing it to store unique data samples for 22 weeks when it takes a sample twice a day,

$$(320 \text{ sequences}) \;/\; (2 \text{ per day}) \;/\; (7 \text{ days a week}) \;\approx\; (22 \text{ weeks}).$$

After 320 samples, the oldest sequence is overwritten with a new sequence.

To support the OTA download feature (discussed in section 1.1), the target MCU requires a flash memory that can hold at least twice the size of the application image (typically 64kB) and 10kB for boot code and application itself. Since our CC243x has 128kB of on-chip flash, it requires external flash memory to support this feature. In addition, the number of sequences that can be stored also depends on the size of free flash memory.

1.3 Research path

The first phase of this research focuses on a literature study and experimental tests on ZigBee® 's Z-Stack™ . Our ZigBee® development kit [21] does not support communication over GPRS and therefore, a GPRS module must be found, purchased and programmed so that can be connected to the WSN.

Second, a generic frame format must be built and implemented. A demonstration of the project is given at the precision fair of December 2009 in Veldhoven. At this fair, we have showed that a sensor device can

indirectly send its sample data to the server, located in Utrecht, and how a graph can be built from this data[1]. For this demonstration, we used a cellular GPRS connection instead of the GPRS module since the GPRS module was not ready yet.

Third phase is to implement all requirements in the software, e.g. to retrieve older data samples from a sensor device. The fourth and final phase is to test all requirements, and write a thesis about the results. Table 1.1 shows the software used during this project.

Category	**Application name**	**Version**
Debug	Notepad++	4.9.0.0
	RealTerm	2.0.0.57
Documentation	Adobe Reader	8.1.2
	Artisan Studio Uno	7.0.22
	Crimson Editor	3.70
	Microsoft Visio	2007
	Picasa	3.1.0
	TeXnicCenter	1.0
	MiKTeX	2.7
GPRS module	Eclipse SDK	3.2.2
	EclipseME plugin	1.5.5
	Java Development Kit	6.06
	Siemens Module Exchange Suite	TC65 Rel.3
OS related	Microsoft Windows XP	Professional
ZigBee platform	IAR Embedded Workbench IDE	7.30B
	Texas Instruments SmartRF Studio	6.12.0.0
	Texas Instruments ZStack	1.4.3-1.2.1

Table 1.1: Used software and their versions.

1.4 Planning

The official start of the project was September 1 2009 and should be finished on December 15 2009, allowing students to work on the project for 15 weeks. Since this is an honour project (and therefore bigger and more challenging than regular projects) the deadline for thesis hand-in is postponed to February 12 2010. At first, the deadline was not postponed due to the fixed dates of the graduation sessions. In November 2009, it became clear that the original deadline was not feasible.

Date			**Description**
September	01	2009	Start project
October	12	2009	Graduating contract hand-in
October	13	2009	Project plan hand-in
November	09	2009	Thesis concept
December	2-3	2009	Precision fair demonstration
January	18	2010	Thesis design hand-in
February	12	2010	Thesis hand-in (x4)
April		2010	Company transcript hand-in
April		2010	Sessions for graduation

Table 1.2: Fixed dates.

[1] This graph was generated by trainee Wouter Oet

2

ZigBee in general

It is widely acknowledged that standards such as Bluetooth and WLAN are not suited for low power, low-cost, low data rate applications due to their high node costs and complex, power-hungry RF ICs and protocols. The ZigBee® standard however, allows simple, low-cost wireless networks with nodes incorporating diverse applications from different vendors. These types of networks are typically used for monitoring and control purposes, and require very little power, which means they can run for years on inexpensive batteries. The ZigBee® specifications support robust mesh networks that can contain hundreds of nodes. Such networks permit messages to travel a number of different routes to get from one node to another, making a reliable network not dependent on any particular individual node to function. For system developers it is much more cost-effective to design their applications based on a common standardized ZigBee® platform than to create a new proprietary solution from scratch each time. Instead of solving complex radio and network issues, designers can now focus on building applications on top of the ZigBee® framework. [6]

Two different device types can participate in an IEEE 802.15.4 (ZigBee®) network; a full-function device (FFD) and a reduced-function device (RFD). The FFD can operate in three modes serving as a personal area network (PAN) coordinator, a coordinator, or an end-device (see figure 2.1). An FFD can talk to RFDs or other FFDs, while an RFD can talk only to an FFD. An RFD is intended for applications that are extremely simple, such as a light switch or a passive infrared sensor; they do not have the need to send large amounts of data and may only associate with a single FFD at a time. Consequently, the RFD can be implemented using minimal resources and memory capacity. [5, 8]

In the ZigBee® network, the end-devices sleep most of the time and therefore, the communication between coordinator and end-device is set up when the end-device wakes up. The coordinator receives a message from the end device, containing one or more data sample(s), and sends it over RS-232 to the waiting GPRS module. The GPRS module converts this data into an XML format that the server can understand and sends this data over GPRS to the server. The GPRS module also keeps record of the sequence numbers of the data samples and will request missing sequences by sending a request over RS-232 to the coordinator. The coordinator puts the message in a 'pending' list and, when the end-device wakes up, it will send the request for this/these sequence(s).

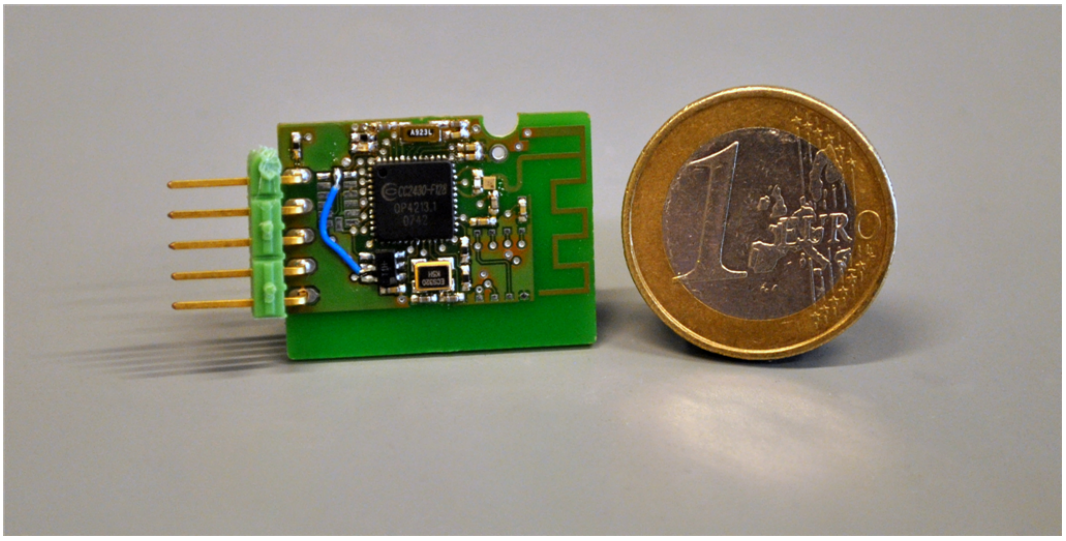


Figure 2.1: End-device prototype with digital temperature sensor.

In ZigBee® , a device can have three network functions: a Coordinator, a Router, or an End-device.

2.1 Coordinator

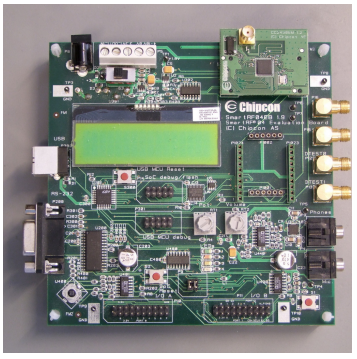


Figure 2.2: ZigBee coordinator.

The ZigBee® coordinator (figure 2.2) is unique in the WSN. It hands out a range of 16b addresses to nearby routers that they can use to hand out to the end-devices. The end-device uses its 64b address to send a request to join the network. If the request is accepted, it will receive a 16b address for further communication. The sample data sent by the end-devices is received by the coordinator. The LCD of the coordinator (if present) can be used to display the various messages received from the end-devices for debug purposes. One of the most important requirements in this project is that older data samples must be able to be retrieved from the end-devices. Since the end-devices do not verify if a data sample is received by the coordinator, some device must hold a list of data samples received and automatically send a request back to the end-device if a the sequence received does not follow up the previous received sequence number. Since being a coordinator already requires up to 70% of its CPU time, we chose not to stress it even further by implementing features like storing all data from each end-device, encode the data, and automatically queue a request to an end-device for a specific sequence (data sample set) that is missing. Instead, we implemented these features in the GPRS module.

2.2 Router

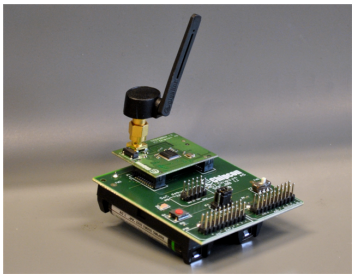


Figure 2.3: ZigBee router.

A ZigBee® router (figure 2.3) can be placed between the coordinator and the end-devices on locations where the end-device's radio is insufficient to reach the coordinator by itself. The purpose of the router is thus mainly to transfer the data from the end-device to the coordinator (possibly through several other routers) and vice versa. The coordinator hands out a range of addresses to the router, which it uses to hand out to end-devices that want to join the network. The network architecture defines how many end-devices can be connected to a single router. How many routers are needed to reach all end-devices is highly influenced by its environment, e.g. metal, wet concrete. Tests indicated that our end-device prototype has a range of 50m in open air. Being an FFD, a router is usually connected to the mains, but it can also be battery-powered, which makes it easier to spread on a building construction.

2.3 End-device

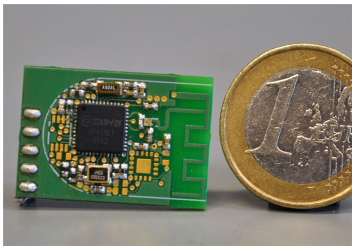


Figure 2.4: End-device prototype.

The end-device (figure 2.4) contains sensors and usually has a battery for power-supply. For monitoring the concrete curing process, the end-device is embedded into the concrete. When enabled, its battery (a button cell) will die in 4-6 weeks. Hence, it is essential to make the end-device as energy efficient as possible. The most energy hungry feature is the radio. The end-device can poll its sensor(s) up to 1000 times before it consumes the same amount of energy as sending the sample data to the coordinator only once. One should determine the necessity of each sample data, e.g. two temperature samples that are equal to each other are not as important as when they would have a difference of 70°C. The ideal threshold depends on the project. Therefore, it should be adjustable (see section 6.3 Send new parameters to end-devices).

Because the end-devices can be programmable over the air, a minor adjustment in the threshold settings can be made before sending the final application to the device, but such an adjustment can also easily be made by implementing a listener for new parameters to the end-device.

2.4 CC2430 microchip

Figure 2.5: CC2430 microchip.

Previous research [34] have indicated that there are a few different manufacturers of μcontrollers that incorporate both μcontroller and transceiver in the same package or chip and allow user-code to be run on the controller. Texas Instruments' CC2430 (see figure 2.5) was the most expensive, but also most energy efficient at the time. Hence, the CC2430 ZigBee® development kit [21] was purchased.

In [31] is stated: "The CC2430 comes in three different flash versions: CC2430F32/64/128, with 32/64/128 KB of flash memory respectively. The CC2430 is a true System-on-Chip (SoC) solution specifically tailored for IEEE 802.15.4 and ZigBee® applications. It enables ZigBee® nodes to be built with very low total bill-of material costs. The CC2430 combines the excellent performance of the leading CC2420 RF transceiver with an industry-standard enhanced 8051 MCU, 32/64/128 KB flash memory, 8 KB RAM and many other powerful features. Combined with the industry leading ZigBee® protocol stack (Z-Stack™) from Texas Instruments, the CC2430 provides the market's most competitive ZigBee® solution."

2.5 Zigbee Stack

ZigBee® is written in C, but uses its own stack: Z-Stack™ [23, 27].

Sleep modes

Power management is used by battery-powered end-devices to minimize power consumption between brief periods of scheduled activity (*LITE* sleep) or during long periods of inactivity (*DEEP* sleep) [17]. The Z-Stack™ provides two major sleeping modes, *LITE* sleep, and *DEEP* sleep. *LITE* sleep is used when the system needs to wake up periodically to perform an activity. *DEEP* sleep is used when no activity is scheduled and the system does not need to be woken up by a timer. This sleeping mode requires an external stimulus (e.g. a button press) to wake the system up.

The two major sleeping modes can be divided into four power modes, PM0 being the active mode, as shown in table 2.1.

Power Mode	High frequency oscillator	Low frequency oscillator	Voltage regulator (digital)	Wake up by timer	Typical current consumption
PM0	32 MHz XOSC 16 MHz RCOSC	32.768kHz RCOSC & 32.768kHz XOSC	On	-	Depending on activity
PM1	None	32.768kHz RCOSC & 32.768kHz XOSC	On	True	190 μA
PM2	None	32.768kHz RCOSC & 32.768kHz XOSC	Off	True	0.9 μA
PM3	None	None	Off	False	0.6 μA

Table 2.1: CC2430 Power Modes.

When in power mode PM1, the power consumption is reduced to a few mA, whereas power mode PM2 and PM3 can reduce the power consumption to a few μA. For power mode PM1 and PM2, the system can set a hardware timer or an OSAL timer to wake up. A 24-bit hardware timer, driven by a 32.768 kHz crystal clock, is typically used to extend the sleeping time. The sleep timer has a 24-bit counter and a 24-bit comparator. The longest sleep time is therefore 510 ($2^{24}/32768 = 512$) seconds (rounded). The OSAL uses a 16-bit timer structure. Therefore, the OSAL timer has a limitation of 65 ($2^{16}/1000$) seconds, based on 1 ms timer tick. [24]

To prevent the system from going into power mode PM2, one can use compiler option `PM1_ONLY`=TRUE [20] (which is not defined in [26]) to restrict the device to use PM1 only. This FALSE by default [24]. The `PM1_ONLY` option is evaluated in *hal_sleep.c*, where we can also see that the minimum (safe) sleeping time is 14 ms. There is no known compiler option to prevent the system from going into power mode PM3.

Sleep routine

If no task has an event scheduled, and power management capability is enabled, the system can decide to go into a sleeping mode. All of the following conditions must be met in order for the device to enter a sleep mode:

- Sleep enabled by the **POWER_SAVING** compile option [26];
- ZDO node descriptor indicates 'RX is off when idle'. This is done by setting `RFD_RCVC_ALWAYS_ON` to **FALSE** in *f8wConfig.cfg*;
- All Z-Stack™ tasks 'agree' to permit power savings, i.e., the power management state of every task is evaluated and should all be `PWRMGR_CONSERVE`. Every new task can add

  ```
  osal_pwrmgr_task_state( <task_id>, PWRMGR_CONSERVE );
  ```

  when it is ready to save energy. Since the power state of a new task is set to `PWRMGR_CONSERVE` by default [27], the task does not need to call this function if it always wants to converse power;
- Z-Stack™ tasks have no scheduled activity;
- The MAC has no scheduled activity.

When no activities are scheduled, `osal_pwrmgr_powerconserve` in *OSAL_Pwrmgr.c* puts the device to a sleeping mode, as shown in example 2.1. Here, one can see that the maximum sleeping time is calculated by evaluating when the first timer will expire.

It is also possible to put the end-device to sleep manually, but, since it is unclear if all task 'agree' and all buffers are flushed before the end-device is put to sleep, one should try to avoid this.

Example 2.1: Routine for saving power.

```
1  void osal_pwrmgr_powerconserve( void ) {
2    uint16 next;
3
4    // Should we even look into power conservation
5    if ( pwrmgr_attribute.pwrmgr_device != PWRMGR_ALWAYS_ON ) {
6      // Are all tasks in agreement to conserve
7      if ( pwrmgr_attribute.pwrmgr_task_state == 0 ) {
8        // Get next time-out
9        next = osal_next_timeout();
10
11       // Put the processor into sleep mode
12       OSAL_SET_CPU_INTO_SLEEP( next );
13     }
14   }
15 }
```

RS-232

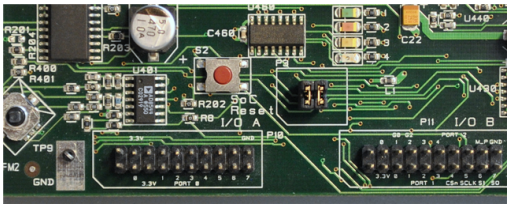
The SmartRF04EB we use for coordinator (see figure 2.2), is equipped with an RS-232 interface, by which it communicates with the GPRS module (see figure 3.1). RS-232 is a complete standard (defined in 1962) for serial communication. Unlike many standards, which simply specify the electrical characteristics of a given interface, RS-232 specifies electrical, functional, and mechanical characteristics. [7]

RS-232 is an old, but reliable standard and supports both software and hardware flow control. We are using the RS-232 standard to connect the ZigBee® coordinator with the GPRS module with a hardware flow control (CTS/RTS) and 115200bps baud rate. Table 43 of [31] indicates the commonly used baud rate settings for a 32 MHz system clock. The actual baud rates supported by the CC2430 differ from this table. They are defined in header file *hal_uart.h* of the Z-Stack™ , where we can see that the maximum baud rate supported is 115200bps.

The Z-Stack™ is equipped with a few sample applications that use the RS-232 (or serial) interface, e.g. *SerialApp* and *SampleApp*.

The application *SerialApp* functions as a cable replacement. A PC (or other device) sends data to the serial port on this application's device. This device transmits the message to another device with the same application running. The other device receives the over-the-air message and sends it to a PC (or other device) connected to its serial port. Using *SerialApp*, the SmartRF04EB can also be used as a bridge to connect a serial cable to a CC2430DB[1]. [11]

The application *SampleApp* uses a hardware jumper setting on P0_2 and P0_3 (see table 2.2) of I/O A of the SmartRF04EB (see table 2.2) to become either a coordinator or router, which is present by default [11, 19, 28, 29]. On the SmartRF04EB, the RS-232 interface is connected to I/O A [11] and also uses P0_2 and P0_3. Therefore, the RS-232 features in this build is disabled. The compiler will assert if one tries to enable it. In order to use the RS-232 feature fully, the jumper on I/O A must be removed and the default application *genericApp* should be used. Logically, if you leave this jumper on, it will result in a loop-back.



Pin	Function
9	P0_2 / UART_RD
11	P0_3 / UART_TD
13	P0_4 / UART_RTS
15	P0_5 / UART_CTS
20	GND

Table 2.2: SmartRF04EB I/O A pin-out.

The serial port uses a callback function to pass on events that were generated by the hardware (interrupts) or by polling mechanism (UART poll) to an upper layer. If these functions are executed in the context of an interrupt, it must be efficient and not perform CPU-intensive operations or use critical sections. [22]

Since the RS-232 communication can be configured to work on interrupts, it is assigned to its own task. After receiving the a message over RS-232, the coordinator decodes this message and puts the information in a struct (see example 2.2). In this struct, the message is saved until the end-device (with address `address16`) comes on-line and sends data.

Since end-devices should not poll for available messages every 100 ms (to save energy), putting the message on the regular queue would result in a time-out after several seconds and the message will not be sent. In this design, it will hold for as long as the end-device is not sending data and is therefore presumably unreachable.

After decoding the message, the struct is sent to the main task, as shown in example 2.3.

Example 2.2: `Struct` of a request for an end-device.

```
1 typedef struct {
2   osal_event_hdr_t hdr; // Event identifier
3   uint16    address16;   // 16b Address of the end-device
4   uint8     appDataLen;  // Length of the data
5   uint8     appData[ MAX_SAMPLE_STORE * SEQUENCE_LEN ]; // The data
6 } serialSysAppMsg_t;
```

Task Management

The Task Management API is used to add and manage tasks in the Operating System Abstraction Layer (OSAL) system. Each task is made up of an initialization function and an event processing function. The OSAL uses a task table to call the event processor for each task. [27]

The message management API provides a mechanism for exchanging messages between tasks, e.g. when a new message is received over RS-232 that holds a request for sequences, which is further processed by a different task. Function names in the *message management* start with `osal_msg_`. Example 2.3 shows the process of sending a `struct` from one task to another, whereas example 2.4 shows how such a message is received.

[1]Note that I/O A is used for serial communication and not I/O B as shown in picture 4 of [11]

Example 2.3: Sending an inter-task message.

```
1  // ... 'data' has been received and needs to be send to 'MainApp' ...
2
3  // Allocate a message buffer
4  serialSysAppMsg_t* msgToTask = (serialSysAppMsg_t*) osal_msg_allocate( sizeof(serialSysAppMsg_t) );
5
6  // If memory has been allocated, fill memory
7  if ( msgToTask ) {
8    msgToTask->hdr.event = SERIAL_SYS_APP_MSG; // Event identifier
9    msgToTask->address16 = nodeAddress16; // received from coordinator
10   msgToTask->appDataLen = dataLen;
11   for( int i=0; i < dataLen; i++ ){
12     msgToTask->appData[i] = data[i];
13   }
14
15   // Send msgToTask to task with id: 'MainApp_ TaskID'
16   osal_msg_send( MainApp_TaskID, (uint8 *)msgToTask );
17 }
```

Example 2.4: Receiving an inter-task message.

```
1  UINT16 MainApp_ProcessEvent( byte task_id, UINT16 events ) {
2    afIncomingMSGPacket_t *MSGpkt;
3
4    // If a message is waiting event, represented by SYS_ EVENT_ MSG
5    if ( events & SYS_EVENT_MSG ) {
6
7      // Fetch messages for task with id 'MainApp_ TaskID'
8      MSGpkt = (afIncomingMSGPacket_t *)osal_msg_receive( MainApp_TaskID );
9      while ( MSGpkt ) {
10
11       // Define which event is triggered
12       switch ( MSGpkt->hdr.event ) {
13
14         // Event by RS-232 task
15         case SERIAL_SYS_APP_MSG:
16           // Handle sequences (not in this example)
17           serialSysAppMsg_t *ssam = (serialSysAppMsg_t *)MSGpkt;
18           break;
19
20         // ... other cases are left out for brevity purposes ...
21       }
22
23       // Release the memory
24       osal_msg_deallocate( (uint8 *)MSGpkt );
25
26       // Fetch next message (if any)
27       MSGpkt = (afIncomingMSGPacket_t *)osal_msg_receive( GenericApp_TaskID );
28     }
29   }
30 }
```

3

GPRS module

The GPRS module (figure 3.1) reads data from the coordinator over a RS-232 Universal Asynchronous Receiver/Transmitter (UART) connection and saves the information in a local database per end-device with a RTC time stamp (its Real-Time Clock can be synchronized with the time-server of the GSM network). When it receives a data sample of an end-device, it looks for missing sequences and sends a request to the coordinator if a sequence is (or multiple sequences are) missing. The coordinator queues the message until the end-device is reachable again. When receiving enough data samples, the GPRS module can decide to open a GPRS connection to the server and send the data samples to the server in an XML format.

Figure 3.1: TC65 terminal.

3.1 Description

The Cinterion (formerly Siemens) TC65 terminal was selected because it meets more than all requirements needed for receiving data, temporarily store them and finally send them over GPRS to a server and has a variety of useful Java examples. Amongst various other features, the GPRS module is equipped with a SIM card interface, Quad-Band GSM, a 20Ω antenna, a GPRS multi-slot class 12, a TCP/IP stack, a serial interface, a I^2C bus, a SPI bus, J2ME™ profile IMP-NG, 400KB RAM, 1.7MB Flash and an ARM© Core Blackfin© DSP. [15]

At first, we tested a GPRS module of Wavecom Fastrack (obtained by Adrie van Doesburg), but this module did not came equipped with TCP stack on it. We decided to discard the Wavecom Fastrack and purchase the Cinterion TC65 terminal for this project.

3.2 Java ME

As mentioned above, the TC65 terminal supports J2ME™, which we used to program our application. Java Platform, Micro Edition (Java ME[1]) provides a flexible environment for applications running on mobile/embedded devices. Java ME includes security, built-in network protocols, and support for networked and off-line applications that can be downloaded dynamically. Applications based on Java ME are portable across many devices. The mobile and embedded Java platform (Java ME) was formerly known as Java 2 Platform, Micro Edition (J2ME™). The '2' has been dropped.

Since the hardware specifications of an embedded module are less than that of a regular PC, Java ME's API is reduced. This means that the programmer may be required to write methods himself that are available in Java, but not in Java ME.

The default packages of J2ME™ might not support the full extension of the module, which is also the case for the TC65 terminal. To use the full extension of the module, a Siemens package can be implemented.

3.3 Siemens package

The Siemens package [3] extends the features of Java ME for the execution of AT commands [13, 14], easy access of the flash file system of a wireless module, actuating on the signal state changes, capabilities for receiving bearer (e.g. GPRS/EDGE or CSD) state information (can also be done by AT commands), and the ability to control the CPUs hardware watchdog from application level. The methods used, e.g. *CommConnection*, are explained below.

[1] http://java.sun.com/javame

CommConnection

The interface *CommConnection* [3] defines a logical serial port connection. A 'logical' serial port is defined as a logical connection through which bytes are transferred serially. The logical serial port is defined within the underlying operating system and may not necessarily correspond to a physical RS-232 serial port. For instance, IrDA IRCOMM ports can commonly be configured as a logical serial port within the operating system so that it can act as a 'logical' serial port. Example 3.1 shows an example how to use *CommConnection*.

Example 3.1: How to use *CommConnection*.

```
1 // Open RS-232 with 115200 8-N-1
2 String strCOM = "comm:com0;baudrate=115200;bitsperchar=8;parity=none;stopbits=1;autocts=on;autorts=on";
3 CommConnection COMCon = (CommConnection) Connector.open(strCOM);
4
5 // Open streams
6 inStream = COMCon.openInputStream();
7 outStream = COMCon.openOutputStream();
```

3.4 ATCommands

AT commands are command line commands that can be sent to the module over RS-232. The 'AT' or 'at' prefix must be set at the beginning of each command line and a command line enter `<CR>` (or `\r`) must be set at the end to terminate. [14]

The *ATCommand* class supports the execution of AT commands on the same way, as it would be done through a serial interface. It provides a simple manner to send strings directly to the AT-Interpreters of the device. The command line command for activating our Java application is "`AT^JRA=a:/GatewayApp.jar\r`" (assuming that *GatewayApp.jar* and *GatewayApp.jad* are present in the root). *GatewayApp.jar* contains the actual Java code, whereas *GatewayApp.jad* contains parameters for this Java code. Using the *.jad* file, one can easily change parameters (e.g. the server's address, RS-232 settings, or SIM password) without having to recompile the Java code.

The *ATCommandListener* interface defines the capabilities for receiving unsolicited AT-Events (e.g. like "RING") and changes of the relevant incoming interface signals (RING, DCD and DSR). See subsection 3.4 Balance check for an example of ATCommandListener.

SIM card

Example 3.2 shows an example of an attempt to log on the SIM card. If the password (which is defined in *GatewayApp.jad*) is invalid, the number of retries is written into the log and the system will exit. Since the module does not have an LCD display or an adjustable LED pattern, the log is the only feature the module can use to register an error (assuming the UART is already connected to the coordinator). If the system failed to log in twice, the user still has one attempt left to log in the SIM card manually.

Example 3.2: SIM card initialisation.

```
1 ATCommand atc = new ATCommand( false );
2
3 // Request retries
4 String retries = atc.send("AT^SPIC\r");
5
6 if (Integer.parseInt(retries) > 1) {
7   // Enter PIN
8   str = atc.send("AT+CPIN=" + SIMPIN + "\r");
9   if (str.indexOf("OK") == -1) {
10    throw new Exception("Invalid SIM PIN: " + SIMPIN + ". Warning: " + retries + " attempts left!");
11  }
12 }
```

Balance check

Example 3.4 shows how to use the *ATEvent* trigger [13] for requesting the credit on the SIM card. The credit can be requested by calling a predefined phone number (in our case "`*101#`") and then, when the GSM network decides to send it, an *ATEvent* is triggered and a string is received containing the amount of credit left on the SIM card. Example 3.3 shows an example of an *ATEvent* by requesting the credit.

Example 3.3: AT event by AT commands.

```
Command
ATD*101#;\r

Responds
OK\r
+CUSD: 2,"Uw Prepaid TeGoed is Euro 7.38. Geldig tot 02/11/2010.",0
```

Balance check is only valid when the SIM card is prepaid, i.e. it does not have unlimited internet access and will prevent internet access if it has insufficient credit. If the SIM card is in a contract, the responds will be an error and the system will assume it has enough credit.

If the credit is less than €0.02, an error is written in the log and the application exits. There is (almost) insufficient credit left on the SIM card to be able to send a full data sample. The user will have to buy more credit and try again. In the future, it should also be possible to upgrade the credit by sending a message from the server.

Example 3.4: Processing an AT Event in Java.

```
1  ATCommand atc = new ATCommand(false);
2
3  // Listen to +CUSD for credit
4  ATListener listen = new ATListener("credit");
5
6  // Callback is activated
7  atc.addListener(listen);
8
9  // Call number to check balance
10 atc.send("ATD" + BlancePhonenumber + ";\r");
11
12 // Listener class
13 public class ATListener implements ATCommandListener {
14   String listenFor = "";
15
16   public ATListener(String waitingResponse) {
17     listenFor = waitingResponse;
18   }
19
20   public void ATEvent(String response) {
21     if (listenFor.equals( "credit" )) {
22       SystemModule sysM = new SystemModule();
23       sysM.setBalance(response);
24     }
25   }
26 }
```

Time synchronisation using NITZ

When a device re-attaches itself to the GSM network, when it enters a location area with different time zone or when a daylight change occurs, the GSM network can send a Network Identity and Time Zone (NITZ). This time synchronization using NITZ is used by most mobile phones to automatically update the system clock, as it is part of the official GSM standard since 1999. [16]

A NITZ indicator may consist of the following parameters: Universal Time (UT), local Time Zone (TZ), and Daylight Saving Time (DST). From the NITZ, we can extract the current date and time and create a time stamp, using Java's Calendar class.

The NITZ can be looked up by sending AT command "`AT^SIND=nitz,2\r`" [14]. Example 3.5 shows that the returned value might be out of date when the device did not re-attach itself to the GSM network. In

Example 3.5: Execute NITZ with and without re-attaching.

```
NITZ command without re-attaching
AT^SIND=nitz,2\r

Responds
^SIND: nitz,0,"10/01/21,10:42:04",+04,0

NITZ command after re-attaching to the GSM network
AT+CGATT=0\r
AT+CGATT=1\r
AT^SIND=nitz,2\r

Responds
OK\r
OK\r
^SIND: nitz,0,"10/01/21,10:50:02",+04,0
```

this example, TZ is +04, showing a time offset of +1 hour (east) to Universal Time/GMT. DST is 0, which indicates that no hour was added to TZ because of Daylight Saving Time (summer/winter time). [14]

The internal clock of the module can be modified, but once up and running, the J2ME™ internal clock cannot. In most cases, the current time and date is calculated by calculating how many milliseconds have passed since January 1 1970. Using the same technique, we can count the milliseconds since the last synchronization. As shown in example 3.6, we can store the milliseconds of Java's Date (represents January 1 1970, 00:00:00 + the system up time in milliseconds) and the milliseconds representation of the current date and time (e.g. January 21 2010, 11:50:02). When in the application a time stamp is needed, we can count how many milliseconds have passed since the last synchronization + the milliseconds of the actual date and time of the last synchronization.

Example 3.6: Getting the current time stamp in Java after time synchronization.

```
1  long static timestampSync = 0;
2  long static timestampLastSync = 0;
3
4  public static void setDate(long timestamp){
5      timestampSync = timestamp;
6      timestampLastSync = new Date().getTime();
7  }
8
9  public static Date getDate() {
10   return new Date(timestampSync + (new Date().getTime() - timestampLastSync));
11 }
```

3.5 Logging

The logging mechanism attempts to write to a file on the flash memory by using Siemens' *FileConnection* class. When in debug mode, the GPRS module holds a record of data received and sent with the current time stamp on its flash memory, as shown in example 3.7. Since this is an embedded module and therefore has limited memory, the log can easily overflow if not handled properly. Before the data is written in the log file, the available space on the flash memory is determined. The data to be sent over GPRS is also stored on the flash memory and therefore, 10KB is preserved for the GPRS file to grow. When there is insufficient space available, the log file's size is reduced by half, leaving only the latest half behind (based on the First In First Out principle). When not in debug mode, only exceptions in Java are logged.

Example 3.7: Data logged.

```
1  2010/01/21 17:17:07
2  Balance is Euro 7.38
```

3.6 Threads

In our application, the GPRS module uses three threads for internal, UART, and GPRS communication, called Main, UART and GPRS.

Main

The main thread is used to initialise the application, temporarily store and detect missing sequences, prepare these sequences for server interpretation and to wake up the GPRS thread when a package is ready to be sent. Example 3.8 shows in pseudo code the steps of the main thread.

In the initialisation stage, the main thread registers the SIM card to the GSM network (see subsection 3.4 SIM card) and calls a predefined number to determine how much credit is left on the SIM card (see subsection 3.4 Balance check). Next, the time is set by re-attaching itself to the GSM network (see subsection 3.4 Time synchronisation using NITZ).

The main thread is notified when the UART thread has received data. It fetches this data and tries to extract the sequence using the method *readFromCoordinator()*.

When a sequence contains more than one data sample, the length of the message increases and the data length retained from the UART does not necessarily contain a full sequence (since this depends on the amount of data samples stored in it). Therefore, once the amount of data samples is determined and the current length is insufficient, *readFromCoordinator()* might have to hold until the remaining data is retained.

After receiving a data sample, the latest 300 sequences (this number can be modified) are verified. Due to the low capacity of the ZigBee® end-devices, it is improbable that it has stored more than 300 sequences. If one sequence is or more sequences are missing, a message is created and sent to the coordinator.

Next, the internal database is consulted if it has more than 10 new data samples (in total). If so, it fetches these data samples, converts them into XML (see section 5.3 GPRS module to server) and the 'new' flag for these samples is unset.

Example 3.8: Main thread in pseudo code.

```
1  initialisation ();
2
3 createThreads();
4
5 while( true ){
6   // Sleep until notified or after 5 minutes automatically
7   wait( 5 * 60 * 1000 );
8
9   if (uART.getMsgReceivedSize() > 0) {
10    readFromCoordinator( uART.getMsgReceived() );
11
12    checkMissingSequences();
13  }
14
15  if ( database.getNewSequencesSize() > 10) {
16    List nodeList = database.getNewSequencesOfAllNodes();
17
18    String txtToFile = "";
19    foreach( nodeList AS tempNode )
20      txtToFile += gprs.formatNodeData(tempNode);
21
22    // Add a final encapsulation and time stamp in XML to be sent
23    txtToFile = gprs.formatToXML(txtToFile);
24
25    gprs.appendToOutputBuffer(txtToFile);
26
27    database.optimize();
28
29    notifyGprs();
30  }
31 }
```

UART

The UART thread opens a RS-232 UART connection with a baud rate of 115200bps, 8-N-1, with hardware flow control (using RTS/CTS [2, 7], see figure 3.2[2]). It is communicates to the ZigBee® coordinator over a null modem cable [2].

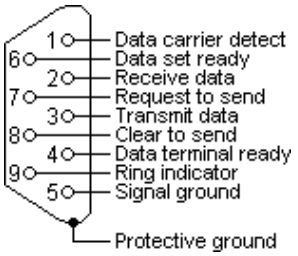


Figure 3.2: DB9 pin out.

First, it sends the pending messages over UART (if any). Then, it reads the bytes available from the UART and appends it to its buffer.

After storing a sequence into its buffer, the main thread is notified that it can interpreter the message. If the main thread detects a missing sequence, it assembles a message that orders the coordinator to send a request to an end-device and puts this in the outgoing buffer (more about this communication in chapter 5.4 GPRS module to coordinator). If no bytes are available to receive, the thread is put to sleep for 10ms to save system resources.

In the original design, the UART thread also interpreted the message, but the interpretation took too long (more or less 400ms to 700ms) and since the coordinator has a small RS-232 buffer (by default 128 bytes), it can therefore easily overflow. In the current design, the UART thread is for the UART communication only.

GPRS

The General Packet Radio Service (GPRS) is a data extension of the mobile telephony standard GSM that is emerging as the first true packet-switched architecture to allow mobile subscribers to benefit from high-speed transmission rates and run data applications from their mobile terminals [4], e.g. a mobile phone or data terminal.

The GPRS thread is used for communicating with the server and is only running if the main thread notified it or after one minute automatically. The communication for GPRS needs to be in a separate thread since it uses the slow flash memory frequently (where the data to sent is stored) and, more importantly, the system will hold for 30 000 ms (set by the GSM network, in our case, Vodafone), waiting for a time-out while trying to open the GPRS port, risking to miss sequences sent by the coordinator. Reducing the time-out value by setting the 'timeout' parameter option is not allowed by Vodafone. Sending a 'keep alive' is not allowed by the module [3]. Hence, when new data is ready to be sent, a new connection has to be set up.

If there is data to sent, the thread attempts to open a port to the server over GPRS and send all data through a socket connection, shown in example 3.9. The main thread assembles XML of the sequences of

Example 3.9: Open a socket connection to a server over GPRS.

```
1 SocketConnection sc = (SocketConnection) Connector.open("socket://145.89.131.225:9876;bearer_type=gprs;"
2                             + "access_point=office.vodafone.nl;username=vodafone;password=vodafone");
3
4 OutputStream outStream = sc.openOutputStream();
```

each node (see example 3.10). The data itself is converted into hexadecimal to save characters. The outcome

Example 3.10: XML data samples of each node.

```
1 <end MAC="DFDEE1186F6F5">
2   <seq id="0" tos="FDE8" tor="126568588EB"><s id="4" d="0,112" off="FDE8" /></seq>
3   <seq id="1" tos="1FBD0" tor="126568686D3"><s id="4" d="0,112" off="1FBD0" /></seq>
4   <seq id="2" tos="2F9B8" tor="126568784BB"><s id="4" d="0,112" off="2F9B8" /></seq>
5 </end>
```

is encapsulated with information about the coordinator, i.e., its 64b IEEE address, where after it is sent to the server. Then, the output buffer is updated by the number of characters sent (since it can already be modified), based on the First-In-First-Out principle (FIFO).

[2]http://www.lammertbies.nl/comm/cable/RS-232.html

4

Data samples from end-device to server

In this chapter, the outcome of various meetings is discussed. One of the first thing that became clear in these meetings, was that the server needs some information only once, e.g., 64b IEEE address of the ZigBee® coordinator (should never change) and address of the end-device's parent device, the closest router / coordinator (for localisation of the node). Because this information does not have to be sent every time the GPRS module sends data to the server, we chose to use to format the data in XML. XML is well ordered (if designed well), easily readable for humans, and allows changes (which makes it generic). However, the overhead for using XML is significant and, since the WSN is designed to be energy efficient and the XML protocol requires a great amount of characters, therefore not used within the WSN.

4.1 Socket connection

When the GPRS module is connected to the server over GPRS, several types of connection can be chosen, e.g. HTTP[1], SSL[2], or a socket connection[3]. SSL is the most secure of the three, HTTP is by default supported by both the GPRS module and Seam framework[4] used by the server, and a socket connection would have the least overhead. In a meeting with employer Erik Puik, it was decided that the communication itself is the most important in this project and that features, such as security, can be recommended for future work (see section 6.4 Add security). The socket connection [3] was chosen because it has the least overhead and has the benefit of being the cheapest of the three.

4.2 Communication is not incidental

All end-devices have their own clock that holds a timer when to send the next data sample(s). This makes the communication periodic and not incidental. Although the end-device will only send data at a periodic time, it does not have to send it, i.e., it should not waste its energy by sending redundant data. In case that the difference between the current data and the last sent data is below a specific threshold, the end-device should not send the data, but will retry at the next periodic interval. This threshold should be adjustable afterwards (see section 6.3 Send new parameters to end-devices).

4.3 Two way communication

In another meeting with our employer, we discussed the different scenarios to get data from the end-devices:

1. automatically, when the end-device wakes up and decides it has relevant data;
2. upon request by the GPRS module, when it asks for a specific data set;
3. upon request from a mobile router (e.g. a hand-held) that can join the network at any time and ask for a specific data set or for all data available.

The end-device sends its data samples to the coordinator, which will send it to the GPRS module, and the GPRS module will, on its turn, send it to the server. It is clear that down-up communication is necessary.

Top-down communication on the other hand, will increase the network-load, complexity of the protocol and decrease battery life, i.e., the end-device has to enable its radio for an extended period without knowing if it will have to receive packages at all. One should think twice before adding the top-down communication as a requirement. However, the situation might occur that the GPRS module is missing a sequence, and will have to send a request to the end-device for it. In addition, to make the application of the end-device more flexible, certain values can be parameterized, e.g., the minimum time to wait before sending new values. To support these features, a two-way communication is a necessity.

[1] see http://www.w3.org/Protocols/rfc2616/rfc2616.html
[2] see http://download.oracle.com/docs/cd/E12531_01/tuxedo100/pdf/security.pdf
[3] see http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/io/SocketConnection.html
[4] see http://seamframework.org

4.4 Time synchronization in an energy efficient WSN

Time synchronization is a common requirement for a WSN for an accurate analysis and real-time event monitoring, but is typically energy inefficient and heavy-weighted.

If each node in a hierarchic WSN is capable of broadcasting packages, the lightweight synchronization protocol LEvel Synchronization by Sender, Adjuster and Receiver (LESSAR) can be used [35]. LESSAR is a protocol that divides a WSN into levels. The nodes at level $i+1$ are *receivers* that synchronize their local clock with the node at level i (their *sender*). By promoting one of the receivers to an adjuster and perform a two-way message exchange, the sender can estimate the delay time (Process, Access, Propagation and Receive delay, PADR [35]) and updates the time synchronization package with this delay time, which it will then send to every receiver. By selecting just one of the receivers to estimate the time delay, it decreases the time synchronization packets that are required.

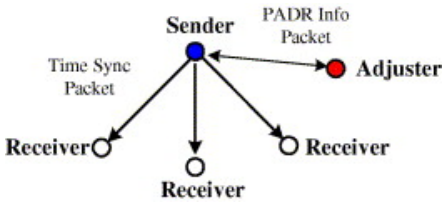


Figure 4.1: LESSAR time synchronization protocol.

Another, more energy efficient, but less accurate, method is to add the time stamp at the first node that has a Real-Time Clock (RTC) [9]. A network router with RTC hardware capabilities may have to insert all the time stamps if none of the nodes has the mechanism for doing so. Otherwise, the insertion of the time stamp is left to the network coordinator or, in our case, the GPRS module. This method assumes that a sample it taken and directly transmitted to the network coordinator and ignores the PADR delays.

When designing the data structure for transmitting the data samples, research assistant Dennis van Wijk pointed out that it should (in the future) also be possible to take multiple samples over time and send them all together (which is more energy efficient), i.e., the time stamp of measurement and time stamp of sending the sample(s) might be different. Having this in mind, we created a data structure where the end-device's local (not synchronized) clock is used (see figure 5.3 on page 19).

4.5 Data retrieval

As mention before, being able to retrieve data samples is an important, but heavyweight requirement in this project, which affects all devices in the network:

- The end-device must hold a list of sequences, keep record which sequences are sent, keeping the radio on-line for a longer period (to receive a request for an old sequence), and be able to resend all requested sequences if it is still available.
- The coordinator must hold a list of sequences that are to be requested to an end-device when it comes on-line, be able to handle the data flow when it receives several sequences at once, and accept a request for sequences from the GPRS module.
- The GPRS module must hold a record of the latest sequences receives of every end-device, look for missing sequences, and request the missing sequence only once. The list of sequences can be updated when the sequences are sent to the server, but it should prevent an underflow. When for example, previous requested sequence 100 is received and the list of $90-130$ is updated (cleared) to 120, it will request every sequence between 100 and 120, which are received before and already sent to the server. Therefore, the list can only be updated to the last requested sequence (if any).

In the GPRS module, the data in XML is written in the 1.7MB flash memory and the redundant sequences are removed from the list in the 400kB RAM [15]. This is to prevent the system from running

out of memory and prevent the system's performance to lack, i.e., without removing redundant sequences from the list of sequences, the list will grow and thus the time it will take to find missing sequences will increase. In addition, when a power outage occurs, the data to be sent to the server is preserved and will be sent when the application is operational again. However, the list of sequences will then be empty. When the first new sequence of an end-device is received, the 300 previous sequences will be requested, since the system assumes it did not receive them.

5

Communication in detail

In figure 5.1, one can see the model of the communication between the several devices[1], which consist of sequences and requests for sequences. Furthermore, it shows how each device communicates with actors and other devices through I/O devices and which events are handled [32]. In this chapter, the data structure of the communication between the various devices is discussed.

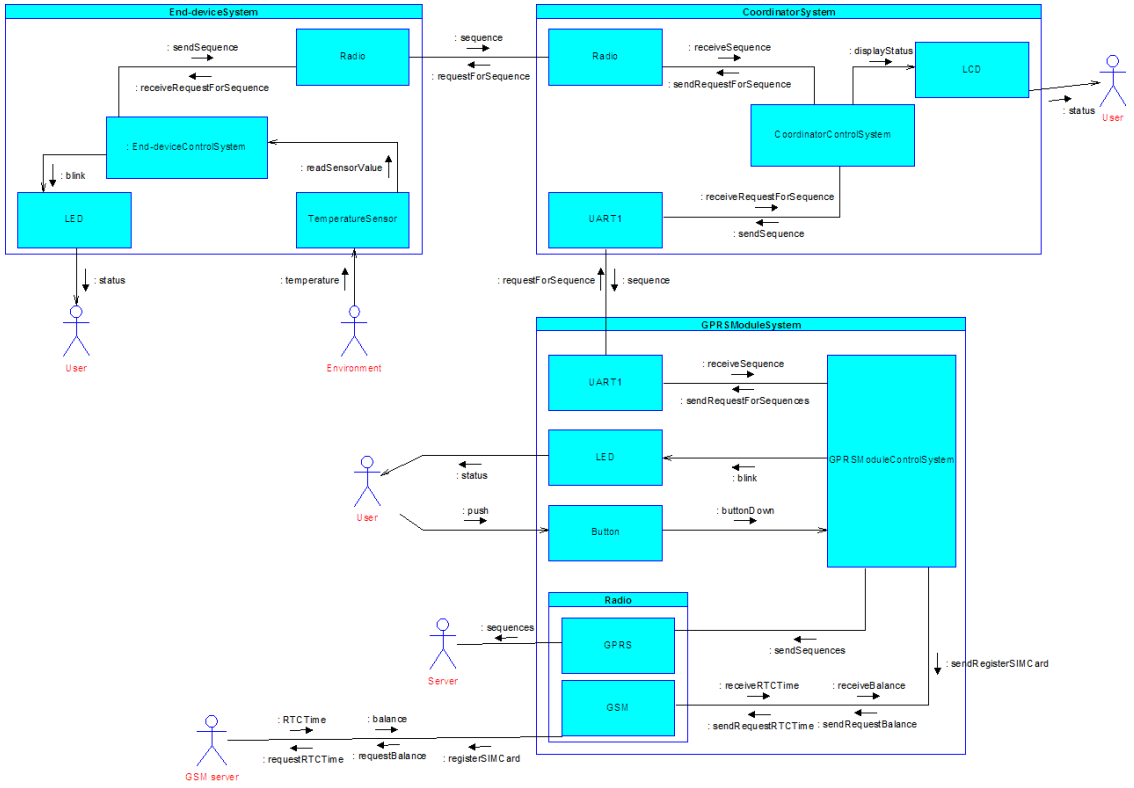


Figure 5.1: Context diagram of the communication between the various devices.

5.1 End-device to Coordinator

Since an end-device has a limited power supply and wireless communication is very power consuming, its communication to the coordinator should be as short and seldom as possible, but the information transferred to the server needs to be sufficient to be able build a model and be generic for possible extensions in the future. Figure 5.2 shows which part of the context diagram in figure 5.1 is discussed.

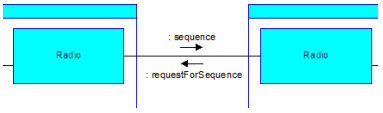


Figure 5.2: Send sequence.

One can take multiple samples before sending them. These data samples can be aggregated in a sequence. Such a sequence consists of:

- a time stamp of when the data is sent;
- a sequence number;
- the amount of data samples in the sequence, each containing:
  - when it was taken;

[1] A larger version of this context diagram, as well as several other models, can be seen in appendix A.

- which sensor was polled;
- the length of the sample;
- the sample itself.

Since the internal clock of an end-device is not synchronized, the actual time when the sample was taken must be computed by the server. As shown in figure 5.3, data samples can be taken without directly sending them to the coordinator afterwards. Hence, to be able to compute when the sample was taken, the sequence needs a time stamp when the data was sent. In addition, a sequence can be requested later on, making a time stamp in de sequence necessary.

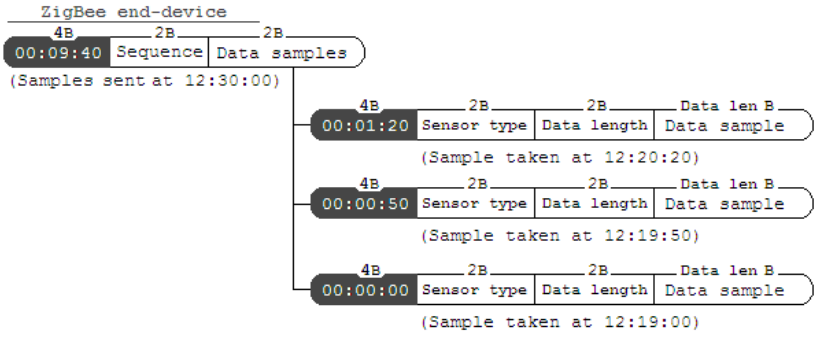


Figure 5.3: Data structure of a sequence, containing three data samples, that is sent to the coordinator.

5.2 Coordinator to GPRS module

When a sequence from an end-device is received by the coordinator, the coordinator looks up the unique 64b IEEE address of the end-device using `APSME_LookupExtAddr()`. `APSME_LookupExtAddr()` will lookup the address in the Address Manager and does not start a network (over-the-air) IEEE lookup [25]. The 64b address is used, since the 16b network address can alter when the end-device binds to a different router. Figure 5.4 shows which part of the context diagram in figure 5.1 is discussed.

Figure 5.5 shows the data representation of each message of a sequence, sent from the coordinator to the GPRS module over RS-232. This message consists of:

- a prefix, two bytes (0xAA and 0x55, together equal to 0xFF) to indicate the start of the transmission;
- the length of the entire package;
- the IEEE address of the coordinator, required by the server.
- the number of devices in this message (usually just one), each containing:
  - the unique 64b IEEE address of the end-device;
  - the message received by the end-device(s).

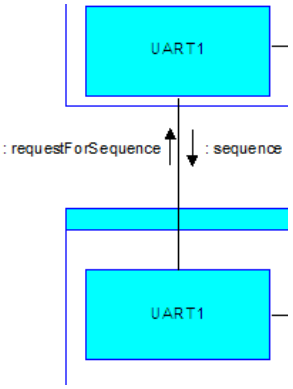


Figure 5.4: Send sequence.

5.3 GPRS module to server

Figure 5.6 shows which part of the context diagram in figure 5.1 is discussed; the GPRS module sends sequences to the server.

To make the communication from GPRS module to server as generic as possible, the data is structured in XML, as shown in example 5.1. The values of each segment are formatted into hexadecimal, to save characters.

In the XML scheme, the following elements and attributes are used:

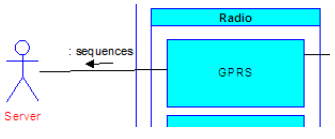


Figure 5.6: Send sequences.

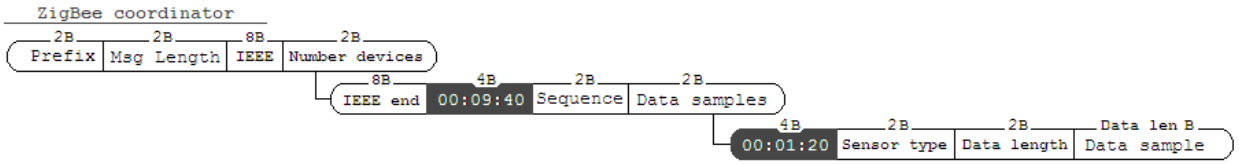


Figure 5.5: Data structure of data samples sent to the GPRS module.

- `<?xml version="1.0" encoding="UTF-8"?>` is added as a prefix to generate a valid XML scheme;
- `net` is the network coordinator:
  - `id` is the 64b IEEE address of the coordinator;
  - `time` is the RTC time stamp of the GPRS module when this XML code is generated;
- `end` is an end-device:
  - `MAC` is the 64b IEEE address of the end-device;
- `seq` represents a sequence sent by the end-device:
  - `id` is the sequence number;
  - `tos` (Time Of Sent), the (unsynchronized) time stamp of the end-device when it sent the sequence;
  - `tor` (Time Of Receive) represents the RTC time stamp when the GPRS module received the message. This is used to compute when the samples were taken;
- s represents a data sample of a sensor:
  - `id` represents the sensor id;
  - `d` represents the data of the sensor, separated by a comma, since at this point, it is uncertain how the actual value should be calculated. There could be several sensors on the end-device, each with s different data representation;
  - `off` represents the offset (time stamp) of when the sample was taken.

Example 5.1: Data structure example of four sequences that are sent to the server.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <net id="180E5EE62CB8" time="1265686E2BA">
3   <end MAC="DFDEE1186F6F5">
4     <seq id="0" tos="5910" tor="126568588EB"><s id="4" d="0,112" off="5910" /></seq>
5     <seq id="2" tos="A7C0" tor="1265685B79C"><s id="4" d="0,112" off="A7C0" /></seq>
6     <seq id="4" tos="F605" tor="1265685E5D2"><s id="4" d="0,108" off="F605" /></seq>
7     <seq id="1" tos="F67A" tor="1265685E9EE"><s id="4" d="0,112" off="5910" /></seq>
8   </end>
9 </net>
```

As one can see in example 5.1, the sequence numbers are not ordered by number, but by the time that they were received by the GPRS module. In the test phase, all end-devices only sent the even sequences and store the uneven sequences. The GPRS module notices that there are sequences missing and requests them through the coordinator. In this example, one can see that sequence `1` has been requested and received. This can be verified by looking at the `tos` and `off` attributes. Sequence `0`, `2` and `4` are sent directly after taking the sample. Therefore, the `tos` is equal to the `off`. Sequence `1` however, shows a higher `tos`, which means that it was sent later on.

5.4 GPRS module to coordinator

Figure 5.7 shows which part of the context diagram in figure 5.1 is discussed; sending a request for sequences to the coordinator, as shown in figure 5.8. Since the WSN uses the 16b address and not the 64b IEEE address, this address needs to be looked up (if known). The coordinator uses the `APSME_LookupNwkAddr()` function to lookup the address in the Address Manager [25]. The coordinator then stores the message in its buffer until the end-device wakes up (see subsection 2.5 Task Management).

Figure 5.8 shows the data representation of the transmission from the GPRS module to the coordinator over RS-232. The message consists of:

- a prefix, two bytes to indicate the start of the transmission;
- the length of the address (can also be used as message identifier);
- the IEEE address of the end-device.
- the number of sequences in this message, each containing:
  - the sequence number.

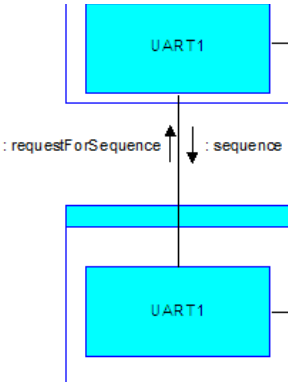


Figure 5.7: Send request for sequence.

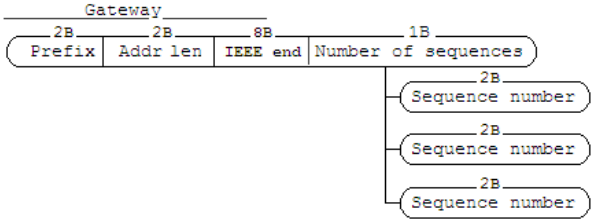


Figure 5.8: Data structure of a request for (a) sequence(s) from the GPRS module.

5.5 Coordinator to end-device

Figure 5.9 shows which part of the context diagram in figure 5.1 is discussed; send a request for (a) sequence(s) to the end-device.

When an end-device sends a message to the coordinator, the coordinator sends any pending messages back, before processing the received message. This is to prevent that the end-device is already going to sleep before receiving a message from the coordinator. The unique message identifier in `AF_DataRequest` (which is used to send the message) indicates that this message is a request for sequences. Hence, the message itself only has to contain the requested sequence numbers as shown in figure 5.10.

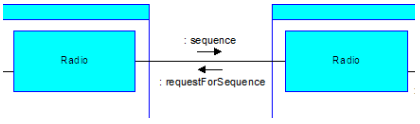


Figure 5.9: Send request for sequence.

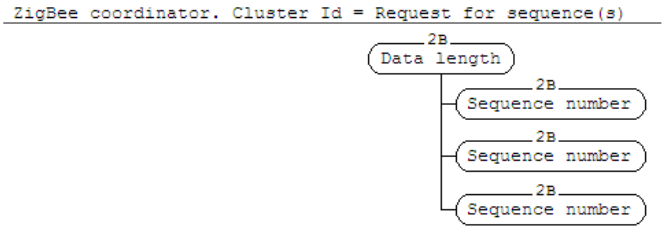


Figure 5.10: Data structure of a request for sequences from the ZigBee coordinator.

6

Conclusion and recommendation for future work

To embed sensor devices into concrete to monitor the concrete's compressive strength during its curing process and to send the samples to a server for analysis is an innovative and challenging approach. It requires many embedded devices, each with their own capabilities and limitations, to work together, using a variety of protocols to reach a united goal: provide as many data samples to the server as possible and keep the sensor devices alive for as long as data samples are needed.

For the author, this meant facing many challenges, including choosing the right GPRS module for this project with enough potential for future projects, getting familiar with various protocols in a variety of programming languages, and selecting and implementing the most efficient time synchronization method, which is a lot for one person in the short time span planned for this project.

Monitoring the concrete during its curing process is just one of the many environments where this remote WSN can be used. It has a great potential for the future. The research centre for product development can assign new (honour) students to continue and improve the current remote WSN.

6.1 Credit upgrade over GPRS

To upgrade the credit on a prepaid SIM card, one has to call a specific phone number and enter a unique credit code. The GPRS module also supports this feature, through AT commands (see section 3.4). Currently, this can only be done by removing the SIM card, putting it in a mobile phone, and entering the unique credit code there. This can be automated by providing the unique credit code to the server, which sends it to the GPRS module. The GPRS module can be configured so, that it calls this predefined phone number and enters the unique credit code by itself. Of course, this all is not necessary when the SIM card has unlimited internet access.

6.2 Battery level indicator

When the server receives the battery level of an end-device, it can calculate when the battery will die and therefore, how long the end-device can send data samples. If, with the current settings, the battery dies sooner than intended, it can decide to send new parameters, e.g., to take samples less often, which will prolong the battery life. If, with the current settings, the battery is still alive long after data samples are needed, the server can decide to send new parameters, e.g., to take samples more often. Moreover, it can optimise the data flow by the end-device's battery.

Sending the battery level with every message creates an unnecessary overhead. One has to determine how often the battery level needs to be sent. This feature is not implemented, but it has a great potential.

6.3 Send new parameters to end-devices

Currently, it is not possible to send new parameters to an end-device. This is partially because our end-device is still a prototype and its hardware can therefore change in the next prototype, and also because this was not the focus of this project.

To implement this feature, all parametrizable variables need to be determined first. A few examples are:

- How many samples to take before sending a sequence;
- How often to poll/read a specific sensor;
- Threshold of difference in value to the previous sample before transmitting this sample (discussed in section 4.2 Communication is not incidental);

- When to send the battery level.

Then, the coordinator must hold a record of messages to send to the end-device when it comes on-line. This is currently only implemented with requests for sequences.

6.4 Add security

Currently, both the ZigBee® and GPRS network are not secured, i.e., data over these networks are not encrypted and any device will be allowed to join the network when they request it. Compiler option `SECURE` [26] can enable ZigBee® security (`SECURE`=0 to disable, `SECURE`=1 to enable), but this is also defined as `-DSECURE`=0 in *f8wConfig.cfg*. How to enable security in the network and how secure the network becomes needs further investigation.

The communication between the GPRS module and the server is currently based on an unsecured socket connection. This can be upgraded to SSL, which is more secure.

6.5 Prevent UART transmit buffer overflow

Both the transmit and receive buffer for serial communication on the ZigBee® coordinator is by default 128 bytes and will trigger an interrupt when one tries to write more data to this buffer, which can happen if the coordinator receives multiple sequences at a time and the buffer is not flushed in time. At first, it was believed that the GPRS module was reading its incoming serial buffer too slowly. The UART thread (see UART on page 14) sleeps for 5ms to 10ms and that this short period was still too long to cause the coordinator's transmit buffer to overflow.

Later, while testing the ZigBee® coordinator's capabilities with terminal application RealTerm[1], we discovered that the coordinator was sending an incomplete sequence, due to a buffer overflow, i.e., after receiving 128 characters, the communication was stopped and was resumed after the coordinator received a new sequence.

One way to prevent a buffer overflow, is to increase the buffer's size. When testing to increase both transmit and receive buffers, the CTS (Clear To Send) did not produce a solid signal, and no communication was possible. In `hal_uart.c`, a compiler option is evaluated called `HAL_UART_BIG_TX_BUF` (which is not defined in [26]), which allows the transmit buffer to be increased to a large size. When setting `HAL_UART_BIG_TX_BUF=TRUE` in the compiler options, it should be possible to increase the transmit buffer over 255 bytes. Further research is needed if increasing just the transmit buffer's size (and adding the `HAL_UART_BIG_TX_BUF` compiler option) will produce a solid signal and stop the transmit buffer to overflow.

6.6 Energy-efficiency

When an end-device decides to go to *LITE* sleep, it looks for the first timer to expire. This defines how long it can sleep. Currently, it is only possible to sleep for about 50ms, which is not very energy efficient, considering that the end-device is taking a sample only once a minute. Further investigation is needed why the end-device is not sleeping longer. It is believed that the short sleeping time is caused by the key polling rate (100 ms by default) and network polling rate settings, defined in *f8wConfig.cfg*:

- `-DPOLL_RATE`=1000, the number of milliseconds to wait between data request polls to the coordinator.
- `-DQUEUED_POLL_RATE`=100, defines how many milliseconds to wait after receiving a data confirmation to poll for **queued** messages;
- `-DRESPONSE_POLL_RATE`=100, defines how many milliseconds to wait after receiving a data confirmation to poll for **response** messages;
- `-DREJOIN_POLL_RATE`=440, used as an alternate response poll rate only for rejoin request. This rate is determined by the response time of the parent that the device is trying to join.

[12, 24] confirm that the power consumption is strongly influenced by the poll rate (and the channel conditions), and that the battery lifetime may be extended by decreasing the poll rate, when this is acceptable in the application.

[1] http://realterm.sourceforge.net

6.7 Optimisation for large networks

The ZigBee® coordinator is currently capable of storing 20 requests for sequences for 10 individual end-devices. The memory space for these requests is reserved at compiler-time and not at run-time. This is a perfect testing environment, but the memory space should be allocated at run-time when it goes into production. To increase the number of end-devices that can be held in the requests for sequences list to `MAX_ENDPOINTS` (defined in *ZDConfig.h*), the coordinator requires additional memory.

The feature of being able to request older sequences (and their data samples) is a heavy-weighted requirement that affect all embedded devices in the model. The feature was required so that the server can build an accurate analysis, but how much data does the server actually need to build this analysis and is it not possible that the server uses algorithms to calculate these missing data samples, rather than requesting them?

Bibliography

[1] Baronti, Paolo, Prashant Pillai, Vince W.C. Chook, Stefano Chessa, Alberto Gotta, and Y. Fun Hu: *Wireless sensor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards*. Computer Communications, 30(7):1655 – 1695, 2007, ISSN 0140-3664. http://www.sciencedirect.com/science/article/B6TYP-4MP569D-2/2/2cb7b0fa0bd9d0dec76e4702a4d76937, Wired / Wireless Internet Communications.

[2] Bies, Lammert: *RS232 serial cable pinout information*. http://www.lammertbies.nl/comm/cable/RS-232.html.

[3] Cinterion: *Api*. Javadoc, Cinterion, April 2008.

[4] Ghribi, Brahim and Luigi Logrippo: *Understanding GPRS: the GSM packet radio service*. Computer Networks, 34(5):763 – 779, 2000, ISSN 1389-1286. http://www.sciencedirect.com/science/article/B6VRG-41BV9HN-5/2/1f3fba4f4368a040989e51f9de1243e8, Future Wireless Networks.

[5] IEEE Computer Society, LAN/MAN Standards Committee of the: *Telecommunications and information exchange between systems- local and metropolitan area networks- specific requirements–part 15.4: Wireless MAC and PHY specifications for low-rate WPANs*. Technical Report 802.15.4-2006, IEEE Standard for Information technology, 3 Park Avenue, New York, USA, September 2006. http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf.

[6] Janssen, M.A.: *Over-the-air download using ZigBee*. Technical report, Utrecht University of Applied Sciences, Utrecht, NL, July 2008. Unpublished technical report.

[7] Maxim: *Fundamentals of RS-232 serial communications*. Application note AN83, Maxim, March 2001. http://www.maxim-ic.com/an83.

[8] Microchip: *Microchip stack for the ZigBee protocol*. Application note AN965 rev. C, Microchip, San Diego, CA USA, January 2007. ww1.microchip.com/downloads/en/AppNotes/00965c.pdf.

[9] Morais, Raul, Miguel A. Fernandes, Samuel G. Matos, Carlos Serôdio, P.J.S.G. Ferreira, and M.J.C.S. Reis: *A ZigBee multi-powered wireless acquisition device for remote sensing applications in precision viticulture*. Computers and Electronics in Agriculture, 62(2):94 – 106, 2008, ISSN 0168-1699. http://www.sciencedirect.com/science/article/B6T5M-4RN4862-1/2/cb34c3133c13c5d7158a96ab96fb193c.

[10] Norris, Ashley, Mohamed Saafi, and Peter Romine: *Temperature and moisture monitoring in concrete structures using embedded nanotechnology / microelectromechanical systems (MEMS) sensors*. Construction and Building Materials, 22(2):111 – 120, 2008, ISSN 0950-0618. http://www.sciencedirect.com/science/article/B6V2G-4KYY3PS-7/2/a7ef7fb2a8ad88ab92273bfe74ab57d3.

[11] Selvig, B.: *Z-Tool*. Application note AN045, Texas Instruments, San Diego, CA USA, December 2006. http://www.ti.com/lit/pdf/swra119.

[12] Selvig, B.: *Measuring power consumption with CC2430 and Z-Stack*. Application note: AN053 (Rev. 1.0) SWRA144, Texas Instruments, San Diego, CA USA, July 2007. http://www.ti.com/lit/pdf/swra144.

[13] Siemens: *Java user's guide siemens cellular engines*. Data sheet wm-java-usersguide-v12, Siemens, February 2008.

[14] Siemens: *TC65 AT command set*. Data sheet TC65-ATC-V03.000, Siemens, February 2008.

[15] Stieglitz, Stephanie: *TC65T datasheet*. Data sheet tc65t Datasheet, Cinterion, Germany, November 2006. http://www.cinterion.com/tl_files/cinterion/downloads/tc65t_intranet_291002.pdf.

[16] Sultan, Alain: *Global roadmap for SMG and sa plenary wis*. Information TSG S14 (99)436, ETSI Sophia Antipolis, Miami, USA, July 1999.

[17] Sundet, Torgeir: *Dn106 – power modes*. Design note: SWRA162, Texas Instruments, San Diego, CA USA, November 2007. http://www.ti.com/litv/pdf/swra162b.

[18] Tammer, M.D.: *Wireless monitoring of the concrete curing process*. Master thesis, Utrecht University of Applied Sciences, Utrecht, NL, September 2009.

[19] Texas Instruments: *CC1110DK,CC2430DK,CC2510DK development kit*. User manual SWRU039B, Texas Instruments, San Diego, CA USA, October 2006.

[20] Texas Instruments: *CC2430 revision D, power modes*. Application note draft AN044, Texas Instruments, San Diego, CA USA, December 2006.

[21] Texas Instruments: *CC2430DK development kit*. Description SWRD040A, Texas Instruments, San Diego, CA USA, October 2006. http://www.ti.com/lit/pdf/SWRD040A.

[22] Texas Instruments: *HAL Drivers, Application Programming Interface*. API F8W-2005-1504 v1.2, Texas Instruments, San Diego, CA USA, November 2006.

[23] Texas Instruments: *802.15.4 MAC, Application Programming Interface*. API F8W-2005-1503 v1.1, Texas Instruments, San Diego, CA USA, March 2007.

[24] Texas Instruments: *Power management for the CC2430DB*. Application note F8W-2006-0019, Texas Instruments, San Diego, CA USA, August 2007.

[25] Texas Instruments: *Z-Stack Application Programming Interface*. API F8W-2006-0021 v1.2, Texas Instruments, San Diego, CA USA, July 2007.

[26] Texas Instruments: *Z-Stack compile options*. Developer notes: F8W-2005-0038, Texas Instruments, San Diego, CA USA, April 2007.

[27] Texas Instruments: *Z-Stack OS Abstraction Layer API*. API F8W-2003-0002 v1.5, Texas Instruments, San Diego, CA USA, December 2007.

[28] Texas Instruments: *Z-Stack sample application for CC2430DB*. Developer notes: F8W-2007-0017, Texas Instruments, San Diego, CA USA, March 2007.

[29] Texas Instruments: *Z-Stack user's guide for CC2430ZDB*. Developer notes: F8W-2005-0036, Texas Instruments, San Diego, CA USA, December 2007.

[30] Texas Instruments: *Over Air Download for CC2430*. Developer notes: F8W-2008-0003, Texas Instruments, San Diego, CA USA, November 2008.

[31] Texas Instruments: *A true system-on-chip solution for 2.4 GHz IEEE 802.15.4 / ZigBee*. Data sheet SWRS036F (rev. 2.1), Texas Instruments, San Diego, CA USA, November 2008. http://www.ti.com/lit/pdf/swrs036f.

[32] Wensink, M.: *Real-time Software Ontwikkeling*. Technical report, Utrecht University of Applied Sciences, Utrecht, NL, November 2007. Unpublished reader.

[33] Whelan, Matthew J., Michael V. Gangone, Kerop D. Janoyan, and Ratneshwar Jha: *Real-time wireless vibration monitoring for operational modal analysis of an integral abutment highway bridge*. Engineering Structures, 31(10):2224 – 2235, 2009, ISSN 0141-0296. http://www.sciencedirect.com/science/article/B6V2Y-4W7HP0W-2/2/8c869c9ef3a59ab9b664e878edf7dd88.

[34] Wijk van, Dennis: *ZigBee solution choice motivation*. Technical report, Utrecht University of Applied Sciences, March 2007. Unpublished document.

[35] Ye, Qing, Yuecheng Zhang, and Liang Cheng: *A study on the optimal time synchronization accuracy in wireless sensor networks*. Computer Networks, 48(4):549 – 566, 2005, ISSN 1389-1286. http://www.sciencedirect.com/science/article/B6VRG-4F8TTJM-3/2/8e414dee55b7620323134d529525b3b5.

Appendices

A

Models

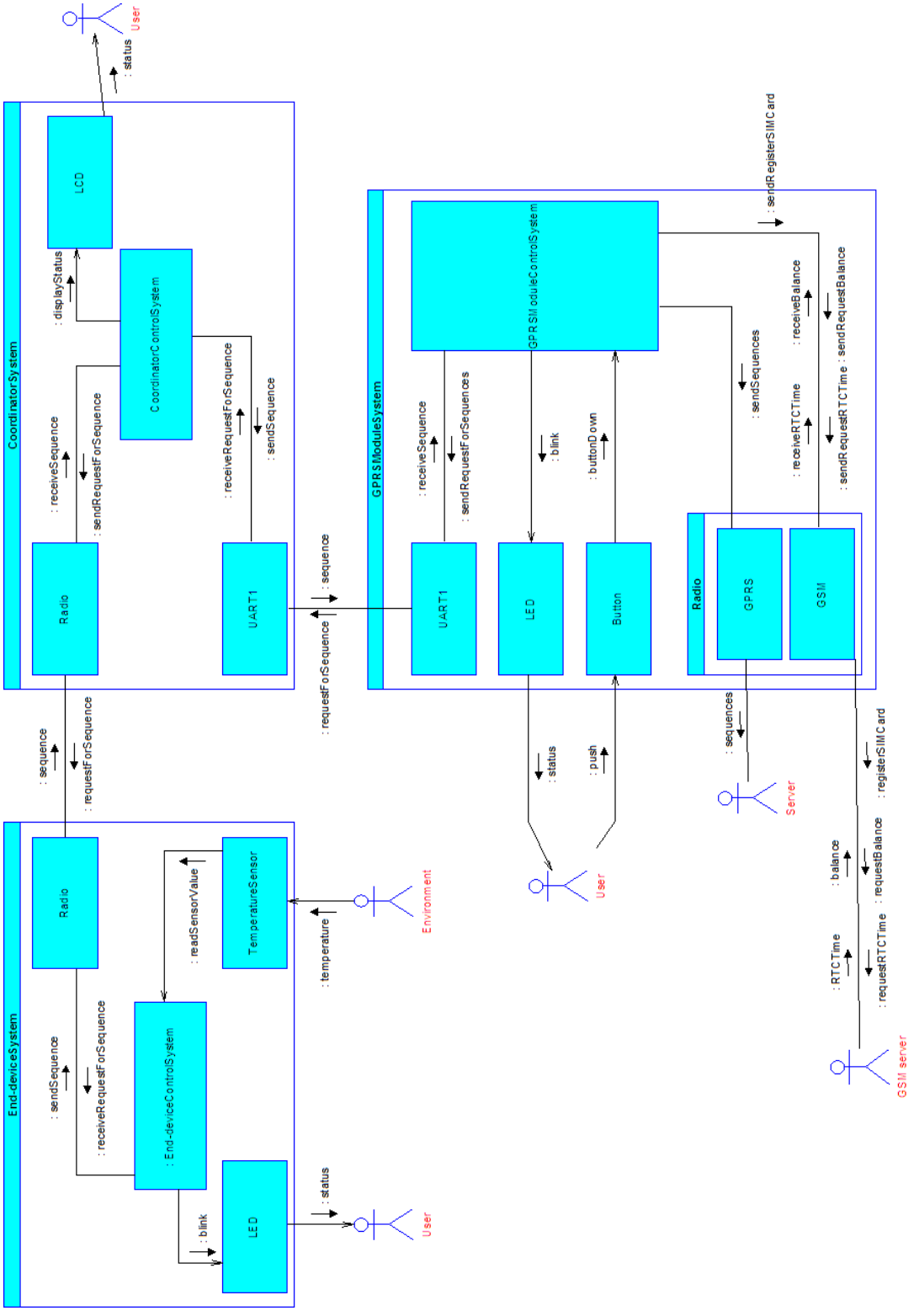


Figure A.1: Context diagram (large).

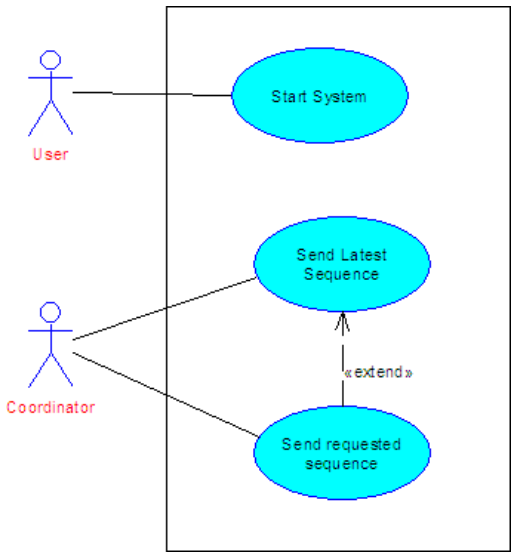


Figure A.2: Use case diagram of the ZigBee end-device.

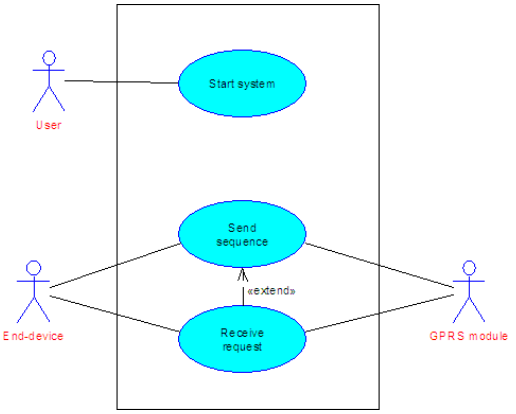


Figure A.3: Use case diagram of the ZigBee coordinator.

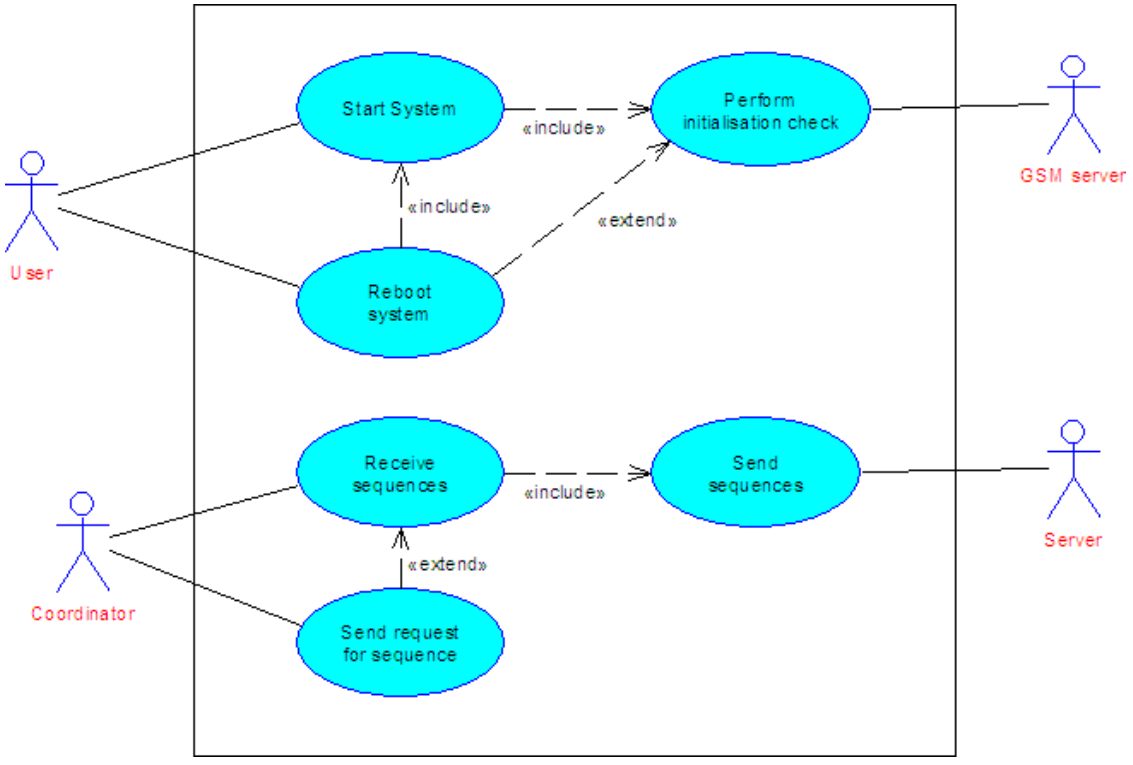


Figure A.4: Use case diagram of the GPRS module.

End-device case: Start system

Purpose:	Activates the system
Pre-condition:	System is not active
Post-condition:	System is active and initialised
Description:	Initialise ZigBee
	Set timer for next measurement
	Write status to LED.

End-device case: Send Latest Sequence

Purpose:	Take a sample from a sensor
Pre-condition:	System is active and initialised
Post-condition:	A sequence is sent to the actor coordinator
Description:	Register to ZigBee network
	Send latest sequence
Exceptions:	IF the actor 'Coordinator' has a request for a sequence stored
	THEN execute <extend> "Receive request for sequence"
	IF ZigBee network is unreachable
	THEN sequence is not sent.

End-device case: Send requested sequence

Purpose:	Sends all requested sequences to actor 'Coordinator'
Pre-condition:	Actor 'coordinator' has sent a request for a sequence
Post-condition:	All available sequences are sent
Description:	Receive a request for a sequence
	Consult local database if sequence is still available
	Send all the sequences that are still available

Coordinator case: Start system

Purpose:	Activates the system
Pre-condition:	System is not active
Post-condition:	System is active and initialised
Description:	Initialise ZigBee
	Write status to display

Coordinator case: Send sequence

Purpose:	Actor 'End-device' sends a sequence, which is sent to actor 'GPRS module'
Pre-condition:	Actor 'End-device' is available and has a new sequence ready to send
Post-condition:	A sequence has been received from the actor 'End-device' and sent to actor 'GPRS module'
Description:	Receive a sequence from the actor 'End-device'
	Send sequence to GPRS module

Coordinator case: Receive request

Purpose:	Actor 'GPRS module' send a request for (a) sequence(s), which needs to be sent to actor 'End-device'
Pre-condition:	Actor 'GPRS module' is missing a sequence
Post-condition:	Request for sequence(s) has been sent to actor 'End-device'
Description:	Receive request for (a) sequence(s) from actor 'GPRS module'
	Store the sequence until actor 'End-device' is available

Use case diagram of GPRS module case: Start system

Purpose:	Start the system to pass sequences through
Pre-condition:	The system is de-activated
Post-condition:	The system is active and the application has been started
Description:	Turn on the system
	Activate the application
	Execute <include> "Perform initialisation check"

Use case diagram of GPRS module case: Perform initialisation check

Purpose:	Performs a system initialisation before it is operational
Pre-condition:	The system is active and the application has been started
Post-condition:	The system is initialised and operational
Description:	Log in the SIM card
	Check balance
	Perform time synchronization
Exceptions:	Execute <extend> "Reboot system" if SIM login failed or when no credit

Use case diagram of GPRS module case: Reboot system

Purpose:	Turns the system off
Pre-condition:	The system is active
Post-condition:	The system is rebooted
Description:	IF the button is pressed OR the "Perform initialisation check" failed
	THEN reboot the system

Use case diagram of GPRS module case: Receive sequences

Purpose:	Receives a sequence and store it
Pre-condition:	The system is initialised and operational
Post-condition:	A request for a sequence has been received and stored
Description:	Receive sequence
	Store the sequence in the 'new sequences' list
	Execute <include> Send sequences
	IF sequence number is in the 'requested sequences' list
	THEN remove sequence number from list
Exceptions:	Execute <extend> "Send request for sequence" when the sequence number do not succeed each other

Use case diagram of GPRS module case: Send sequences

Purpose:	Send stored sequence to actor 'Server'
Pre-condition:	A request for a sequence has been received and stored
Post-condition:	New sequences have been sent to actor 'Server'
Description:	Send the sequences
	Remove sequences from the 'new sequences' list

Use case diagram of GPRS module case: Send request for sequence

Purpose:	Send a request for a sequence to actor 'Coordinator'
Pre-condition:	A sequence has been received and stored
Post-condition:	A request has been sent to actor 'Coordinator'
Description:	Determine missing sequence numbers that are not in the 'requested sequences' list
	Send request for these numbers to actor 'Coordinator'
	Store numbers in the 'requested sequences' list

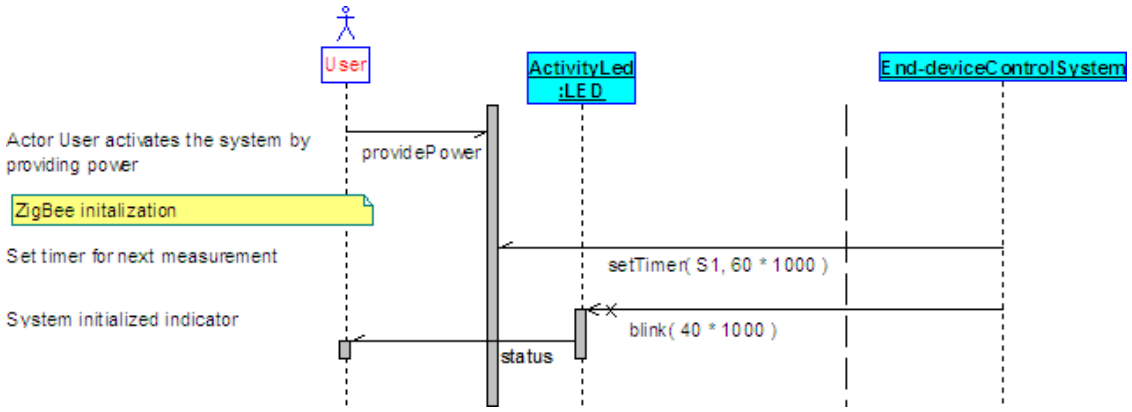


Figure A.5: Sequence diagram of the end-device; Start system.

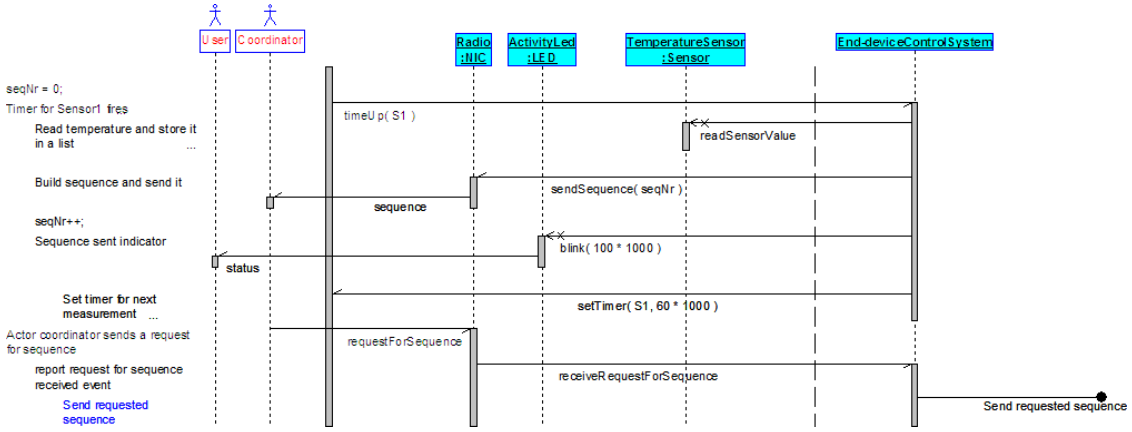


Figure A.6: Sequence diagram of the end-device; Send latest sequence.

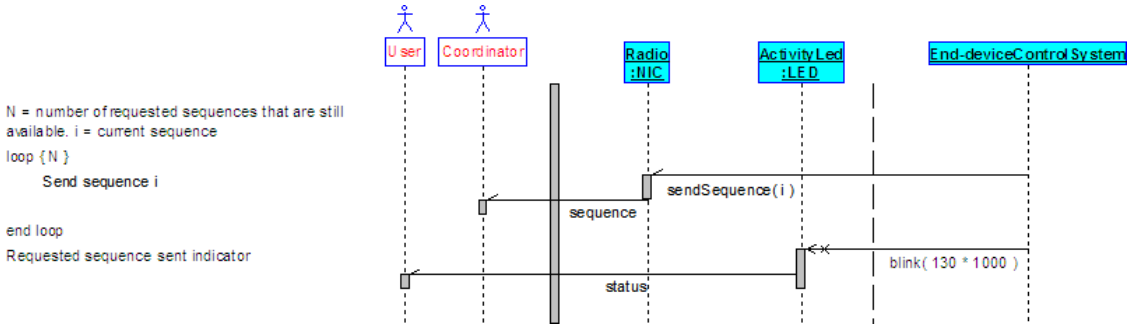


Figure A.7: Sequence diagram of the end-device; Send requested sequence.

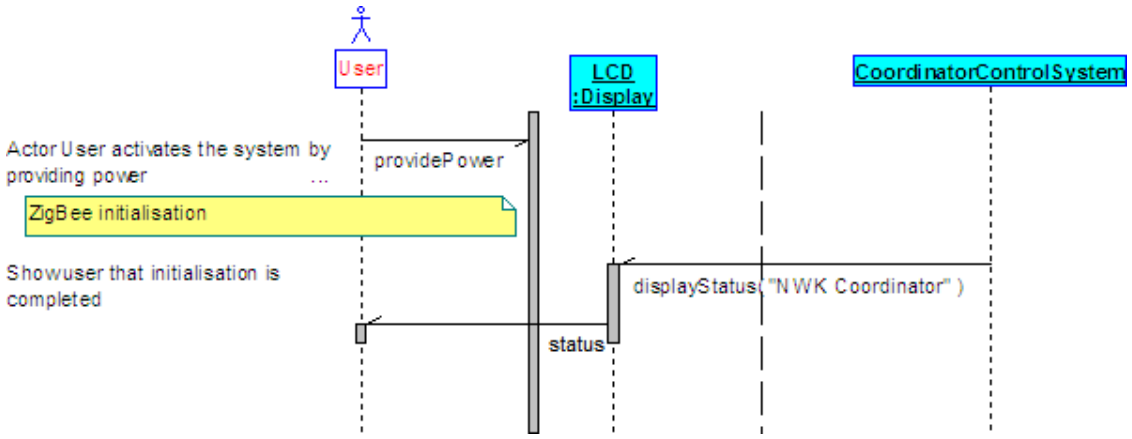


Figure A.8: Sequence diagram of the coordinator; Start system.

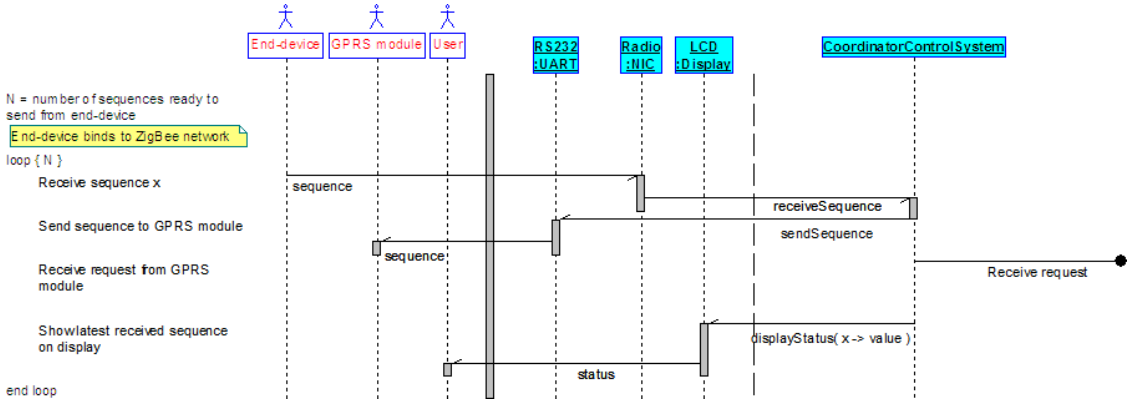


Figure A.9: Sequence diagram of the coordinator; Send sequence.

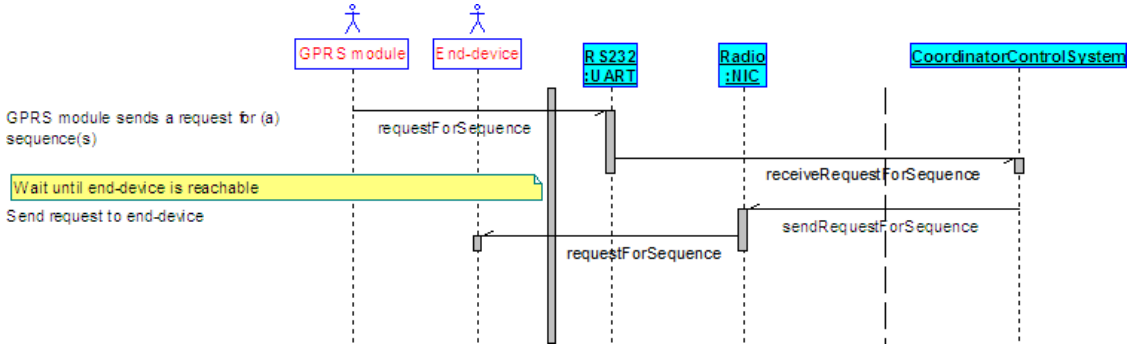


Figure A.10: Sequence diagram of the coordinator; Receive request.

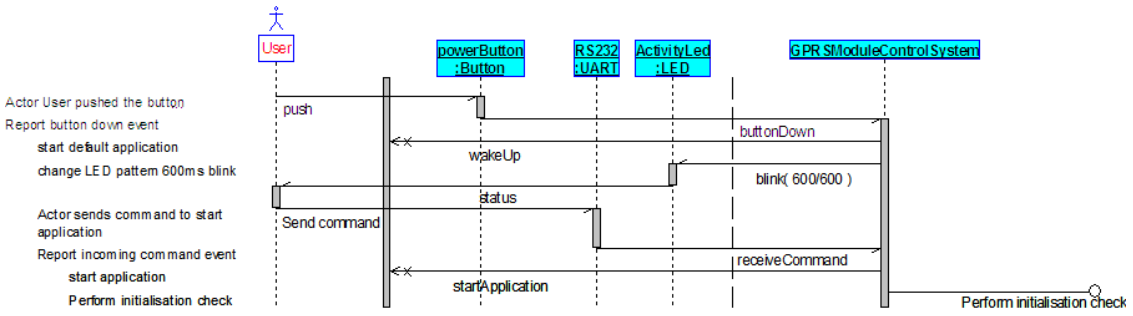


Figure A.11: Sequence diagram of the GPRS module; Start system.

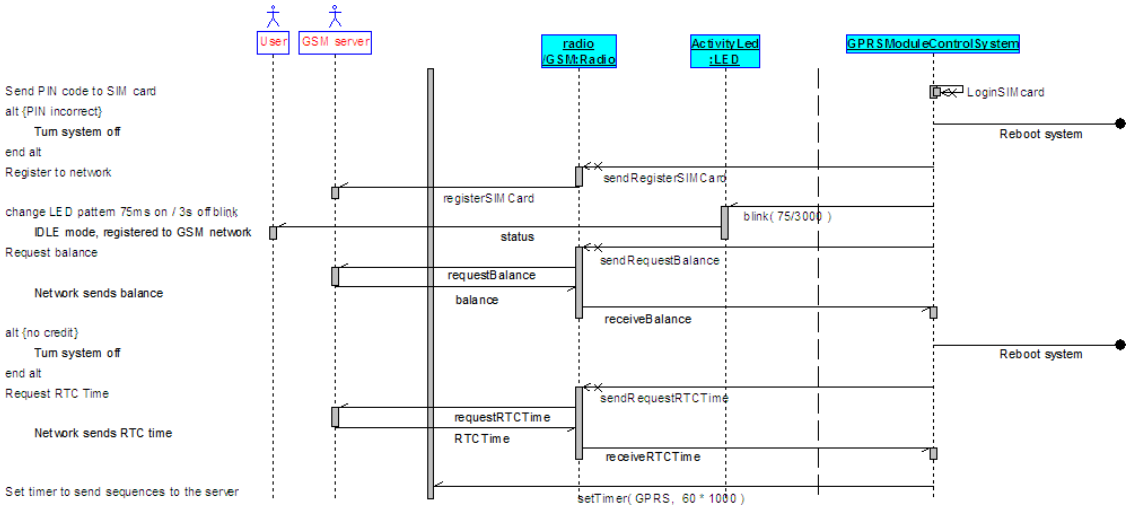


Figure A.12: Sequence diagram of the GPRS module; Perform initialisation check.

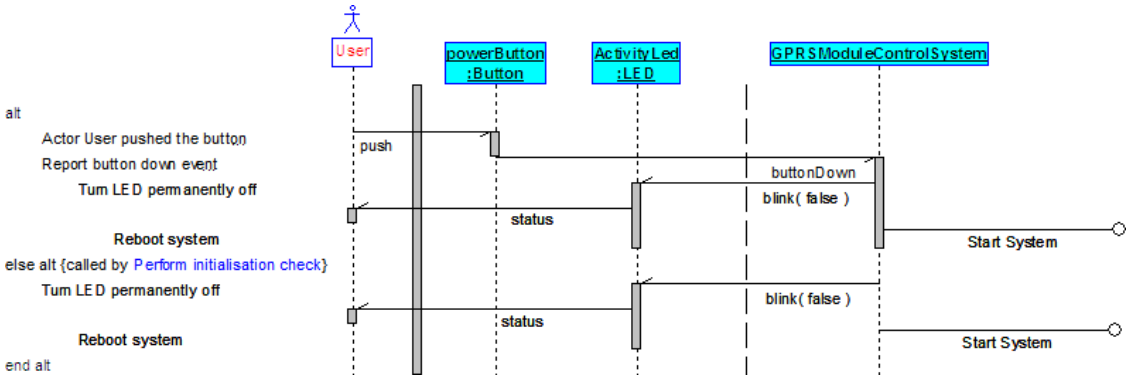


Figure A.13: Sequence diagram of the GPRS module; Reboot system.

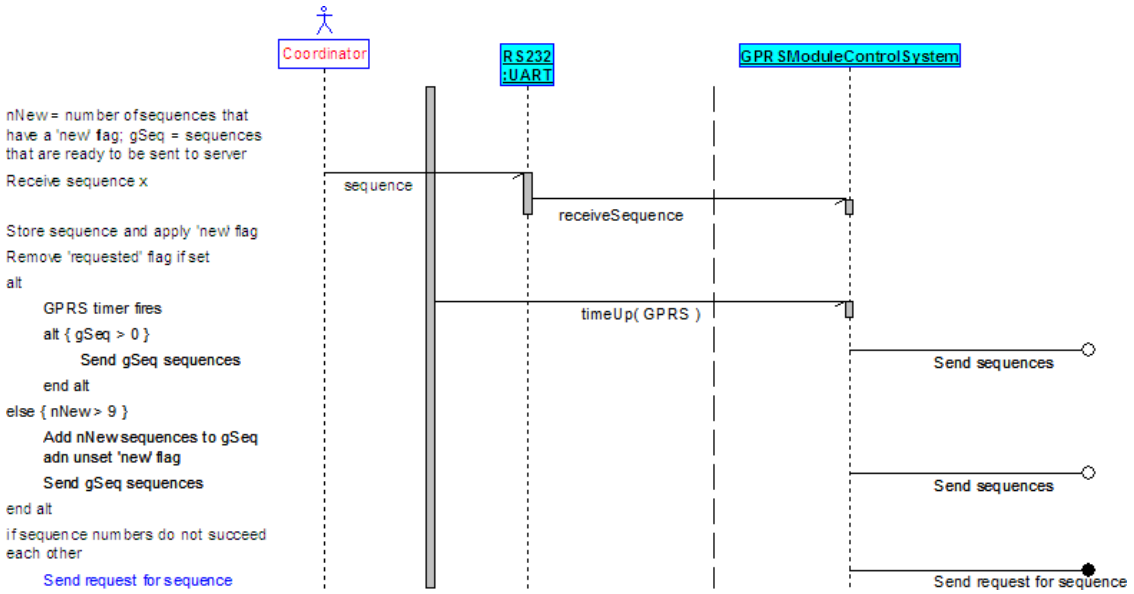


Figure A.14: Sequence diagram of the GPRS module; Receive sequences.

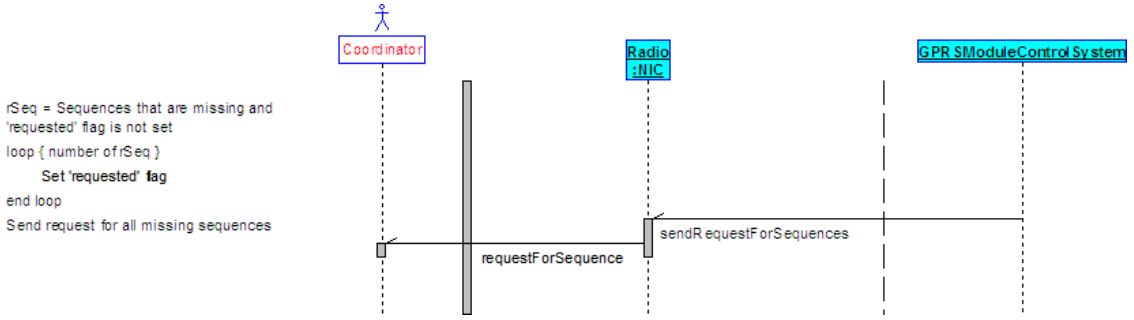


Figure A.15: Sequence diagram of the GPRS module; Send request for sequence.

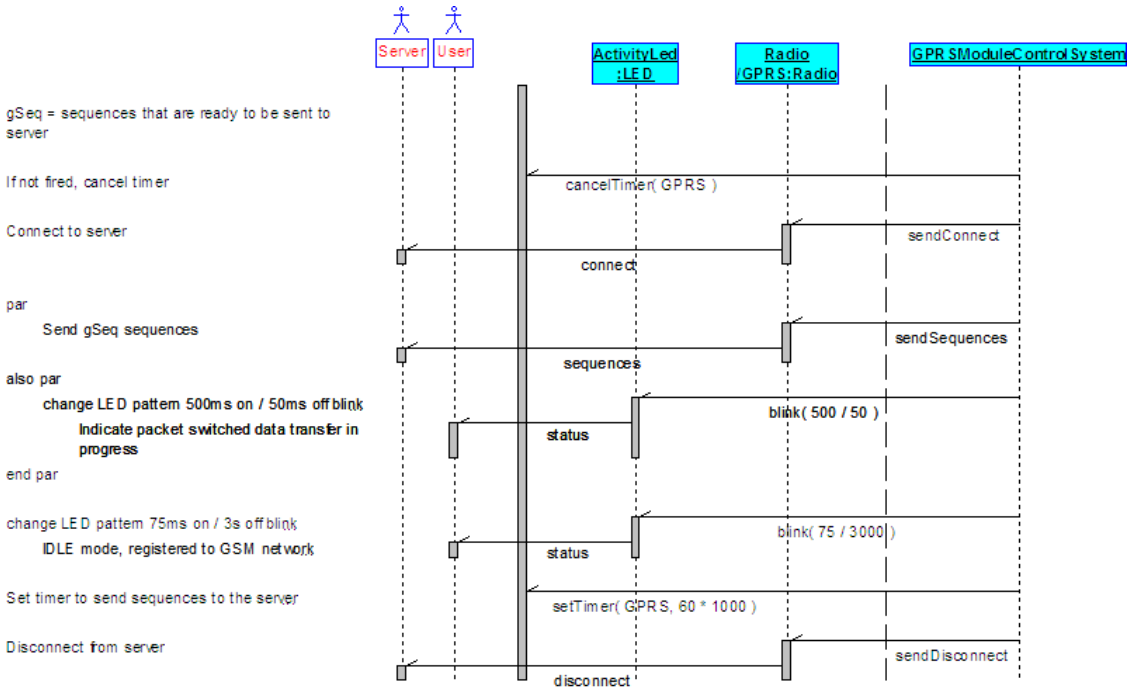


Figure A.16: Sequence diagram of the GPRS module; Send sequence.

B

Useful AT commands

```
// Debug settings:
%ProgramFiles%\Siemens\SMTK\TC65_R3\WTK\bin\WM_Debug_config.ini.

// Enter debug mode with extended text format
AT+CMEE=2\r

// Pin log in:
AT+CPIN?\r (if responds is :+CPIN: READY, no code necessary)
AT+CPIN=0000\r

// (RSSI:received signal strength indication, BIR:bit error rate)
// RSSI is "0":very bad - "31":very good, BIR is "0":OK "99":Not measured
AT+CSQ\r

// Disable / enable network registration URC:
AT+CGREG=0\r
AT+CGREG=1\r



// Set connection ID 1 for GPRS
AT^SICS=1,conType,GPRS0\r
AT^SICS=1,user,vodafone\r
AT^SICS=1,passwd,vodafone\r
AT^SICS=1,apn,office.vodafone.nl\r
AT^SICS=1,inactTO,300\r

// Create a socket ID, based on connection 1
AT^SISS=1,srvType,Socket\r
AT^SISS=1,alphabet,1\r
AT^SISS=1,tcpMR,15\r
AT^SISS=1,tcpOT,3000\r
AT^SISS=1,address,socktcp://<ip_address>:<port>\r
AT^SISS=1,conId,1\r

// Attach GPRS
AT+CGATT=0\r
AT+CGATT=1\r

// Open socket connection, based on socket ID 1
AT^SISO=1\r

// Ping IP address, based on socket ID 1
AT^SISW=1,5\r

// Close socket and disconnect GPRS
AT^SISC=1\r
AT+CGATT=0\r


// Request credit
ATD*101#;\r
--> +CUSD: 2,"Uw Prepaid TeGoed is Euro 0.52. Geldig tot 21/10/2010.",0
ATH\r // Hang up

// Time synchronization
AT+CGATT=0\r
AT+CGATT=1\r
AT^SIND=nitz,2\r
AT+CCLK="04/07/23,00:39:20"\r

// Start Java application
AT^SJRA=a:/GatewayApp.jar\r

// Reboot system
AT+CFUN=1,1\r
```

C

GPRS module application

C.1 GatewayApp.java

```
1  package src;
2
3  import java.util.Enumeration;
4  import java.util.Vector;
5
6  import javax.microedition.midlet.MIDlet;
7  import javax.microedition.midlet.MIDletStateChangeException;
8
9  import com.siemens.icm.io.ATCommandFailedException;
10
11 /**
12  * @author M.A. Janssen
13  *
14  */
15 public class GatewayApp extends MIDlet {
16   // Log
17   Log log;
18
19   // RS232
20   UART uART;
21
22   // GPRS
23   GPRS gprs;
24
25   // Main thread
26   public static Object lock = new Object();
27
28   // TimeSync
29   TimeSynchronization timeSync;
30
31   // Database of samples
32   public Database database;
33
34   // System functions
35   SystemModule sysModule;
36
37   // Defined in JAD file
38   private String apn = "";
39   private String host = "";
40   private String hostport = "";
41   private long timeoutSend;
42   private String login = "";
43   private String pass = "";
44   private String SIMPIN = "";
45   private String BlancePhonenumber = "";
46   public String COMport = "";
47   public String COMBaudrate = "";
48   public String COMDataBits = "";
49   public String COMParity = "";
50   public String COMStopBits = "";
51   public static String creditFormat = "";
52   public static String creditFormat2 = "";
53   private int minSamplesToSend = 0;
54   private int uartSleep = 0;
55
56   private boolean initialized = false;
57   public static boolean mainFlag = true;
58
59   private boolean DEBUG_MODE = false;
60
```

```
61  // RS-232
62  public static final char SYNC_1 = 0xAA;
63  public static final char SYNC_2 = 0x55;
64  public static final short SERIAL_DATA_LEN = 1;
65  public static final short IEEE_LEN = 8;
66  public static final short QUANTITY_LEN = 2;
67  public static final short SEQ_LEN = 2;
68  public static final short SEQ__RES_LEN = 1;
69  public static final short SAMPLE_LEN = 2;
70  public static final short SENSOR_TYPE_LEN = 2;
71  public static final short DATA_SAMPLE_LEN = 2;
72  public static final short TIME_STAMP_LEN = 4;
73  public static final int MIN_REC_LEN = 34;
74
75  public GatewayApp() {
76    log = new Log("Gateway.txt");
77    if (DEBUG_MODE) {
78      log.write("GatewayApp()");
79    }
80
81    // get specific values from the jad file
82    apn = getAppProperty("APN");
83    login = getAppProperty("login");
84    pass = getAppProperty("pass");
85    host = getAppProperty("Host");
86
87    hostport = getAppProperty("HostPort");
88    timeoutSend = Long.parseLong(getAppProperty("maxTimeoutGPRS"));
89    SIMPIN = getAppProperty("SIMPIN");
90    BlancePhonenumber = getAppProperty("BalanceNumber");
91    minSamplesToSend = Integer.parseInt(getAppProperty("minSamplesToSend"));
92
93    // COM Settings
94    COMport = getAppProperty("COMPort");
95    COMBaudrate = getAppProperty("COMBaudrate");
96    COMDataBits = getAppProperty("COMDataBits");
97    COMParity = getAppProperty("COMParity");
98    COMStopBits = getAppProperty("COMStopBits");
99    uartSleep = Integer.parseInt(getAppProperty("UARTSleep"));
100   GatewayApp.creditFormat = getAppProperty("creditFormat");
101   GatewayApp.creditFormat2 = getAppProperty("creditFormat2");
102
103   database = new Database();
104   timeSync = new TimeSynchronization();
105   sysModule = new SystemModule();
106
107 }
108
109 /*
110  * (non-Javadoc)
111  *
112  * @see javax.microedition.midlet.MIDlet#destroyApp(boolean)
113  */
114 protected void destroyApp(boolean arg0) throws MIDletStateChangeException {
115   if (DEBUG_MODE) {
116     log.write("destroyApp(" + arg0 + ")");
117   }
118
119   if (uART != null) {
120     // Reset flag
121     UART.flag = false;
122     try {
123       uART.join();
124     } catch (InterruptedException e) {
125     }
126
127   }
128   if (gprs != null) {
129     GPRS.flag = false;
130     // Notify waiting thread
131     gprs.notify();
132     try {
```

```
133         gprs.join();
134       } catch (InterruptedException e) {
135       }
136     }
137     notifyDestroyed();
138   }
139
140   /*
141    * (non-Javadoc)
142    *
143    * @see javax.microedition.midlet.MIDlet#pauseApp()
144    */
145   protected void pauseApp() {
146     if (DEBUG_MODE) {
147       log.write("pauseApp()");
148     }
149   }
150
151   /*
152    * (non-Javadoc)
153    *
154    * @see javax.microedition.midlet.MIDlet#startApp()
155    */
156   protected void startApp() throws MIDletStateChangeException {
157     if (DEBUG_MODE) {
158       log.write("startApp");
159     }
160
161     try {
162       // Initialize module
163       init();
164
165       // Open RS232
166       uART = new UART(COMport, COMBaudrate, COMDataBits, COMParity, COMStopBits, uartSleep);
167       uART.start();
168
169       // Open GPRS
170       gprs = new GPRS(database, host, hostport, apn, login, pass, timeoutSend);
171       gprs.start();
172
173       String RS232MsgRec = "";
174       while (mainFlag) {
175         // Wait until notified
176         synchronized (GatewayApp.lock) {
177           try {
178             // Look every 5 minutes if not notified
179             GatewayApp.lock.wait(5 * 60 * 1000);
180           } catch (InterruptedException e) {
181           }
182         }
183
184         // Check if data received
185         if (uART.getMsgReceivedSize() > 0) {
186           RS232MsgRec += uART.getMsgReceived(false);
187
188           // Save in DB
189           try {
190             RS232MsgRec = readFromCoordinator(RS232MsgRec);
191           } catch (Exception e) {
192             log.write("readFromCoordinator -> " + e.toString());
193           }
194
195           // Check for new sequences
196           if (database.size() > 0) {
197             for (Enumeration nodeList = database.elements(); nodeList.hasMoreElements();) {
198               Node tempNode = (Node) nodeList.nextElement();
199
200               Vector seq = tempNode.getMissingSequencesNotRequested();
201               if (seq != null && seq.size() > 0) {
202                 if (DEBUG_MODE) {
203                   log.write("missing seq.size() > 0");
204                 }
```

```
205                 for (Enumeration e = seq.elements(); e.hasMoreElements();) {
206                   String seqStr = (String) e.nextElement();
207                   // Set sequence request in node (so will only be requested once)
208                   tempNode.setRequestForSequence(Integer.parseInt(seqStr));
209                 }
210                 uART.setSequencesToReceive(tempNode.getIEEE(), seq);
211               }
212             }
213           }
214         }
215
216         // Check for new Nodes and/or sequences
217         if (minSamplesToSend < database.getNewSequencesSize()) {
218
219           // Get new sequences and reset new-flag
220           Vector nodeList = database.getNewSequences();
221
222           if (DEBUG_MODE) {
223             // Test if correct
224             log.write(nodeList.toString());
225           }
226
227           // Write to outputFile
228           String txtToFile = "";
229           for (Enumeration e = nodeList.elements(); e.hasMoreElements();) {
230             Node tempNode = (Node) e.nextElement();
231             txtToFile += gprs.formatNodeData(tempNode);
232           }
233           txtToFile = gprs.formatToXML(txtToFile);
234           gprs.appendToOutputBuffer(txtToFile);
235
236           // Empty local database
237           database.optimize();
238
239           // Notify gprs that a new package is ready to be sent
240           gprs.notifyGprs();
241         }
242       }
243       if (DEBUG_MODE) {
244         log.write("mainFlag: " + mainFlag);
245       }
246       destroyApp(true);
247
248     } catch (Exception e) {
249       log.write("Big error\n" + e.toString() + "\n" + e.getMessage());
250       log.write("mainFlag: " + mainFlag);
251       destroyApp(true);
252     }
253   }
254
255   /**
256    * System initialisation .
257    * <ol>
258    * <li>Activate and register SIM card</li>
259    * <li>Check balance on SIM card</li>
260    * <li>Request RTC time and reset local time</li>
261    * </ol>
262    *
263    * @return
264    */
265   private boolean init() {
266     if (! initialized ) {
267       // Init SIM
268       if (DEBUG_MODE) {
269         log.write("INIT SIM");
270       }
271       try {
272         sysModule.initSim(SIMPIN, BlancePhonenumber);
273       } catch (IllegalStateException e1) {
274         log.write("sysMod IllegalStateException " + e1.toString());
275       } catch (ATCommandFailedException e1) {
276         log.write("sysMod ATCommandFailedException " + e1.toString());
```

```
277       } catch (Exception e1) {
278         log.write("sysMod Exception " + e1.toString());
279       }
280
281       // TimeSync
282       timeSync.setDateTime();
283
284        initialized  = true;
285     }
286     return initialized;
287   }
288
289   /**
290    * Read list of nodes, sequences and datasamples and puts them in the
291    * database.
292    *
293    * @return Remainingp part of the String that is not used (if any).
294    */
295   private String readFromCoordinator(String inMsg) {
296     String debug = "";
297     if (DEBUG_MODE) {
298       log.write("readFromCoordinator");
299     }
300     int msgIndex = 0;
301     int max = inMsg.length();
302
303     int sync1Index = inMsg.indexOf((int) SYNC_1);
304     while (sync1Index != -1 && sync1Index < (max - MIN_REC_LEN) && inMsg.charAt(sync1Index + 1) == (
          int) SYNC_2) {
305       msgIndex = sync1Index + 2; // Update pointer
306
307       String debugInt = "";
308       for (int i = 0; i < inMsg.length(); i++) {
309         debugInt += (int) (inMsg.charAt(i)) + " ";
310       }
311       log.write(debugInt);
312
313       // Debug
314       long startComm = SystemModule.getDate().getTime();
315
316       // Data length
317       int dataLength = 0;
318       for (short i = 0; i < SERIAL_DATA_LEN; i++) {
319         dataLength = (dataLength << 8) + inMsg.charAt(msgIndex++);
320       }
321       int sampleLen = 2 + SERIAL_DATA_LEN + IEEE_LEN + QUANTITY_LEN + IEEE_LEN +
              TIME_STAMP_LEN + SEQ_LEN + SAMPLE_LEN + TIME_STAMP_LEN +
              SENSOR_TYPE_LEN + DATA_SAMPLE_LEN + 1;//
              38
322
323       // Data length should be long enough
324       if (dataLength < sampleLen) {
325         log.write("( " + dataLength + " < " + sampleLen + " )");
326         // ERROR, TOO LITTLE DATA, ignore it
327         return inMsg.substring(sync1Index + dataLength);
328       }
329
330       // Get Coordinator IEEE
331       String CoordinatorIEEE = "";
332       for (short i = 0; i < IEEE_LEN; i++) {
333         int temp = inMsg.charAt(msgIndex++);
334         CoordinatorIEEE += temp;
335         if (temp == 0) {
336           // 0 will be 00
337           CoordinatorIEEE += temp;
338         }
339       }
340
341       try {
342         // Set database coordinator
343         database.setCoordinator(CoordinatorIEEE);
344         if (DEBUG_MODE) {
```

```
345           debug += CoordinatorIEEE;
346         }
347       } catch (Exception e) {
348         log.write("database.setCoordinator EX " + e.toString());
349       }
350
351       // Posible point of failure, rest is data from end device. If not
352       // send, next two bytes are sync bytes.
353       if (inMsg.charAt(msgIndex) == (int) SYNC_1 && inMsg.charAt(msgIndex + 1) == (int) SYNC_2) {
354         log.write("numberOfDevices = SYNC_1 & SYNC_2");
355         // ERROR, data incorrect
356         return inMsg.substring(sync1Index + SERIAL_DATA_LEN + IEEE_LEN);
357       }
358
359       // Get number of devices for samples
360       int numberOfDevices = 0;
361       for (short i = 0; i < QUANTITY_LEN; i++) {
362         numberOfDevices = (numberOfDevices << 8) + inMsg.charAt(msgIndex++);
363       }
364       if (DEBUG_MODE) {
365         debug += numberOfDevices;
366       }
367
368       // For number of devices, receive sequences
369       for (int deviceX = 0; deviceX < numberOfDevices; deviceX++) {
370         // Get IEEE
371         String IEEE = "";
372         for (short i = 0; i < IEEE_LEN; i++) {
373           int temp = inMsg.charAt(msgIndex++);
374           IEEE += temp;
375           if (temp == 0)
376             IEEE += temp;
377         }
378         if (DEBUG_MODE) {
379           debug += IEEE;
380         }
381
382         // Get node if available, otherwise make new
383         int nodeIndex = database.getNodeIndex(IEEE);
384         if (nodeIndex < 0) {
385           if (DEBUG_MODE) {
386             debug += "New Node";
387           }
388           database.appendNode(new Node(IEEE));
389         }
390
391         Node node = database.getNode(IEEE);
392         node.setCoordinatorIEEE(CoordinatorIEEE);
393
394         // Get time stamp of data sent
395         long tos = 0;
396         for (short i = 0; i < TIME_STAMP_LEN; i++) {
397           tos = (tos << 8) + inMsg.charAt(msgIndex++);
398         }
399         if (DEBUG_MODE) {
400           debug += " tos:" + tos;
401         }
402
403         // Get Sequence
404         int sequence = 0;
405         for (short i = 0; i < SEQ_LEN; i++) {
406           sequence = (sequence << 8) + inMsg.charAt(msgIndex++);
407         }
408         if (DEBUG_MODE) {
409           debug += " sequence:" + sequence;
410         }
411
412         // Get sample quantity
413         int sampleQuantity = 0;
414         for (short i = 0; i < SAMPLE_LEN; i++) {
415           sampleQuantity = (sampleQuantity << 8) + inMsg.charAt(msgIndex++);
416         }
```

```
417         if (DEBUG_MODE) {
418           debug += " sampleQuantity:" + sampleQuantity;
419         }
420 
421         // New Sequence
422         node.addSequence(tos, "" + sequence, sampleQuantity);
423 
424         // For every data sample
425         for (int sample = 0; sample < sampleQuantity; sample++) {
426           // Test if inMsg.length() is sufficient
427           int minLength = TIME_STAMP_LEN + SENSOR_TYPE_LEN + DATA_SAMPLE_LEN + 1;
428 
429           if ((msgIndex + minLength) > max) {
430             int lengthNeeded = msgIndex + minLength - max;
431             inMsg += waitForMsg(lengthNeeded);
432             max = inMsg.length();
433           }
434 
435           // Time stamp
436           long time_stamp = 0;
437           for (short i = 0; i < TIME_STAMP_LEN; i++) {
438             time_stamp = (time_stamp << 8) + inMsg.charAt(msgIndex++);
439           }
440 
441           // Sensor type
442           int sensor_type = 0;
443           for (short i = 0; i < SENSOR_TYPE_LEN; i++) {
444             sensor_type = (sensor_type << 8) + inMsg.charAt(msgIndex++);
445           }
446 
447           // Data length
448           int data_len = 0;
449           log.write("data_len: " + (int) (inMsg.charAt(msgIndex + 1)) + ", "
450               + (int) (inMsg.charAt(msgIndex + 2)));
451           log.write(debug);
452           for (short i = 0; i < DATA_SAMPLE_LEN; i++) {
453             data_len = (data_len << 8) + inMsg.charAt(msgIndex++);
454           }
455           if (DEBUG_MODE) {
456             debug += " data_len:" + data_len;
457           }
458 
459           // Data. Format byte1,byte2,byte2...
460           if ((msgIndex + data_len) > max) {
461             int lengthNeeded = msgIndex + data_len - max;
462             log.write("lengthNeeded" + lengthNeeded);
463             inMsg += waitForMsg(lengthNeeded);
464             max = inMsg.length();
465           }
466           String data = "";
467           for (short i = 0; i < data_len; i++) {
468             if (i != 0) {
469               data += ",";
470             }
471             data += (int) inMsg.charAt(msgIndex++);
472           }
473           if (DEBUG_MODE) {
474             debug += " data:" + data;
475           }
476           node.addSample("" + sequence, "" + sensor_type, data, time_stamp);
477         }
478 
479         // Display updated database
480         if (DEBUG_MODE) {
481           // Time
482           long commTime = SystemModule.getDate().getTime() - startComm;
483           log.write(debug);
484           debug = "";
485           log.write("Communication took " + commTime + "ms.\nNode data " + node.toString());
486         }
487       }
488 
```

```
489       // Send received msg
490       uART.setMsgToSend("" + (char) 0x04, true);
491
492       if (msgIndex > max) {
493         return "";
494       } else {
495         inMsg = inMsg.substring(msgIndex);
496         max = inMsg.length();
497       }
498
499       // search next sync
500       sync1Index = inMsg.indexOf((int) SYNC_1);
501     }
502     return inMsg;
503   }
504
505   /**
506    * Append to String from UART when too short
507    *
508    * @param minMsgLen
509    * @return
510    */
511   private String waitForMsg(int minMsgLen) {
512     if (DEBUG_MODE) {
513       log.write("waitForMsg(" + minMsgLen + " )");
514     }
515     int recLen = 0;
516     String msg = "";
517     while (recLen < minMsgLen) {
518
519       synchronized (GatewayApp.lock) {
520         try {
521           // Look every 1s if not notified
522           GatewayApp.lock.wait(1000);
523         } catch (InterruptedException e) {
524         }
525       }
526       if (uART.getMsgReceivedSize() > 0) {
527         msg += uART.getMsgReceived(false);
528         recLen = msg.length();
529       }
530     }
531     return msg;
532   }
533 }
```

C.2 Log.java

```
1  package src;
2
3  import java.io.DataOutputStream;
4  import java.io.IOException;
5  import java.io.InputStream;
6  import java.io.OutputStream;
7
8  import javax.microedition.io.Connector;
9  import javax.microedition.io.InputConnection;
10
11 import com.siemens.icm.io.file.FileConnection;
12
13 /**
14  * Creates a log file. Automatically removes the first half of the file when the file becomes too big.
15  *
16  * @author M.A. Janssen
17  *
18  */
19 public class Log {
20   private String filename;
21   private FileConnection fconn;
22   private InputConnection inputConn;
23   private long freeSpace = 0;
```

```
24
25  public Log(String filename) {
26    this.filename = "A:/" + filename;
27
28    // Create if not exsist
29    try {
30      fconn = (FileConnection) Connector.open("file:///" + this.filename + "", Connector.READ_WRITE);
31      if (!fconn.exists()) {
32        fconn.create();
33      }
34      fconn.close();
35    } catch (IOException ioe) {
36    } catch (Exception e) {
37    }
38  }
39
40  /**
41   * Append str to the file .
42   *
43   * @param str
44   * @return Number of character written.
45   */
46  public int write(String str) {
47    return append(str);
48  }
49
50  /**
51   * Appends str to log file , including a time stamp and new line characters
52   *
53   * @param str
54   * @return Number of character written.
55   */
56  private int writeFile(String str) {
57    int nrWritten = 0;
58
59    try {
60      String fileNewContent = "";
61
62      // Time
63      String newDateTime = SystemModule.getDateString();
64
65      fileNewContent = newDateTime + "\n" + str + "\n\n"; // One new line
66
67      // Assume file exsists
68      fconn = (FileConnection) Connector.open("file:///" + filename, Connector.READ_WRITE);
69      // Opening with fconn.fileSize() prevents overwrite
70      OutputStream os = fconn.openOutputStream(fconn.fileSize());
71      DataOutputStream dos = new DataOutputStream(os);
72
73      // Write to file
74      byte[] contentInByte = fileNewContent.getBytes();
75      dos.write(contentInByte);
76
77      nrWritten = fileNewContent.length();
78
79      // Close connections
80      dos.close();
81      os.close();
82      fconn.close();
83    } catch (IOException ioe) {
84      // if write of log fails, what to do?
85    }
86    return nrWritten;
87  }
88
89  /**
90   * Returns the content of the file
91   *
92   * @return The file&#039;s content
93   */
94  public String read() {
95    String str = "";
```

```
96
97      try {
98        inputConn = (InputConnection) Connector.open("file:///" + filename, Connector.READ);
99
100       // Read from file
101       InputStream in = inputConn.openInputStream();
102       int ch;
103       while ((ch = in.read()) != -1) {
104         str += (char) ch;
105       }
106
107       // Close connections
108       in.close();
109       inputConn.close();
110     } catch (IOException ioe) {
111     }
112
113     // Return read chars
114     return str;
115   }
116
117   /**
118    * Append content to log, removing half FIFO if necessary
119    *
120    * @param msg
121    * @return Number of character written.
122    */
123   private int append(String msg) {
124
125     if (!enoughSpace(msg.length())) {
126       // Append half of the file to msg and empty the file
127       String fileContent = read();
128
129       // Find first enter character in the centre
130       int centre = fileContent.length() / 2;
131       int index = centre + fileContent.substring(centre).indexOf("\n") + 1;
132
133       // New data
134       msg = fileContent.substring(index) + msg;
135
136       // Empty Log file
137       try {
138         fconn = (FileConnection) Connector.open("file:///" + filename, Connector.READ_WRITE);
139         fconn.truncate(0l);// Removes content
140         fconn.close();
141       } catch (IOException e) {
142       }
143     }
144     // Append msg
145     return writeFile(msg);
146   }
147
148   /**
149    * Checks if the system has enough space for the string to write. We reserve
150    * 10 kB for GPRS output buffer
151    *
152    * @param strLen
153    * @return Returns of tehre is enough space.
154    */
155   private boolean enoughSpace(int strLen) {
156     long fileSize;
157     // Create if not exist
158     try {
159       fconn = (FileConnection) Connector.open("file:///" + this.filename + "", Connector.READ_WRITE);
160        fileSize  = fconn.fileSize();
161       freeSpace = fconn.availableSize();
162       fconn.close();
163     } catch (Exception e) {
164       return false;
165     }
166
167     // Calculate new file size, + extra
```

```
168     Integer strLenI = new Integer(strLen);
169      fileSize  += strLenI.longValue() + (10 * 1024);
170
171     // Return if enough space
172     return (fileSize < freeSpace) ? true : false;
173   }
174
175   /**
176    * Closes the active connection (if opened)
177    */
178   public void close() {
179     // Close Fileconnector
180     if (fconn != null && fconn.isOpen()) {
181       try {
182         fconn.close();
183       } catch (IOException ioe) {
184       }
185     }
186   }
187 }
```

C.3 SystemModule.java

```
1  package src;
2
3  import java.util.Calendar;
4  import java.util.Date;
5  import javax.microedition.midlet.MIDletStateChangeException;
6
7  import com.siemens.icm.io.ATCommand;
8  import com.siemens.icm.io.ATCommandFailedException;
9
10 /**
11  * All kinds of methods that may come in handy.
12  *
13  * @author M.A. Janssen
14  *
15  */
16 public class SystemModule {
17   private static Log log;
18   public static double balance = (-1.0 * Double.MAX_VALUE);
19   private boolean DEBUG_MODE = true;
20   private static long timestampLastSync = 0;
21   private static long timestampSync = 0;
22
23   public SystemModule() {
24     log = new Log("SystemModule.txt");
25   }
26
27   /**
28    * Initialize access to the SIM card, waits for a connection to the network,
29    * and checks for balance.
30    *
31    * @throws IllegalStateException
32    * @throws ATCommandFailedException
33    * @throws MIDletStateChangeException
34    */
35   public void initSim(String SIMPIN, String BlancePhonenumber) throws IllegalStateException,
36       ATCommandFailedException, MIDletStateChangeException, Exception {
37     ATCommand atc = new ATCommand(false);
38     String str;
39
40     // Query SIM and Chip Card Holder Status
41     str = atc.send("at^scks?\r");
42     if (str.indexOf(",1") == -1) {
43       // No SIM card found
44       throw new Exception("No SIM card found. Please check SIM.");
45     }
46
47     // Send PIN
48     str = atc.send("AT+CPIN?\r");
```

```
49    if (str.indexOf("SIM PIN") != -1) {
50      // SIM requeres a PIN code, at least 1 try should be left
51
52      String retries = atc.send("AT^SPIC\r");
53      if (retries == null) {
54        throw new Exception("AT^SPIC returned null ");
55      }
56
57      /* response &#094;SPIC: SIM PIN OK */
58      int index = retries.indexOf(":");
59      retries = retries.substring(index + 2, index + 3);
60      if (retries != null && retries != "R" && Integer.parseInt(retries) > 1) {
61        // Enter PIN
62        str = atc.send("AT+CPIN=" + SIMPIN + "\r");
63        if (str.indexOf("OK") == -1) {
64          throw new Exception("Invalid SIM PIN: " + SIMPIN + ". Warning : " + retries + " attempts left.");
65        }
66      } else {
67        // ERROR, print where the error came from
68        str = atc.send("AT^SPIC?\r");
69        throw new Exception("SIM ERROR: " + str);
70      }
71    } else if (str.indexOf("READY") == -1) {
72      // SIM is requesting for a different authentication
73      throw new Exception("SIM PIN not ready: " + str);
74    }
75
76    // Wait for network
77    while (atc.send("AT+COPS?\r").indexOf("\"") == -1) {
78      try {
79        Thread.sleep(1000);
80      } catch (InterruptedException e) {
81        // Do nothing
82      }
83    }
84
85    // Network ready, now wait to check credit
86    SystemModule.balance = checkBalance(BlancePhonenumber, true);
87    if (SystemModule.balance < 0.02) {
88      throw new Exception("Not enough credit: " + SystemModule.balance);
89    }
90
91    if (DEBUG_MODE) {
92      log.write("Network ready");
93    }
94    atc.release();
95  }
96
97  /**
98   * Sends a request to the GSM network to send the current balance
99   *
100  * @param atc
101  * @return
102  */
103 public double checkBalance(String BlancePhonenumber, boolean wait) {
104   if (DEBUG_MODE) {
105     log.write("checkBalance(" + BlancePhonenumber + ", " + wait + ")");
106   }
107
108   try {
109     ATCommand atc = new ATCommand(false);
110
111     // Listen to +CUSD for credit
112     ATListener listen = new ATListener("credit");
113
114     // Call number to check balance
115     atc.addListener(listen);
116     atc.send("ATD" + BlancePhonenumber + ";\r");
117
118     // Will return in ATListener
119     if (wait) {
120       // Make blocking, wait 60sec at maximum
```

```
121         boolean notified = false;
122         try {
123           synchronized (GatewayApp.lock) {
124             GatewayApp.lock.wait(60 * 1000);
125           }
126           notified = true;
127         } catch (InterruptedException e) {
128         }
129         /*
130          * It is recommended to finalize or escape a pending USSD user
131          * interaction before further actions are done to prevent
132          * blocking situations
133          */
134         // Cancel session
135         if (!notified) {
136           log.write("checkBalance CANCEL");
137           atc.send("AT+CUSD=2\r");
138         }
139       }
140
141       if (DEBUG_MODE) {
142         log.write("Balance is Euro " + SystemModule.balance);
143       }
144
145       // Hang up
146       atc.send("ATH\r");
147     } catch (IllegalStateException e) {
148       if (DEBUG_MODE) {
149         log.write("SystemModule,checkBalance exception" + e.toString());
150       }
151     } catch (IllegalArgumentException e) {
152       if (DEBUG_MODE) {
153         log.write("SystemModule,checkBalance exception" + e.toString());
154       }
155     } catch (ATCommandFailedException e) {
156       if (DEBUG_MODE) {
157         log.write("SystemModule,checkBalance exception" + e.toString());
158       }
159     }
160     return SystemModule.balance;
161   }
162
163   /**
164    * Gets the balance in received and notifies waiting Thread that balance has
165    * been received.
166    *
167    * @param received
168    */
169   public void setBalance(String received) {
170     if (received.length() > 0) {
171
172       String creditFormatPrefix = GatewayApp.creditFormat;
173       String creditFormatPostfix = GatewayApp.creditFormat2;
174
175       int formatIndex = received.indexOf(creditFormatPrefix);
176       int formatIndex2 = received.indexOf(creditFormatPostfix);
177
178       if (formatIndex > -1 && formatIndex2 > 0) {
179         SystemModule.balance = Double.parseDouble(received.substring(formatIndex + creditFormatPrefix.length()
                , formatIndex2));
180       } else {
181         // ERROR
182         log.write("checkBalance setBalance WRONG = " + received + " ->" + creditFormatPrefix + "." +
                creditFormatPostfix);
183
184         // Assume enough credit
185         SystemModule.balance = 10.0;
186       }
187
188     }
189
190     // Notify that balance is changed
```

```
191     synchronized (GatewayApp.lock) {
192       GatewayApp.lock.notify();
193     }
194   }
195
196   /**
197    * Set the current RTC time.
198    *
199    * @param timestamp
200    */
201   public static void setDate(long timestamp) {
202     timestampSync = timestamp;
203     timestampLastSync = new Date().getTime();
204   }
205
206   /**
207    * Returnes the RTC time if setDate() has been set.
208    *
209    * @return
210    */
211   public static Date getDate() {
212     return new Date(timestampSync + (new Date().getTime() - timestampLastSync));
213   }
214
215   /**
216    * Retunes a String representation of the current time
217    *
218    * @return
219    */
220   public static String getDateString() {
221     Calendar cal = Calendar.getInstance();
222     cal.setTime(SystemModule.getDate());
223     String newDateTime = SystemModule.fill(cal.get(Calendar.YEAR), 2) + "/"
224         + SystemModule.fill((cal.get(Calendar.MONTH) + 1), 2) + "/"
225         + SystemModule.fill(cal.get(Calendar.DATE), 2) + " "
226         + SystemModule.fill(cal.get(Calendar.HOUR_OF_DAY), 2) + ":"
227         + SystemModule.fill(cal.get(Calendar.MINUTE), 2) + ":" + SystemModule.fill(cal.get(Calendar.
                SECOND), 2);
228     return newDateTime;
229   }
230
231   /**
232    * Fills up msg with &amp;#039;0&amp;#039; prefix...
233    *
234    * @param msg
235    * @param fill
236    * @return (fill * &amp;#039;0&amp;#039;) + msg
237    */
238   public static String fill(int msg, int fill) {
239     String s = "" + msg;
240     for (int i = 0; s.length() < fill; i++) {
241       s = "0" + s;
242     }
243     return s;
244   }
245
246   /**
247    * Fills up msg with &amp;#039;0&amp;#039; prefix...
248    *
249    * @param msg
250    * @param fill
251    * @return (fill * &amp;#039;0&amp;#039;) + msg
252    */
253   public static String fill(String msg, int fill) {
254     String s = "" + msg;
255     for (int i = 0; s.length() < fill; i++) {
256       s = "0" + s;
257     }
258     return s;
259   }
260
261   /**
```

```
262     * @see src#strToAscii(String msg, int radix, int  fill )
263     * @param msg
264     * @param radix
265     * @return
266     */
267    public static String strToAscii(String msg, int radix) {
268      return strToAscii(msg, radix, 0);
269    }
270
271    /**
272     * @param msg
273     * @param radix
274     * @param fill
275     * @return
276     */
277    public static String strToAscii(String msg, int radix, int fill) {
278      String ascii = "";
279      for (int i = 0; i < msg.length(); i = i + radix) {
280        String str = "";
281        for (int j = 0; j < radix && (i + j) < msg.length(); j++) {
282          str += msg.charAt(i + j);
283        }
284        ascii += (char) Byte.parseByte(str);
285      }
286      for (int i = 0; ascii.length() < fill; i++) {
287        ascii = '\u0000' + ascii;
288      }
289      return ascii;
290    }
291
292    /**
293     * Removes char c from string s
294     *
295     * @param s
296     * @param c
297     * @return
298     */
299    public static String removeChar(String s, char c) {
300      String r = "";
301      for (int i = 0; i < s.length(); i++) {
302        if (s.charAt(i) != c)
303          r += s.charAt(i);
304      }
305      return r;
306    }
307
308    private final static char[] HEX = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D
         ', 'E', 'F'};
309
310    /**
311     * @see src#longtoHex(long value, boolean trim)
312     *
313     * @param v
314     * @return
315     */
316    public static String longtoHex(long v) {
317      return SystemModule.longtoHex(v, true);
318    }
319
320    /**
321     * Converts a long to hexadecimal, removing the prefix zeroes  if  trim  is
322     * true.
323     *
324     * @param value
325     * @param trim
326     * @return
327     */
328    public static String longtoHex(long value, boolean trim) {
329      char[] hexs;
330      hexs = new char[16];
331      for (int i = 0; i < 16; i++) {
332        int c = (int) (value & 0xf);
```

```
333       hexs[16 - i - 1] = HEX[c];
334       value = value >> 4;
335     }
336
337     // remove prefix zeroes?
338     if (!trim) {
339       return new String(hexs);
340     }
341     String h = "";
342     for (int i = 0; i < 16; i++) {
343       char ch = hexs[i];
344       if (!(h == "" && ch == '0')) {
345         h += ch;
346       }
347     }
348     return h;
349   }
350 }
```

C.4 ATListener.java

```
1  package src;
2
3  import com.siemens.icm.io.ATCommandListener;
4
5  /**
6   * Helper class for AT messages
7   * @author M.A. Janssen
8   *
9   */
10 public class ATListener implements ATCommandListener {
11   String listenFor = "";
12
13   public ATListener(String waitingResponse) {
14     this.listenFor = waitingResponse;
15   }
16
17   /*
18    * (non-Javadoc)
19    *
20    * @see com.siemens.icm.io.ATCommandListener#ATEvent(java.lang.String)
21    */
22   public void ATEvent(String response) {
23     if (listenFor.equals( "credit" )) {
24       SystemModule sysM = new SystemModule();
25       sysM.setBalance(response);
26     }
27   }
28
29   /*
30    * (non-Javadoc)
31    *
32    * @see com.siemens.icm.io.ATCommandListener#CONNChanged(boolean)
33    */
34   public void CONNChanged(boolean arg0) {
35     // TODO Auto-generated method stub
36   }
37
38   /*
39    * (non-Javadoc)
40    *
41    * @see com.siemens.icm.io.ATCommandListener#DCDChanged(boolean)
42    */
43   public void DCDChanged(boolean arg0) {
44     // TODO Auto-generated method stub
45   }
46
47   /*
48    * (non-Javadoc)
49    *
50    * @see com.siemens.icm.io.ATCommandListener#DSRChanged(boolean)
```

```
51     */
52    public void DSRChanged(boolean arg0) {
53      // TODO Auto-generated method stub
54    }
55
56    /*
57     * (non-Javadoc)
58     *
59     * @see com.siemens.icm.io.ATCommandListener#RINGChanged(boolean)
60     */
61    public void RINGChanged(boolean arg0) {
62      // TODO Auto-generated method stub
63    }
64  }
```

C.5 TimeSynchronization.java

```
1  package src;
2
3  import java.util.Calendar;
4
5  import com.siemens.icm.io.ATCommand;
6  import com.siemens.icm.io.ATCommandFailedException;
7
8  /**
9   * Time synchronisation with GSM network
10  *
11  * @author M.A. Janssen
12  *
13  */
14 public class TimeSynchronization {
15   private Log log;
16   private int UT = 1;
17   private boolean DEBUG_MODE = false;
18
19   public TimeSynchronization() {
20     if (DEBUG_MODE) {
21       log = new Log("TimeSynchronization.txt");
22     }
23   }
24
25   /**
26    * Requests new date/time from network and resets the local time. Attention:
27    * a re-attach to the network is necessary..
28    *
29    * @return
30    */
31   public String setDateTime() {
32     String msg = "";
33     try {
34       ATCommand atc = new ATCommand(false);
35
36       // Reattach to network
37       atc.send("AT+CGATT=0\r");
38       atc.send("AT+CGATT=1\r");
39
40       // Get Network Identity and Time Zone indication (NITZ)
41       String nitz = atc.send("AT^SIND=nitz,2\r");
42
43       // DateTime
44       int iBegin = nitz.indexOf("\"") + 1;
45       int iEnd = nitz.indexOf("\"", iBegin);
46       String dateTime = nitz.substring(iBegin, iEnd);
47
48       if (dateTime.length() < 17) {
49         if (DEBUG_MODE) {
50           log.write("TimeSynchronization error: dateTime.length() < 17");
51         }
52         return "";
53       }
54
```

```
55        // WinterTime
56        String winterTime = nitz.substring(55, 56); // 55, 56, iEnd + 1,//
57        // iEnd + 2
58
59        // Local date
60        int year = Integer.parseInt(dateTime.substring(0, 2));
61        int month = Integer.parseInt(dateTime.substring(3, 5));
62        int day = Integer.parseInt(dateTime.substring(6, 8));
63
64        // Local time
65        int hour = Integer.parseInt(dateTime.substring(9, 11));
66        int min = Integer.parseInt(dateTime.substring(12, 14));
67        int sec = Integer.parseInt(dateTime.substring(15, 17));
68
69        // New date
70        Calendar cal = Calendar.getInstance();
71        cal.set(Calendar.DATE, day);
72        cal.set(Calendar.MONTH, month - 1);
73        cal.set(Calendar.YEAR, 2000 + year);
74        cal.set(Calendar.HOUR_OF_DAY, hour);
75        cal.set(Calendar.MINUTE, min);
76        cal.set(Calendar.SECOND, sec);
77
78        hour += Integer.parseInt(winterTime) + UT;
79        // If new day
80        if (hour > 23) {
81
82          hour = hour % 24;
83          day += 1;
84
85          // Assume no more than one day
86          cal.set(Calendar.DATE, day);
87        }
88        cal.set(Calendar.HOUR_OF_DAY, hour);
89        // New date
90        String newATDateTime = SystemModule.fill(cal.get(Calendar.YEAR), 2).substring(2, 4) + "/"
91            + SystemModule.fill(cal.get(Calendar.MONTH) + 1, 2) + "/"
92            + SystemModule.fill(cal.get(Calendar.DATE), 2) + ","
93            + SystemModule.fill(cal.get(Calendar.HOUR_OF_DAY), 2) + ":"
94            + SystemModule.fill(cal.get(Calendar.MINUTE), 2) + ":"
95            + SystemModule.fill(cal.get(Calendar.SECOND), 2);
96
97        // Set new date/time
98        String succes = atc.send("AT+CCLK=\"" + newATDateTime + "\"\r");
99        if (succes.indexOf("OK") > -1) {
100         // Also set Java time
101         // Date tempDate = cal.getTime();
102
103         // New time
104         SystemModule.setDate((cal.getTime()).getTime());
105         String newDateTime = SystemModule.getDateString();
106         if (DEBUG_MODE) {
107           log.write("TimeSynchronization: New DateTime = " + newDateTime);
108         }
109         msg = newDateTime;
110       } else {
111         if (DEBUG_MODE) {
112           log.write("TimeSynchronization ERROR FOR New DateTime = " + newATDateTime);
113         }
114       }
115
116       // Release memory
117       atc.release();
118     } catch (ATCommandFailedException ate) {
119       if (DEBUG_MODE) {
120         log.write("Timesync AT exception " + ate.toString());
121       }
122       msg = ate.getMessage();
123     } catch (Exception e) {
124       if (DEBUG_MODE) {
125         log.write("Timesync exception " + e.getMessage());
126       }
```

```
127     }
128     return msg;
129   }
130 }
```

C.6 UART.java

```
1  package src;
2
3  import java.io.DataInputStream;
4  import java.io.EOFException;
5  import java.io.IOException;
6  import java.io.InputStream;
7  import java.io.OutputStream;
8  import java.util.Enumeration;
9  import java.util.Vector;
10
11 import javax.microedition.io.CommConnection;
12 import javax.microedition.io.Connector;
13
14 /**
15  * @author Marten A. Janssen
16  *
17  */
18 public class UART extends Thread {
19
20   // RS232
21   CommConnection COMCon;
22   InputStream inStream;
23   OutputStream outStream;
24   DataInputStream dis;
25
26   // Thread
27   public static boolean flag = true;
28   private StringBuffer msgToSend = new StringBuffer();
29   private StringBuffer msgReceived = new StringBuffer();
30
31   // Defined in JAD file
32   String COMport = "";
33   String baudrate = "";
34   String dataBits = "";
35   String parity = "";
36   String stopBits = "";
37   int uartSleep = 0;
38
39   boolean DEBUG_MODE = true;
40
41   // Log for debug
42   Log log;
43
44   UART(String COMport, String baudrate, String dataBits, String parity, String stopBits, int uartSleep) {
45     this.COMport = COMport;
46     this.baudrate = baudrate;
47     this.dataBits = dataBits;
48     this.parity = parity;
49     this.stopBits = stopBits;
50     this.uartSleep = uartSleep;
51
52     log = new Log("WSNServer.txt");
53   }
54
55   public void run() {
56     // Open COM port
57     if (!RS232Open()) {
58       // Unable to open RS232 port... Closing thread
59       UART.flag = false;
60       log.write("Unable to open RS-232");
61     }
62
63     // Execute thread until flag is false
64     while (UART.flag) {
```

```
65
66      try {
67        // Send message
68        if (getMsgToSendSize() > 0) {
69          String msgToSend = getMsgToSend(false);
70          write(msgToSend);
71        }
72
73        // Read if available
74        String msgRead = read();
75        if (msgRead.length() > 0) {
76          setMsgReceived(msgRead, true);
77        }
78
79        // Notify if enough received
80        if (getMsgReceivedSize() > GatewayApp.MIN_REC_LEN) {
81          // notify GatewayApp
82          notifyReceived();
83        }
84
85      } catch (EOFException e) {
86        // The stream reaches the end before reading all the bytes.
87        // Thread extermination not required
88        log.write("EOF: " + e.getMessage());
89      } catch (IOException e) {
90        log.write("IOException: " + e.getMessage());
91        // reopen RS-232 port
92        RS232Close();
93        // Open COM port
94        if (!RS232Open()) {
95          // Unable to open RS232 port, retry in 1s
96          try {
97            Thread.sleep(1000);
98          } catch (InterruptedException ie) {
99          }
100       }
101     } catch (Exception e) {
102       log.write("Run Exception: " + e.getMessage());
103     }
104   }
105   // Close COM port
106   RS232Close();
107 }
108
109 /**
110  * If append, the received message will be appended to the StringBuffer
111  * msgReceived. Otherwise, msgReceived will cleared before the message
112  * received is appended.
113  *
114  * @param msg
115  * @param append
116  */
117 public void setMsgReceived(String msg, boolean append) {
118   if (!append) {
119     msgReceived.setLength(0);
120   }
121   msgReceived.append(msg);
122 }
123
124 /**
125  * Returns the message received from RS-232. If peek, the message will not
126  * be cleared afterwards.
127  *
128  * @param peek
129  * @return
130  */
131 public String getMsgReceived(boolean peek) {
132   String msg = msgReceived.toString();
133   if (!peek) {
134     msgReceived.setLength(0);
135   }
136   return msg;
```

```
137   }
138
139   /**
140    * Returns the size of the message received from RS-232.
141    *
142    * @return
143    */
144   public int getMsgReceivedSize() {
145     return msgReceived.length();
146   }
147
148   /**
149    * Notifies the main Thread that a message has been received.
150    */
151   private void notifyReceived() {
152     synchronized (GatewayApp.lock) {
153       GatewayApp.lock.notify();
154       if (DEBUG_MODE) {
155         log.write("GatewayApp notified that a new message is received over rs232");
156       }
157     }
158   }
159
160   /**
161    * If append, the message to sent ocer RS-232 will be appended to the
162    * StringBuffer msgToSend. Otherwise, msgToSend will cleared before the
163    * message to sent is appended.
164    *
165    * @param msg
166    * @param append
167    */
168   public void setMsgToSend(String msg, boolean append) {
169     if (!append) {
170       msgToSend.setLength(0);
171     }
172     msgToSend.append(msg);
173   }
174
175   /**
176    * Returns the message to sent over RS-232. If peek, the message will not be
177    * cleared afterwards.
178    *
179    * @param peek
180    * @return
181    */
182   public String getMsgToSend(boolean peek) {
183     String msg = msgToSend.toString();
184     if (!peek) {
185       msgToSend.setLength(0);
186     }
187     return msg;
188   }
189
190   /**
191    * Returns the size of the message to sent over RS-232.
192    *
193    * @return
194    */
195   public int getMsgToSendSize() {
196     return msgToSend.length();
197   }
198
199   /**
200    * Creates a message to be sent over RS-232 to request a sequence from a
201    * node
202    *
203    * @param node
204    *            Node to request sequences
205    * @param seqs
206    *            Sequences to request
207    */
208   public void setSequencesToReceive(String IEEE, Vector seqs) {
```

```
209     if (DEBUG_MODE) {
210       log.write("setSequencesToReceive( " + IEEE + ", " + seqs + ")");
211     }
212
213     String msg = "";
214     msg += ((char) GatewayApp.SYNC_1);
215     msg += ((char) GatewayApp.SYNC_2);
216     msg += SystemModule.strToAscii("" + GatewayApp.IEEE_LEN, 2, 2);
217     msg += SystemModule.strToAscii(IEEE, 2, (int) GatewayApp.IEEE_LEN);
218     msg += SystemModule.strToAscii("" + seqs.size(), 2, (int) GatewayApp.SEQ__RES_LEN);
219
220     // Missing sequences
221     for (Enumeration e = seqs.elements(); e.hasMoreElements();) {
222       // Convert sequence number into byte
223       String seqStr = (String) e.nextElement();
224       int seqInt = Integer.parseInt(seqStr);
225       Integer parse = new Integer(seqInt);
226       String hexStr = "" + parse.byteValue();
227
228       // Fill numbers to SEQ LEN (2 byte)
229       msg += SystemModule.strToAscii(hexStr, 2, (int) GatewayApp.SEQ_LEN);
230     }
231     setMsgToSend(msg, true);
232   }
233
234   /**
235    * Reads a string from the stream. This method will read the amount of bytes
236    * available (if any). If no bytes are available, it puts the thread to
237    * sleep for 10ms.
238    *
239    * @return
240    * @throws IOException
241    */
242   private String read() throws IOException {
243     String msg = "";
244     byte[] bInput;
245     int available = dis.available();
246     if (available > 0) {
247       bInput = new byte[available];
248       inStream.read(bInput, 0, available);
249       msg = new String(bInput);
250     } else {
251       try {
252         Thread.sleep(uartSleep);
253       } catch (InterruptedException ie) {
254       }
255     }
256     return msg;
257   }
258
259   /**
260    * Writes a string to the stream
261    *
262    * @param msg
263    *            The string to write
264    * @throws IOException
265    */
266   private void write(String msg) throws IOException {
267     char[] output = msg.toCharArray();
268     for (int i = 0; i < msg.length(); i++) {
269       outStream.write(output[i]);
270     }
271     outStream.flush();
272   }
273
274   /**
275    * Opens an RS232 connection with the predefined settings
276    *
277    * @return Success
278    */
279   private boolean RS232Open() {
280     if (DEBUG_MODE) {
```

```
281       log.write("RS232Open");
282     }
283
284     try {
285       // Hardware handshaking RTS/CTS
286       String strCOM = "comm:" + COMport + ";baudrate=" + baudrate + ";bitsperchar=" + dataBits + ";parity
              =" + parity + ";stopbits=" + stopBits + ";blocking=on;autocts=on;autorts=on";
287
288       COMCon = (CommConnection) Connector.open(strCOM);
289
290       // Open streams
291       inStream = COMCon.openInputStream();
292       outStream = COMCon.openOutputStream();
293
294       dis = new DataInputStream(inStream);
295
296       return true;
297     } catch (IOException e) {
298       log.write("RS232Open: " + e.getMessage());
299     }
300     return false;
301   }
302
303   /**
304    * Closes the active RS232 connection
305    *
306    * @return Success
307    */
308   private boolean RS232Close() {
309     // Close RS232 connection
310     try {
311       if (dis != null) {
312         dis.close();
313       }
314       if (inStream != null) {
315         inStream.close();
316       }
317       if (outStream != null) {
318         outStream.close();
319       }
320       if (COMCon != null) {
321         COMCon.close();
322       }
323       return true;
324     } catch (IOException e) {
325       log.write("RS232Close: " + e.getMessage());
326     }
327     return false;
328   }
329 }
```

C.7 GPRS.java

```
1  package src;
2
3  import java.io.DataInputStream;
4  import java.io.DataOutputStream;
5  import java.io.EOFException;
6  import java.io.IOException;
7  import java.io.InputStream;
8  import java.io.OutputStream;
9  import java.util.Date;
10 import java.util.Enumeration;
11
12 import javax.microedition.io.Connector;
13 import javax.microedition.io.InputConnection;
14 import javax.microedition.io.OutputConnection;
15 import javax.microedition.io.SocketConnection;
16
17 import com.siemens.icm.io.file.FileConnection;
18
```

```
19 /**
20  * GPRS handler. Reads file and sends the content over GPRS. File is updated by
21  * the number of characters sent.
22  *
23  * @author M.A. Janssen
24  *
25  */
26 public class GPRS extends Thread {
27
28   // GPRS
29   SocketConnection sc;
30   InputStream inStream;
31   OutputStream outStream;
32   String prefix = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>";
33
34   // Thread
35   public static boolean flag = true;
36   public static Object gprsHandle;
37   private String msgReceived = "";
38   private String msgToSend = "";
39   private Database database;
40
41   // Defined in JAD file
42   String host = "";
43   String hostport = "";
44   String connProfile = "";
45   long timeoutSend;
46
47   boolean DEBUG_MODE = true;
48
49   // Log for debug
50   Log log;
51
52   // File output buffer
53   public String outputFileName;
54   FileConnection fconn;
55   OutputConnection outputConn;
56   InputConnection inputConn;
57   DataOutputStream dOS;
58   DataInputStream dIS;
59
60   GPRS(Database database, String host, String hostport, String apn, String login, String pass, long
         timeoutSend) {
61     this.host = host;
62     this.hostport = hostport;
63     this.connProfile = "bearer_type=gprs;access_point=" + apn + ";username=" + login + ";password=" + pass;
64     this.timeoutSend = timeoutSend;
65     this.database = database;
66
67     log = new Log("GPRS.txt");
68     gprsHandle = new Object();
69
70     // OutputFilebuffer
71     this.outputFileName = "A:/GPRS_OUTPUT.txt";
72     // Create if not exsist
73     try {
74       fconn = (FileConnection) Connector.open("file:///" + this.outputFileName + "", Connector.
             READ_WRITE);
75       if (!fconn.exists()) {
76         fconn.create();
77         fconn.setWritable(true);
78       }
79       fconn.close();
80     } catch (IOException ioe) {
81       log.write("GPRS -> " + ioe.getMessage());
82     } catch (Exception e) {
83       log.write("GPRS -> " + e.getMessage());
84     }
85   }
86
87   public void run() {
88     while (GPRS.flag) {
```

```
89      try {
90        // Wait for notification to send msg
91        synchronized (gprsHandle) {
92          if (DEBUG_MODE) {
93            log.write("GPRS WAIT " + timeoutSend);
94          }
95          gprsHandle.wait(timeoutSend);
96        }
97      } catch (InterruptedException ie) {
98      }
99      // Thread can be woken up by timeout or enough sequences in buffer
100     if (DEBUG_MODE) {
101       log.write("GPRS AWAKE");
102     }
103
104     // Read String to write
105     String outputStr = readFromOutputBuffer();
106     if (DEBUG_MODE) {
107       log.write("outputStr = " + outputStr);
108     }
109
110     if (outputStr.length() > 0) {
111
112       // Open GPRS port
113       if (!GPRSOpen()) {
114         // Unable to open GPRS port, try again later
115         log.write("Unable to open GPRS");
116       } else {
117
118         // Add xml version and encoding
119         outputStr = prefix + outputStr;
120
121         // Send over GPRS
122         int charsSent = 0;
123         try {
124           charsSent = write(outputStr);
125
126           // Read if received String
127           String rcv = read(true);
128
129           // Save received message
130           setMsgReceived(rcv, true);
131
132           // TODO: do smth with received message
133
134         } catch (IOException e) {
135           log.write("Write to GPRS IO exception " + e.getMessage());
136         } catch (Exception e) {
137           log.write("Write to GPRS exception " + e.getMessage());
138         }
139
140         // Get number of send characters
141         charsSent = charsSent - prefix.length();
142         if (charsSent > 0) {
143           // TODO: wait for confirmation before updating buffer
144           updateOutputBuffer(charsSent);
145         }
146         // Reset variables
147         // Close GPRS port
148         GPRSClose();
149         if (DEBUG_MODE) {
150           log.write("GPRSClosed");
151         }
152       }
153     }
154   }
155 }
156
157 /**
158  * Set a synchronized received message. Not used
159  *
160  * @param msg
```

```
161     * @param append
162     */
163    private void setMsgReceived(String msg, boolean append) {
164      synchronized (msgReceived) {
165        msgReceived = (append) ? msgReceived + msg : msg;
166      }
167    }
168
169    /**
170     * Reads a synchronized received message. If not peeking, the content will
171     * be removed.
172     *
173     * @param peek
174     * @return
175     */
176    private String getMsgReceived(boolean peek) {
177      synchronized (msgReceived) {
178        String msg = msgReceived;
179        if (!peek) {
180          msgReceived = "";
181        }
182        return msg;
183      }
184    }
185
186    /**
187     * Notify GPRS handler that a message is ready to be sent. Not used.
188     *
189     * @param msg
190     * @param append
191     */
192    private void setMsgToSend(String msg, boolean append) {
193      synchronized (msgToSend) {
194        msgToSend = (append) ? msgToSend + msg : msg;
195      }
196      synchronized (gprsHandle) {
197        gprsHandle.notify();
198        log.write("GPRS notified");
199      }
200    }
201
202    /**
203     * Notify GPRS handler that a message is ready to be sent.
204     */
205    public void notifyGprs() {
206      synchronized (gprsHandle) {
207        gprsHandle.notify();
208      }
209    }
210
211    /**
212     * Reads a message to send. If not peeking, the content will be removed.
213     *
214     * @param peek
215     * @return
216     */
217    public String getMsgToSend(boolean peek) {
218      synchronized (msgToSend) {
219        String msg = msgToSend;
220        if (!peek) {
221          msgToSend = "";
222        }
223        return msg;
224      }
225    }
226
227    /**
228     * Reads a string from the stream. Call to read(false) Not used
229     *
230     * @return
231     * @throws IOException
232     */
```

```
233   private String read() throws IOException {
234     return read(false);
235   }
236
237   /**
238    * Reads a string from the stream. If <i>blocking</i>, this method will
239    * block until it can read something. Otherwise, it will read the amount of
240    * bytes available (if any).
241    *
242    * @param blocking
243    * @return
244    * @throws IOException
245    */
246   private String read(boolean blocking) throws IOException {
247     String msg = "";
248     byte[] bInput;
249     if (blocking) {
250       int ch = 0;
251       StringBuffer str = new StringBuffer();
252       while (ch != -1) {
253         ch = inStream.read();
254         str.append((char) ch);
255       }
256       msg = str.toString();
257     } else {
258       int available = inStream.available();
259       if (available > 0) {
260         bInput = new byte[available];
261         inStream.read(bInput, 0, available);
262         msg += new String(bInput);
263       }
264     }
265     return msg;
266   }
267
268   /**
269    * Writes a string to the stream
270    *
271    * @param msg
272    *            The string to write
273    * @throws IOException
274    */
275   private int write(String msg) throws IOException {
276     if (DEBUG_MODE) {
277       log.write("write(" + msg + ")");
278     }
279     int written = 0;
280     char[] output = msg.toCharArray();
281     for (int i = 0; i < msg.length(); i++) {
282       outStream.write(output[i]);
283       written++;
284     }
285     outStream.flush();
286     return written;
287   }
288
289   /**
290    * Opens a GPRS connection with the predefined settings
291    *
292    * @return Success
293    */
294   private boolean GPRSOpen() {
295     try {
296       String openParm = "socket://" + host + ":" + hostport + ";" + connProfile;
297
298       log.write("GPRSOpen " + openParm);
299       sc = (SocketConnection) Connector.open(openParm);
300       log.write("GPRSOpen socket opened ");
301
302       // Open streams
303       inStream = sc.openInputStream();
304       outStream = sc.openOutputStream();
```

```
305
306       log.write("GPRSOpen streams opened ");
307
308       return true;
309     } catch (IOException e) {
310       log.write("GPRSOpen IO : " + e.toString());
311     } catch (Exception e) {
312       log.write("GPRSOpen E : " + e.toString());
313     }
314     return false;
315   }
316
317   /**
318    * Closes the active GPRS connection
319    *
320    * @return Success
321    */
322   private boolean GPRSClose() {
323     // Close GPRS connection
324     try {
325       if (inStream != null) {
326         inStream.close();
327       }
328       if (outStream != null) {
329         outStream.close();
330       }
331       if (sc != null) {
332         sc.close();
333       }
334       return true;
335     } catch (IOException e) {
336       log.write("GPRSClose: " + e.getMessage());
337     }
338     return false;
339   }
340
341   /**
342    * Returns the sequences and their data samples in XML format for the
343    * server.
344    *
345    * @param node
346    * @return
347    */
348   public String formatNodeData(Node node) {
349     String newMessage = "<end MAC='" + toHex(node.getIEEE()) + "'>\n";
350
351     for (Enumeration e = node.getSequences().elements(); e.hasMoreElements();) {
352       Sequence sequence = (Sequence) e.nextElement();
353
354       // for each sequence
355       newMessage += "\t<seq id='" + toHex("" + sequence.getSequenceNumber()) + "' tos='"
356           + toHex("" + sequence.getTimeStamp()) + "' tor='"
357           + toHex("" + sequence.getrtcTimeStamp().getTime()) + "'>\n";
358       for (Enumeration eS = sequence.getDataSamples().elements(); eS.hasMoreElements();) {
359         DataSample ds = (DataSample) eS.nextElement();
360         newMessage += "\t\t<s ";
361         newMessage += "id='" + toHex(ds.getSensorType()) + "' ";
362         newMessage += "d='" + ds.getDataSample() + "' ";
363         newMessage += "off='" + toHex("" + ds.getTimeStamp()) + "'";
364         newMessage += "/>\n";
365       }
366       newMessage += "\t</seq>\n";
367     }
368     newMessage += "</end>\n";
369
370     return newMessage;
371   }
372
373   /**
374    * Add a final encapsulation and time stamp in XML to be sent
375    *
376    * @param data
```

```
377     * @return
378     */
379    public String formatToXML(String data) {
380      Date date = SystemModule.getDate();
381      long timestamp = date.getTime();
382
383      String newMessage = "";
384      newMessage += "<net id='" + toHex("" + database.getCoordinator()) + "' time='" + toHex("" + timestamp)
             + "'>\n";
385      newMessage += data;
386      newMessage += "</net>\n";
387
388      return newMessage;
389    }
390
391    /**
392     * Append str to output file. Assumes that the file exists.
393     *
394     * @param str
395     *            String to append
396     */
397    public synchronized void appendToOutputBuffer(String str) {
398      // Remove tabs and enters
399      str = SystemModule.removeChar(str, '\n');
400      str = SystemModule.removeChar(str, '\r');
401      str = SystemModule.removeChar(str, '\t');
402
403      try {
404        fconn = (FileConnection) Connector.open("file:///" + outputFileName, Connector.READ_WRITE);
405        OutputStream os = fconn.openOutputStream(fconn.fileSize());
406        DataOutputStream dos = new DataOutputStream(os);
407
408        byte[] contentInByte = str.getBytes();
409        dos.write(contentInByte);
410        dos.flush();
411
412        dos.close();
413        os.close();
414        fconn.close();
415      } catch (IOException e) {
416        log.write(e.getMessage());
417      }
418    }
419
420    /**
421     * Removes the first &amp;#039;written&amp;#039; amount of characters from the file.
422     *
423     * @param written
424     */
425    public synchronized void updateOutputBuffer(int written) {
426      String str = "";
427      try {
428        fconn = (FileConnection) Connector.open("file:///" + outputFileName, Connector.READ);
429        InputStream in = fconn.openInputStream();
430
431        int ch;
432        while ((ch = in.read()) != -1) {
433          str += (char) ch;
434        }
435
436        // Close connections
437        in.close();
438        fconn.close();
439      } catch (EOFException ioe) {
440        log.write("updateOutputBuffer " + ioe.toString());
441      } catch (IOException ioe) {
442        log.write("updateOutputBuffer " + ioe.toString());
443      }
444
445      // If file read and containts text
446      if (str.length() > 0) {
447
```

```
448       // Remove written chars from file buffer
449       if (written < (str.length() + 1)) {
450         str = str.substring(written);
451       }
452
453       // Write new content
454       try {
455         fconn = (FileConnection) Connector.open("file:///" + outputFileName, Connector.READ_WRITE);
456         fconn.truncate(0l);// Removes content
457         OutputStream os = fconn.openOutputStream();
458         // Opens from beginning
459
460         byte[] contentInByte = str.getBytes();
461         os.write(contentInByte);
462         os.flush();
463
464         os.close();
465         fconn.close();
466       } catch (IOException e) {
467         log.write("updateOutputBuffer " + e.toString());
468       }
469     }
470   }
471
472   /**
473    * Read the output file buffer.
474    *
475    * @return
476    */
477   public synchronized String readFromOutputBuffer() {
478     String str = "";
479     try {
480       // read from file
481       fconn = (FileConnection) Connector.open("file:///" + outputFileName, Connector.READ);
482       InputStream in = fconn.openInputStream();
483
484       int ch;
485       while ((ch = in.read()) != -1) {
486         str += (char) ch;
487       }
488
489       // Close connections
490       in.close();
491       fconn.close();
492     } catch (EOFException ioe) {
493       log.write("readFromOutputBuffer " + ioe.toString());
494     } catch (IOException ioe) {
495       log.write("readFromOutputBuffer " + ioe.toString());
496     }
497     return str;
498   }
499
500   /**
501    * Returns the file &#039;s size
502    *
503    * @return
504    */
505   public synchronized long readOutputBufferSize() {
506     long size = 0;
507     try {
508       // read from file
509       fconn = (FileConnection) Connector.open("file:///" + outputFileName, Connector.READ);
510       size = fconn.fileSize();
511     } catch (IOException ioe) {
512       log.write("readFromOutputBuffer " + ioe.toString());
513     }
514     return size;
515   }
516
517   /**
518    * Returns the hexadecimal representation of the String. Max 19 chars long.
519    *
```

```
520     * @param str
521     * @return
522     */
523    public String toHex(String str) {
524      String s = "";
525      int max = Integer.MAX_VALUE / 10;
526      String maxStr = "" + max;
527      try {
528        if (str.length() > maxStr.length()) {
529          // String too large for Integer.
530          long l = Long.parseLong(str);
531          s = SystemModule.longtoHex(l, true);
532        } else {
533          int a = Integer.parseInt(str);
534          s = Integer.toHexString(a).toUpperCase();
535        }
536      } catch (NumberFormatException e) {
537        log.write("toHex(" + str + ") -> " + e.getMessage() + "" + e.toString());
538        return str;
539      }
540      return s;
541    }
542  }
```

C.8 Database.java

```
1  package src;
2
3  import java.util.Enumeration;
4  import java.util.Vector;
5
6  /**
7   * Holds a list of nodes
8   * @author M.A. Janssen
9   *
10  */
11 public class Database {
12   private final int MAX_VALUES = 20;
13   private Log log;
14   private String coordinator = "";
15   private final boolean DEBUG_MODE = false;
16
17   // Node
18   // - Seq
19   // - measurementdata
20   // - timestamp
21
22   private Vector nodeList;
23
24   public Database() {
25     nodeList = new Vector();
26     log = new Log("database.txt");
27   }
28
29   /**
30    * Append node to list
31    *
32    * @param node
33    */
34   public void appendNode(Node node) {
35     synchronized (nodeList) {
36       nodeList.addElement(node);
37     }
38   }
39
40   /**
41    * Returns if the node is known
42    *
43    * @param node
44    * @return
45    */
```

```
46  public boolean isKnown(Node node) {
47    return nodeList.contains(node);
48  }
49
50  /**
51   * Returns the index of the node in the list or -1 if not found
52   *
53   * @param IEEE
54   * @return
55   */
56  public int getNodeIndex(String IEEE) {
57    if (DEBUG_MODE) {
58      log.write("getNodeIndex(" + IEEE + ")");
59    }
60
61    if (nodeList.isEmpty()) {
62      return -1;
63    }
64
65    for (Enumeration e = nodeList.elements(); e.hasMoreElements();) {
66      Node tempNode = (Node) e.nextElement();
67      if (tempNode.getIEEE().equals(IEEE)) {
68        return nodeList.indexOf(tempNode);
69      }
70    }
71
72    return -1;
73  }
74
75  /**
76   * Returns the node associated with the index
77   *
78   * @param index
79   * @return The node or NULL if not found
80   */
81  public Node getNode(int index) {
82    Node tempNode;
83    if (index >= nodeList.size() || index < 0) {
84      if (DEBUG_MODE) {
85        log.write("getNode(" + index + ") returns null");
86      }
87      return null;
88    }
89    tempNode = (Node) nodeList.elementAt(index);
90    return tempNode;
91  }
92
93  /**
94   * Returns the node, based on the IEEE address or null if not found
95   *
96   * @param ieee
97   * @return
98   */
99  public Node getNode(String ieee) {
100   Node tempNode;
101
102   if (nodeList.isEmpty()) {
103     if (DEBUG_MODE) {
104       log.write("getNode(" + ieee + ") returns null");
105     }
106     return null;
107   }
108
109   for (Enumeration e = nodeList.elements(); e.hasMoreElements();) {
110     tempNode = (Node) e.nextElement();
111     if (tempNode.getIEEE().equals(ieee)) {
112       return tempNode;
113     }
114   }
115   if (DEBUG_MODE) {
116     log.write("getNode(" + ieee + ") returns null");
117   }
```

```
118     return null;
119   }
120
121   /**
122    * Returns the amount of data samples for each node
123    *
124    * @return
125    */
126   public int getNewSequencesSize() {
127     // log.write("getNewSequencesSize()");
128     int size = 0;
129     try {
130       for (Enumeration e = nodeList.elements(); e.hasMoreElements();) {
131         Node tempNode = (Node) e.nextElement();
132         size += tempNode.getNewSequencesSize();
133       }
134     } catch (Exception e) {
135       log.write("getNewSequencesSize\n" + e.toString() + "\n" + e.getMessage());
136     }
137     return size;
138   }
139
140   /**
141    * Returns the new data samples for each node It also resets the 'new' flag
142    *
143    * @return
144    */
145   public Vector getNewSequences() {
146     if (DEBUG_MODE) {
147       log.write("getNewSequences()");
148     }
149     Vector nodes = new Vector();
150     try {
151       for (Enumeration e = nodeList.elements(); e.hasMoreElements();) {
152         Node tempNode = (Node) e.nextElement(); // Link to Node
153
154         // Get new sequences and save in newNode
155         Vector v = tempNode.getNewSequences(true); // Resets flag
156         if (v.size() > 0) {
157           Node newNode = new Node(tempNode.getIEEE());
158           for (Enumeration eS = v.elements(); eS.hasMoreElements();) {
159             Sequence seqTmp = (Sequence) eS.nextElement();
160             newNode.addSequence(seqTmp);
161           }
162           nodes.addElement(newNode);
163         }
164       }
165     } catch (Exception e) {
166       log.write("getNewSequences -> " + e.getMessage() + " " + e.toString());
167     }
168     return nodes;
169   }
170
171   /**
172    * Returns the sequence of the node
173    *
174    * @param node
175    * @param seq
176    * @return
177    */
178   public Sequence getData(String node, String seq) {
179     if (DEBUG_MODE) {
180       log.write("getDate()");
181     }
182     int index = nodeList.indexOf(node);
183     if (index < 0) {
184       return null;
185     }
186     Node nodeTemp = (Node) nodeList.elementAt(index);
187     return nodeTemp.getSequence(seq);
188   }
189
```

```
190   /**
191    * Sets the IEEE address of the coordinator associated to this network
192    *
193    * @param coordinator
194    */
195   public void setCoordinator(String coordinator) {
196     this.coordinator = coordinator;
197   }
198
199   /**
200    * Returns the coordinator IEEE address (if known)
201    *
202    * @return
203    */
204   public String getCoordinator() {
205     return coordinator;
206   }
207
208   /**
209    * Returns number of known nodes i.e., the size of nodeList.
210    *
211    * @return
212    */
213   public int size() {
214     return nodeList.size();
215   }
216
217   /**
218    * Returns the first element of NodeList
219    *
220    * @return
221    */
222   public Node firstElement() {
223     return (Node) nodeList.firstElement();
224   }
225
226   /**
227    * Returns the index of node n
228    *
229    * @param n
230    * @return
231    */
232   public int indexOf(Node n) {
233     return nodeList.indexOf(n);
234   }
235
236   /**
237    * Returns node at element index
238    *
239    * @param index
240    * @return
241    */
242   public Node elementAt(int index) {
243     return (Node) nodeList.elementAt(index);
244   }
245
246   /**
247    * Returns the elements of nodeList
248    *
249    * @return
250    */
251   public Enumeration elements() {
252     return nodeList.elements();
253   }
254
255   /**
256    * Optimize database, removing redundant sequences
257    */
258   public void optimize() {
259     /*
260      * Based on: sent sequences (not in NEW list) and # of sequences
261      */
```

```
262     synchronized (nodeList) {
263       // Optimize database
264       nodeList.trimToSize();
265     }
266
267     // Optimise every node seperately
268     for (Enumeration e = nodeList.elements(); e.hasMoreElements();) {
269       Node tempNode = (Node) e.nextElement();
270       tempNode.trimToSize(MAX_VALUES);
271     }
272   }
273 }
```

C.9 Node.java

```
1  package src;
2
3  import java.util.Enumeration;
4  import java.util.Vector;
5
6  /**
7   * In Node, sequences are stored with data samples.
8   *
9   * @author M.A. Janssen
10  *
11  */
12 public class Node {
13   private String IEEE = "";
14   private String networkAddres = "";
15   private String coordinatorIEEE = "";
16   private Node closestNode = null;
17   private Vector requestedSequences;
18   private Log log;
19
20   private Vector values;
21   private Vector indexes;
22   private Vector newSequences;
23
24   // Save latest x sequences
25   private final int MAX_SEQ = 255;
26
27   private boolean DEBUG_MODE = false;
28
29   public Node(String IEEE) {
30     this.IEEE = IEEE;
31     values = new Vector();
32     indexes = new Vector();
33     requestedSequences = new Vector();
34     newSequences = new Vector();
35
36     log = new Log("Node.txt");
37     if (DEBUG_MODE) {
38       log.write("Node(" + this.IEEE + ")");
39     }
40   }
41
42   /**
43    * Add sequence
44    *
45    * @param seq
46    */
47   public void addSequence(Sequence seq) {
48     int seqNr = seq.getSequenceNumber();
49
50     int index = indexes.indexOf("" + seqNr);
51     if (index > -1) {
52       // If exists, overwrite it
53       values.setElementAt(seq, index);
54     } else {
55       values.addElement(seq);
56       indexes.addElement("" + seqNr);
```

```
57      }
58    }
59
60    /**
61     * Add sequence to Node
62     *
63     * @param timestamp
64     *            Read from RS-232
65     * @param sequence
66     *            Read from RS-232
67     * @param capacity
68     *            Read from RS-232
69     */
70    public void addSequence(long timestamp, String sequence, int capacity) {
71      if (DEBUG_MODE) {
72        log.write("addSequence(" + timestamp + ", " + sequence + ", " + capacity + ")");
73      }
74
75      int sequenceInt = Integer.parseInt(sequence);
76      sequence = "" + sequenceInt; // removes prefix 0 (if present)
77      Sequence newSeq = new Sequence(timestamp, sequenceInt, capacity);
78
79      int index = indexes.indexOf(sequence);
80      if (index > -1) {
81        // If exists, overwrite it
82        values.setElementAt(newSeq, index);
83      } else {
84        // If not exist, create new
85
86        // insert at highest point
87        if (indexes.size() < 1) {
88          // List is empty
89          values.addElement(newSeq);
90          indexes.addElement(sequence);
91        } else {
92          // Calculate highest point for sequence, starting at the top
93          for (int i = (indexes.size() - 1); i > -1; i--) {
94            int current = Integer.parseInt((String) indexes.elementAt(i));
95            if (current < sequenceInt) {
96              // Insert sequence at next level
97              synchronized (values) {
98                values.insertElementAt(newSeq, (i + 1));
99              }
100             synchronized (indexes) {
101               indexes.insertElementAt(sequence, (i + 1));
102             }
103             break;
104           } else if (i == 0) {
105             // Reached the end without inserting, sequence is lower
106             // then every other sequence
107             synchronized (values) {
108               values.insertElementAt(newSeq, i);
109             }
110             synchronized (indexes) {
111               indexes.insertElementAt(sequence, i);
112             }
113             break;
114           }
115         }
116       }
117     }
118
119     // Set as new for GPRS
120     synchronized (newSequences) {
121       newSequences.addElement(sequence);
122     }
123
124     // Remove from request list, if on it
125     if (requestedSequences.size() > 0) {
126       if (requestedSequences.indexOf(sequence) != -1) {
127         requestedSequences.removeElement(sequence);
128         if (DEBUG_MODE) {
```

```
129           log.write("remove sequence " + sequence + " from request list");
130         }
131       }
132     }
133   }
134
135   /**
136    * Add sample to sequence. Creates sequence if not present to prevent data
137    * loss, but with a false &amp;#039;time of sent&amp;#039; time stamp
138    *
139    * @param sequence
140    *            String identifier of sequence
141    * @param sensorId
142    *            Id of the sensor
143    * @param data
144    *            Data sample itself
145    * @param timestamp
146    *            Time stamp of data measurement (relative)
147    */
148   public void addSample(String sequence, String sensorId, String data, long timestamp) {
149     if (DEBUG_MODE) {
150       log.write("addSample(" + sequence + ", " + sensorId + ", " + data + ", " + timestamp + ")");
151     }
152     // remove prefix 0 (if present)
153     sequence = "" + Integer.parseInt(sequence);
154
155     // Get sequence of create new
156     int index = indexes.indexOf(sequence);
157     if (index < 0) {
158       indexes.addElement(sequence);
159       // Add sample before sequence added, time of sent is unknown
160       values.addElement(new Sequence(timestamp, Integer.parseInt(sequence)));
161       index = indexes.indexOf(sequence);
162     }
163
164     // Add sample to sequence
165     Sequence temp = (Sequence) values.elementAt(index);
166     temp.addSample(timestamp, sensorId, data);
167   }
168
169   /**
170    * Gets the sequence from the values list
171    *
172    * @param seq
173    *            String representation of the sequence number
174    * @return The sequence or null
175    */
176   public Sequence getSequence(String seq) {
177     if (DEBUG_MODE) {
178       log.write("getSequence(" + seq + ")");
179     }
180
181     if (values.size() < 1) {
182       return null;
183     }
184
185     seq = "" + Integer.parseInt(seq); // remove prefix 0 (if present)
186     int index = values.indexOf(seq);
187     if (index < 0) {
188       return null;
189     }
190     return (Sequence) values.elementAt(index);
191   }
192
193   /**
194    * Sets IEEE address of the node
195    *
196    * @param IEEE
197    */
198   public void setIEEE(String IEEE) {
199     this.IEEE = IEEE;
200   }
```

```
201
202   /**
203    * Gets the IEEE address of the node
204    *
205    * @return
206    */
207   public String getIEEE() {
208     return IEEE;
209   }
210
211   /**
212    * Sets the IEEE address of the coordinator
213    *
214    * @param IEEE
215    */
216   public void setCoordinatorIEEE(String IEEE) {
217     this.coordinatorIEEE = IEEE;
218   }
219
220   /**
221    * Gets the IEEE address of the coordinator
222    *
223    * @return
224    */
225   public String getCoordinatorIEEEC() {
226     return coordinatorIEEE;
227   }
228
229   /**
230    * Sets the 16-bits network address of the node
231    *
232    * @param networkAddres
233    */
234
235   public void setNetworkAddres(String networkAddres) {
236     this.networkAddres = networkAddres;
237   }
238
239   /**
240    * Gets the 16-bit network address of the node
241    *
242    * @return
243    */
244   public String getNetworkAddres() {
245     return networkAddres;
246   }
247
248   /**
249    * Sets the closest node of the node
250    *
251    * @param closestNode
252    */
253   public void setClosestNode(Node closestNode) {
254     this.closestNode = closestNode;
255   }
256
257   /**
258    * Gets the closest node of the node
259    *
260    * @return
261    */
262   public Node getClosestNode() {
263     return closestNode;
264   }
265
266   /**
267    * Gets the number of sequences stored
268    *
269    * @return
270    */
271   public Vector getSequences() {
272     return values;
```

```
273   }
274
275   /**
276    * Returns the list of new sequences.
277    *
278    * @param reset
279    *            Resets the new sequence list
280    * @return
281    */
282   public Vector getNewSequences(boolean reset) {
283     if (DEBUG_MODE) {
284       log.write("getNewSequences(" + reset + ")");
285     }
286     Vector v = new Vector();
287
288     // Set as new
289     synchronized (newSequences) {
290       if (newSequences.size() > 0) {
291         for (int i = 0; i < newSequences.size(); i++) {
292           // NewSeq containts the seq numbers
293           String strID = (String) newSequences.elementAt(i);
294           if (indexes.contains(strID)) {
295             int index = indexes.indexOf(strID);
296             v.addElement((Sequence) values.elementAt(index));
297           }
298         }
299
300         if (reset) {
301           // Remove all new seq
302           newSequences.removeAllElements();
303         }
304       }
305     }
306     if (DEBUG_MODE) {
307       log.write("getNewSequences " + newSequences.size() + " -> " + v);
308     }
309     return v;
310   }
311
312   /**
313    * Gets the amount of data samples that are new (haven&#039;t been sent to GPRS)
314    *
315    * @return
316    */
317   public int getNewSequencesSize() {
318     if (DEBUG_MODE) {
319       log.write("getNewSequencesSize()");
320     }
321     int size = 0;
322     String tmp = "";
323     synchronized (newSequences) {
324       for (Enumeration e = newSequences.elements(); e.hasMoreElements();) {
325         String strID = (String) e.nextElement();
326         if (indexes.contains(strID)) {
327           try {
328             int index = indexes.indexOf(strID);
329             tmp += "index = " + index;
330             Sequence tempSeq = (Sequence) values.elementAt(index);
331             tmp += " tempSeq = " + tempSeq;
332             size += tempSeq.getDataSize();
333             tmp += " size: " + size + "\n";
334           } catch (Exception ex) {
335             log.write("getNewSequencesSize ex. " + ex.toString() + ", " + ex.getMessage());
336             log.write("getNewSequencesSize ex. " + ex.toString() + ", " + ex.getMessage() + "->" + tmp);
337           }
338         }
339       }
340     }
341     return size;
342   }
343
344   /**
```

```
345     * Return a list of the missing sequences of the node
346     *
347     * @return
348     */
349    public Vector getMissingSequencesNotRequested() {
350      if (DEBUG_MODE) {
351        log.write("getMissingSequencesNotRequested()");
352      }
353
354      // Vector of sequences to be requested
355      Vector v = new Vector();
356
357      if (indexes.size() < 2) {
358        // Request all previous!!
359        if (indexes.size() < 1) {
360          return v;
361        }
362        // First seq, request for all previous sequences
363        int onlySeq = Integer.parseInt((String) indexes.lastElement());
364        for (int i = onlySeq - 1; i > (onlySeq - MAX_SEQ - 1) && i > -1; i--) {
365          // Add emelent if not requested
366          if (requestedSequences.size() > 0) {
367            // Check if not already requested
368            if (requestedSequences.indexOf("" + i) == -1) {
369              // Not requested yet
370              v.addElement("" + i);
371            }
372          } else {
373            // Not requested yet
374            v.addElement("" + i);
375          }
376        }
377        return v;
378      }
379
380      // Only lookup for the last MAX SEQ sequences
381      int lastSeq = Integer.parseInt((String) indexes.lastElement());
382      int firstSeq = Integer.parseInt((String) indexes.firstElement());
383      firstSeq = ((lastSeq - MAX_SEQ) < firstSeq) ? firstSeq : (lastSeq - MAX_SEQ);
384
385      // Prevent underflow
386      if (firstSeq < 0) {
387        firstSeq = 0;
388      }
389
390      // Index of start (might not exist)
391      int lasSeqIndex = indexes.indexOf("" + lastSeq);
392      int startIndex = 0;
393      for (int i = 0; i < MAX_SEQ; i++) {
394        int tmp = indexes.indexOf("" + (firstSeq + i));
395        if (tmp != -1) {
396          startIndex = tmp;
397          break;
398        }
399      }
400      if (startIndex < 1) {
401        startIndex = 0;
402      }
403      if (lasSeqIndex < 1) {
404        lasSeqIndex = 0;
405      }
406
407      if (DEBUG_MODE) {
408        log.write("Look for missing sequences from " + firstSeq + ", to " + lastSeq + "\nStarting at " +
               startIndex
409            + " until " + lasSeqIndex + " of size " + indexes);
410      }
411
412      int previous = -1;
413      int current;
414      for (int i = startIndex; i < lasSeqIndex + 1; i++) {
415        // encounting
```

```
416       String tempStr = (String) indexes.elementAt(i);
417       current = Integer.parseInt(tempStr);
418       if (previous != -1 && previous < current && (current - previous) > 1) {
419         // A sequence has been skipped
420         int nrOfMissingSeq = (current - previous);
421         for (int j = 1; j < nrOfMissingSeq; j++) {
422           // Sequences are not sequential
423           if (requestedSequences.size() > 0) {
424             // Check if not already requested
425             if (requestedSequences.indexOf("" + (previous + j)) == -1) {
426               // Not requested yet
427               v.addElement("" + (previous + j));
428             }
429           } else {
430             // Not requested yet
431             v.addElement("" + (previous + j));
432           }
433         }
434       }
435       previous = current;
436     }
437
438     if (DEBUG_MODE) {
439     log.write("requested seq: " + v);
440     }
441     if(v.size() > MAX_SEQ){
442       Vector g = new Vector();
443
444       // Only request latest 20, Vector.setSize only saves the first 20
445       int lastIndex = v.size()-1;
446       for(int i=lastIndex; (lastIndex-MAX_SEQ) < i; i--){
447         g.addElement( v.elementAt(i) );
448       }
449       v=g;
450
451       if (DEBUG_MODE) {
452       log.write("Reduced requested seq: " + v);
453       }
454     }
455     return v;
456   }
457
458   /**
459    * Reduces the amount of sequences stored to a new size.
460    *
461    * @param capacity
462    *            New capacity
463    */
464   public void trimToSize(int capacity) {
465     if (DEBUG_MODE) {
466       log.write("trimToSize(" + capacity + ")");
467     }
468
469     // Only perform when needed
470     if (indexes.size() <= capacity) {
471       return;
472     }
473
474     // Define oldest new seq
475     int oldestNew = -1;
476     for (Enumeration e = newSequences.elements(); e.hasMoreElements();) {
477       String strID = (String) e.nextElement();
478       int tempInt = Integer.parseInt(strID);
479       if (oldestNew == -1 || tempInt < oldestNew) {
480         oldestNew = tempInt;
481       }
482     }
483
484     // Define latest new seq
485     int latestNew = Integer.MAX_VALUE;
486     for (Enumeration e = newSequences.elements(); e.hasMoreElements();) {
487       String strID = (String) e.nextElement();
```

```
488       int tempInt = Integer.parseInt(strID);
489       if (latestNew == Integer.MAX_VALUE || tempInt > latestNew) {
490         latestNew = tempInt;
491       }
492     }
493
494     // Check latest seq te remove according to maxCapacity
495     String lastSequenceNr = (String) indexes.lastElement();
496     int removeUntil = Integer.parseInt(lastSequenceNr) - capacity;
497
498     // If more sequences are new
499     if (oldestNew != -1 && oldestNew < removeUntil) {
500       removeUntil = oldestNew;
501     }
502
503     String debug = "";
504     debug += "indexes.size = " + indexes.size() + "values.size = " + values.size() + "\n";
505     try {
506       // Remove elements
507       synchronized (indexes) {
508         synchronized (values) {
509           for (int i = 0; (indexes.size() - capacity) > 0; i++) {
510             String tempSequenceNr = (String) indexes.elementAt(0);
511             int current = Integer.parseInt(tempSequenceNr);
512
513             if (current < removeUntil) {
514               // Remove element
515               indexes.removeElementAt(0);
516               values.removeElementAt(0);
517
518             } else {
519               // Done
520               break;
521             }
522           }
523
524           // remove top
525           if (latestNew == Integer.MAX_VALUE) {
526             int removeFrom = indexes.indexOf(""+latestNew);
527             for (int i = indexes.size() - 1; removeFrom < i; i--) {
528               debug += "remove indexes["+i+"] = " + (String) indexes.elementAt(i);
529             }
530           }
531         }
532       }
533     } catch (Exception e) {
534       log.write("trimToSize EXCEPTION " + e.toString());
535     }
536
537     // DEBUG
538     if (DEBUG_MODE) {
539       log.write("trimToSize debug\n" + debug);
540       log.write("indexes.size() = " + indexes.size() + "values.size() = " + values.size());
541     }
542   }
543
544   /**
545    * Adds a request for a specific sequence to the list
546    *
547    * @param sequenceNumber
548    *            Number of the sequence to request
549    */
550   public void setRequestForSequence(int sequenceNumber) {
551     if (DEBUG_MODE) {
552       log.write("setRequestForSequence(" + sequenceNumber + ")");
553     }
554     if (requestedSequences.indexOf("" + sequenceNumber) == -1) {
555       requestedSequences.addElement("" + sequenceNumber);
556     }
557   }
558
559   /**
```

```
560     * Returns the list of already requested sequences
561     *
562     * @return Requested sequences.
563     */
564    public Vector getRequestedSequences() {
565      if (DEBUG_MODE) {
566        log.write("getRequestedSequences()");
567      }
568      return requestedSequences;
569    }
570
571    /**
572     * Returns a String object representing this Node&amp;#039;s value.
573     *
574     * @see java.lang.Object#toString()
575     */
576    public String toString() {
577      String str = "Node (" + this.IEEE + ") with coordinator " + this.coordinatorIEEE + "\n";
578      str += "Samples:\n\t";
579
580      if (this.values.size() < 1) {
581        return str;
582      }
583
584      for (Enumeration valueList = values.elements(); valueList.hasMoreElements();) {
585        Sequence tempSequence = (Sequence) valueList.nextElement();
586        str += tempSequence.toString();
587        str += "\n\t";
588      }
589      return str;
590    }
591  }
```

C.10 Sequence.java

```
1  package src;
2
3  import java.util.Date;
4  import java.util.Enumeration;
5  import java.util.Vector;
6
7  /**
8   * @author M.A. Janssen
9   *
10  */
11 public class Sequence {
12   private long timeStamp;
13   private Date rtcTimestamp;
14   private int sequenceNumber;
15   private Vector dataSamples;
16   private boolean sent = false;
17   Log log;
18
19   /**
20    * Create new sequence with default capacity of 1
21    *
22    * @param timeStamp
23    * @param sequenceNumber
24    */
25   Sequence(long timeStamp, int sequenceNumber) {
26     this(timeStamp, sequenceNumber, 1);
27   }
28
29   /**
30    * Creates new sequence
31    *
32    * @param timeStamp
33    * @param sequenceNumber
34    * @param capacity
35    */
36   Sequence(long timeStamp, int sequenceNumber, int capacity) {
```

```
37      this.timeStamp = timeStamp;
38      this.sequenceNumber = sequenceNumber;
39      this.dataSamples = new Vector(capacity);
40      this.rtcTimestamp = SystemModule.getDate();
41      log = new Log("Sequence.txt");
42    }
43    /**
44     * Sets the sequence number
45     *
46     * @param sequenceNumber
47     */
48    public void setSequenceNumber(int sequenceNumber) {
49      this.sequenceNumber = sequenceNumber;
50    }
51
52    /**
53     * Gets the sequence number
54     *
55     * @return Sequence number
56     */
57    public int getSequenceNumber() {
58      return this.sequenceNumber;
59    }
60
61    /**
62     * Sets the time stamp
63     *
64     * @param timeStamp
65     */
66    public void setTimeStamp(long timeStamp) {
67      this.timeStamp = timeStamp;
68    }
69
70    /**
71     * Gets the time stamp
72     *
73     * @return Time stamp
74     */
75    public long getTimeStamp() {
76      return this.timeStamp;
77    }
78
79    /**
80     * Gets the time stamp
81     *
82     * @return Time stamp
83     */
84    public Date getrtcTimeStamp() {
85      return this.rtcTimestamp;
86    }
87
88    /**
89     * Sets that the sequence has been sent
90     *
91     * @param timeStamp
92     */
93    public void setSent(boolean sent) {
94      this.sent = sent;
95    }
96
97    /**
98     * Gets if the sequence is sent
99     *
100    * @return Sent
101    */
102   public boolean isSent() {
103     return this.sent;
104   }
105
106   /**
107    * Adds a sample to the sequence
108    *
```

```
109     * @param timeStamp
110     * @param sensorType
111     * @param dataSample
112     */
113    public void addSample(long timeStamp, String sensorType, String dataSample) {
114      DataSample sample = new DataSample(timeStamp, sensorType, dataSample);
115      this.dataSamples.addElement(sample);
116    }
117
118    /**
119     * Gets the amount of data samples for this sequence
120     *
121     * @return
122     */
123    public int getDataSize() {
124      return dataSamples.size();
125    }
126
127    /**
128     * Gets all data samples
129     *
130     * @return
131     */
132    public Vector getDataSamples() {
133      return dataSamples;
134    }
135
136    /**
137     * Get specific data sample
138     *
139     * @param index
140     * @return
141     */
142    public DataSample getDataSample(int index) {
143      return (DataSample) dataSamples.elementAt(index);
144    }
145
146    /**
147     * Trims the vector to save memory
148     */
149    public void trimToSize() {
150      this.dataSamples.trimToSize();
151    }
152
153    /*
154     * (non-Javadoc)
155     *
156     * @see java.lang.Object#toString()
157     */
158    public String toString() {
159      String str = "Sequence " + this.sequenceNumber + ", sent at " + this.timeStamp + ", received at "
160          + this.rtcTimestamp;
161
162      if (this.dataSamples.size() < 1) {
163        return str;
164      }
165
166      for (Enumeration indexesList = dataSamples.elements(); indexesList.hasMoreElements();) {
167        DataSample sample = (DataSample) indexesList.nextElement();
168        str += "\n\t\t";
169        str += sample.toString();
170      }
171      return str;
172    }
173  }
```

C.11 DataSample.java

```
1  package src;
2  /**
3   * Holds record of the data sample
```

```
4  * @author M.A. Janssen
5  */
6 public class DataSample {
7   private long timeStamp;
8   private String sensorType;
9   private String dataSample;
10
11  DataSample(long timeStamp, String sensorType, String dataSample) {
12    this.timeStamp = timeStamp;
13    this.sensorType = sensorType;
14    this.dataSample = dataSample;
15  }
16
17  /**
18   * Sets the time stamp
19   * @param timeStamp
20   */
21  public void setTimeStamp(long timeStamp) {
22    this.timeStamp = timeStamp;
23  }
24
25  /**
26   * Gets the time stamp
27   * @return Time stamp
28   */
29  public long getTimeStamp() {
30    return this.timeStamp;
31  }
32
33  /**
34   * Sets the sensor type
35   * @param sensorType
36   */
37  public void setSensorType(String sensorType) {
38    this.sensorType = sensorType;
39  }
40
41  /**
42   * Gets the sensor type
43   * @return Sensor type
44   */
45  public String getSensorType() {
46    return this.sensorType;
47  }
48
49  /**
50   * Sets the sata sample
51   * @param dataSample
52   */
53  public void setDataSample(String dataSample) {
54    this.dataSample = dataSample;
55  }
56
57  /**
58   * Gets the data sample
59   * @return Data sample
60   */
61  public String getDataSample() {
62    return this.dataSample;
63  }
64
65  public String toString(){
66    String str = "At " + this.timeStamp + ": " + this.sensorType + " -> " + this.dataSample;
67    return str;
68  }
69 }
```

D

ZigBee application

D.1 OSAL GenericApp.c

```
/**************************************************************************************************
  Filename:       OSAL_GenericApp.c
  Revised:        $Date: 2007-11-08 08:51:47 -0800 (Thu, 08 Nov 2007) $
  Revision:       $Revision: 15894 $

  Description:    This file contains all the settings and other functions
                  that the user should set and change.
**************************************************************************************************/

/*********************************************************************
 * INCLUDES
 */
#include "ZComDef.h"
#include "hal_drivers.h"
#include "OSAL.h"
#include "OSAL_Tasks.h"
#include "OSAL_Custom.h"

#if defined(ZDO_COORDINATOR)
  #include "Serial.h" // MJA
#endif

#include "nwk.h"
#include "APS.h"
#include "ZDApp.h"

#include "GenericApp.h"

/*********************************************************************
 * GLOBAL VARIABLES
 */

// The order in this table must be identical to the task initialization calls below in osalInitTask.
const pTaskEventHandlerFn tasksArr[] = {
  macEventLoop,
  nwk_event_loop,
  Hal_ProcessEvent,
#if defined(ZDO_COORDINATOR)
  Serial_ProcessEvent, // MJA
#endif
  APS_event_loop,
  ZDApp_event_loop,
  GenericApp_ProcessEvent
};

const uint8 tasksCnt = sizeof( tasksArr ) / sizeof( tasksArr[0] );
uint16 *tasksEvents;

/*********************************************************************
 * FUNCTIONS
 *********************************************************************/

/*********************************************************************
 * @fn      osalInitTasks
 *
 * @brief   This function invokes the  initialization  function for each task.
 *
 * @param  void
 *
 * @return  none
```

```
61  */
62 void osalInitTasks( void )
63 {
64   uint8 taskID = 0;
65
66   tasksEvents = (uint16 *)osal_mem_alloc( sizeof( uint16 ) * tasksCnt);
67   osal_memset( tasksEvents, 0, (sizeof( uint16 ) * tasksCnt));
68
69   macTaskInit( taskID++ );
70   nwk_init( taskID++ );
71   Hal_Init( taskID++ );
72
73 #if defined(ZDO_COORDINATOR)
74   Serial_Init( taskID++ ); // MJA
75 #endif
76
77   APS_Init( taskID++ );
78   ZDApp_Init( taskID++ );
79   GenericApp_Init( taskID );
80 }
81
82 /*********************************************************************
83 *********************************************************************/
```

D.2 Serial.h

```
1  /**************************************************************************************************
2    Filename:       SERIAL.h
3    Revised:        February 10 2010
4    Revision:       1
5
6    Description:    This header describes the functions that handle the serial port.
7
8  **************************************************************************************************/
9
10 #ifndef SERIAL_H
11 #define SERIAL_H
12
13 #ifdef __cplusplus
14 extern "C"
15 {
16 #endif
17
18 /**************************************************************************************************
19  *                                            INCLUDES
20  **************************************************************************************************/
21 #include "Onboard.h"
22 #include "APSMEDE.h"
23 /**************************************************************************************************
24  *                                           CONSTANTS
25  **************************************************************************************************/
26 #define SOP_VALUE     0x02
27 #define SPI_MAX_LENGTH 128
28
29 // UART communication
30 #define SERIAL_SYNC1      0xAA //170
31 #define SERIAL_SYNC2      0x55 //85
32 #define SERIAL_EXT_ADD_LEN    (64/8)
33 #define SERIAL_SHORT_ADD_LEN   (16/8)
34 #define MAX_SEQ_RESPONDS   10
35 #ifndef MAC_ADDR_LEN
36   #define MAC_ADDR_LEN     8
37 #endif
38
39 /* Default values */
40 #define SERIAL_PORT       HAL_UART_PORT_0
41 #define SERIAL_OVERFLOW       TRUE
42
43 #define SERIAL_BAUDRATE       HAL_UART_BR_115200
44 #define SERIAL_THRESHOLD      48
45 #define SERIAL_MAX_RX_BUFF    128
```

```
#define SERIAL_MAX_TX_BUFF   255 // 128 by default
#define SERIAL_IDLE_TIMEOUT  6

// Receive buffer of RS-232
#define SERIAL_REQ_BUF    (2 + 2 + SERIAL_EXT_ADD_LEN + 1 + (MAX_SEQ_RESPONDS * 2) + 1)
#define SERIAL_SYS_APP_MSG    0x23    // Data received from RS232

#define MAX_SAMPLE_STORE   20 // in coordinator

/*********************************************************************************************
 *                                          MACROS
 *********************************************************************************************/
typedef struct {
  osal_event_hdr_t hdr;
  uint16       address16;
  uint8              appDataLen;
  uint8              appData[MAX_SAMPLE_STORE * 2];
} serialSysAppMsg_t;

/*
 * Initialization
 */
extern void Serial_Init ( byte task_id );

extern UINT16 Serial_ProcessEvent( byte task_id, UINT16 events );

/*
 * Process
 */
extern void Serial_ProcessData ( uint8 port, uint8 event );

/*********************************************************************************************
*********************************************************************************************/
#ifdef __cplusplus
}
#endif

#endif //SERIAL H
```

D.3 Serial.c

```
/*********************************************************************************************
  Filename:       Serial.c
  Revised:        February 11 2010
  Revision:       1

  Description:    This module handles anything dealing with the serial port.
*********************************************************************************************/

/*********************************************************************************************
 *                                          INCLUDES
 *********************************************************************************************/
#include "ZComDef.h"
#include "OSAL.h"
#include "hal_uart.h"
#include "MTEL.h"
#include "Serial.h"
#include "OSAL_Memory.h"

/*********************************************************************************************
 *                                       GLOBAL VARIABLES
 *********************************************************************************************/
byte Serial_TaskID;
extern byte GenericApp_TaskID;

/*********************************************************************************************
 *                                       LOCAL FUNCTIONS
 *********************************************************************************************/

/*********************************************************************************************
 * @fn      Serial_Init
```

```
31  *
32  * @param  None
33  *
34  * @return  None
35 ******************************************************************************************/
36 void Serial_Init ( byte task_id )
37 {
38    Serial_TaskID = task_id;
39
40   // UART Configuration
41   halUARTCfg_t uartConfig;
42   uartConfig.configured           = TRUE;
43   uartConfig.baudRate             = SERIAL_BAUDRATE;
44   uartConfig.flowControl          = SERIAL_OVERFLOW;
45   uartConfig.flowControlThreshold = SERIAL_THRESHOLD;
46   uartConfig.rx.maxBufSize        = SERIAL_MAX_RX_BUFF;
47   uartConfig.tx.maxBufSize        = SERIAL_MAX_TX_BUFF;
48   uartConfig.idleTimeout          = SERIAL_IDLE_TIMEOUT;
49   uartConfig.intEnable            = TRUE;
50   uartConfig.callBackFunc         = Serial_ProcessData; // Callback
51   HalUARTOpen (SERIAL_PORT, &uartConfig);
52
53
54 }
55
56 /*********************************************************************
57  * @fn      Serial_ProcessEvent
58  *
59  * @brief
60  *
61  *   Event Processor.  This task is put into the
62  *   task table.
63  *
64  * @param  byte task_id - task ID of the Task
65  * @param  UINT16 events - event(s) for the Task
66  *
67  * @return  void
68  *********************************************************************/
69 UINT16 Serial_ProcessEvent( byte task_id, UINT16 events ) {
70   uint8 *msg_ptr;
71
72   // Could be multiple events, so switch won't work
73   if ( events & SYS_EVENT_MSG ) {
74     while ( (msg_ptr = osal_msg_receive( Serial_TaskID )) ) {
75       //Process event, not implemented
76     //MT_ProcessCommand( (mtOSALSerialData_t *)msg_ptr );
77     }
78
79     // Return unproccessed events
80     return (events ^ SYS_EVENT_MSG);
81   }
82
83    // Discard or make more handlers
84   return 0;
85 }
86
87
88 /*********************************************************************************************
89  * Decode received message and send it to GenericApp
90  *********************************************************************************************/
91 void Serial_ProcessData ( uint8 port, uint8 event ){
92   uint8 ch;
93   uint8 msg[ SERIAL_REQ_BUF ];
94
95   uint8 nodeAddressIEEE[ MAC_ADDR_LEN ];
96   uint16 nodeAddress16;
97   uint16 nrofSeq = 0;
98
99   /* Verify events */
100  if (event == HAL_UART_TX_FULL)
101  {
102    // Do something when TX if full
```

```
103     return;
104   }
105
106   if (event & (HAL_UART_RX_FULL | HAL_UART_RX_ABOUT_FULL | HAL_UART_RX_TIMEOUT))
          {
107   uint8 length = 0;
108     while ( Hal_UART_RxBufLen( SERIAL_PORT ) ) {
109       HalUARTRead ( SERIAL_PORT, &ch, 1 );
110
111     // Confirm message, ignore
112     if( length == 0 && ch == 0x04 ){
113     return;
114     }
115
116     // Prevent overflow
117     if( length < SERIAL_REQ_BUF ){
118       msg[ length ] = ch;
119       length++;
120     }
121     }
122
123   // Do something with message and length
124   if(length < 2){
125     // End character
126     return;
127   }
128   if( msg[0]== SERIAL_SYNC1 && msg[1] == SERIAL_SYNC2 ){
129     // Message from GPRS module, send to genericApp
130
131     // if response message, length is at least 15
132     if( length > 14 && msg[2] == 0x00 && msg[3] == SERIAL_EXT_ADD_LEN){
133     // Update pionter
134     uint8 index = 4;
135
136     // Store extended address
137     for( uint8 j=0; j < MAC_ADDR_LEN; j++ ){
138       nodeAddressIEEE[ MAC_ADDR_LEN - j - 1] = msg[ j + index ];
139     }
140     index += MAC_ADDR_LEN;
141
142     // Get number of sequences to request
143     nrofSeq = msg[ index ];
144     if(nrofSeq > 0){
145       index += 1;
146
147       // Prevent overflow
148       if( nrofSeq > MAX_SAMPLE_STORE ){
149       nrofSeq = MAX_SAMPLE_STORE;
150       }
151
152       // Use nrofSeq for length, 2*uint8
153       nrofSeq *= 2;
154
155        // Get Node information
156       if(!APSME_LookupNwkAddr(nodeAddressIEEE, &nodeAddress16)){
157       // Address unknown, ignore message
158       //return;
159       nodeAddress16=0;
160       }
161
162       // Get all sequences to request
163       uint16 seqToReq[MAX_SAMPLE_STORE];
164       for( uint8 n=0; n < nrofSeq; n++ ){
165         seqToReq[n] = msg[ index + n];
166       }
167
168       //allocate a message buffer
169       serialSysAppMsg_t* msgForGeneric = (serialSysAppMsg_t*)osal_msg_allocate( sizeof( serialSysAppMsg_t
              ) + length );
170       if ( msgForGeneric ) {
171         msgForGeneric->hdr.event = SERIAL_SYS_APP_MSG;
172         msgForGeneric->address16 = nodeAddress16;
```

```
173         msgForGeneric->appDataLen = nrofSeq;
174         for( uint8 n=0; n < nrofSeq; n++ ){
175           msgForGeneric->appData[n] = seqToReq[n];
176         }
177
178         osal_msg_send( GenericApp_TaskID, (uint8 *)msgForGeneric );
179
180       }
181     }
182     }
183   }
184   }
185 }
```

D.4 GenericApp.h

```
1  /*********************************************************************************************
2    Filename:       GenericApp.h
3    Revised:        February 11 2010
4    Revision:       1
5
6    Description:    Modified genericApp for creating a WSN using ZigBee and
7            GPRS, with data retrieval.
8  *********************************************************************************************/
9
10 #ifndef GENERICAPP_H
11 #define GENERICAPP_H
12
13 #ifdef __cplusplus
14 extern "C"
15 {
16 #endif
17
18 /*********************************************************************************************
19  * INCLUDES
20  */
21 #include "ZComDef.h"
22 #if defined(ZDO_COORDINATOR)
23   #include "Serial.h"
24 #endif
25 #if defined( POWER_SAVING )
26   #include "OSAL_PwrMgr.h"
27 #endif
28
29 /*********************************************************************************************
30  * CONSTANTS
31  */
32
33 // These constants are only for example and should be changed to the
34 // device's needs
35 #define GENERICAPP_ENDPOINT       20
36
37 #define GENERICAPP_PROFID         0x0F08
38 #define GENERICAPP_DEVICEID       0x0001
39 #define GENERICAPP_DEVICE_VERSION 0
40 #define GENERICAPP_FLAGS          0
41
42 #define GENERICAPP_MAX_CLUSTERS   4
43   #define GENERICAPP_FLASH_CLUSTERID 2
44   #define GENERICAPP_AD_ANSWER_CLUSTERID 3
45   #define GENERICAPP_AD_QUESTION_CLUSTERID 4
46   #define GENERICAPP_REQUEST_SEQ_CLUSTERID 5
47
48 // Send Message Timeout
49 #define GENERICAPP_SEND_MSG_TIMEOUT 65535   // Every 65 seconds
50
51 // Application Events (OSAL) - These are bit weighted definitions.
52 #define GENERICAPP_SEND_MSG_EVT   0x0001
53
54 // Group ID for Flash Command
55 #define GENERICAPP_FLASH_GROUP    0x0001
```

```
56
57 // UART communication
58 #define DATA_LEN_LEN      1
59
60 #define SERIAL_END_QUANTITY_LEN  2
61
62 #define SERIAL_SEQ_LEN     2
63 #define SERIAL_SAMPLE_QUANTITY_LEN  2
64 #define SERIAL_SAMPLE_LEN    2
65
66 #define SERIAL_TYPE_LEN    2
67 #define SERIAL_TIMESTAMP_LEN   4
68
69 #define SERIAL_HEADER_LEN   2 + DATA_LEN_LEN + MAC_ADDR_LEN + SERIAL_END_QUANTITY_LEN + MAC_ADDR_LEN//21
70
71 #define END_DEVICE_HEADER_LEN  8
72 #define END_DEVICE_buflen    10
73
74 // Buffer
75 #define END_D_MAX_SEQ_LEN   10
76 #define END_D_MAX_BUFFER   320
77
78 // LED format
79 #define LED_MSG_INIT     200
80 #define LED_MSG_SEND     316
81 #define LED_MSG_REQ        360
82
83 /*********************************************************************************************
84  * MACROS
85  */
86 #if !defined(ZDO_COORDINATOR)
87 typedef struct
88 {
89   uint16       msgSequence;
90   uint8        appDataLen;
91   uint8        buffer[END_DEVICE_buflen];
92 } dataSample;
93 #endif
94 /*********************************************************************************************
95  * FUNCTIONS
96  */
97
98 /*
99  * Task Initialization for the Generic Application
100  */
101 extern void GenericApp_Init( byte task_id );
102
103 /*
104  * Task Event Processor for the Generic Application
105  */
106 extern UINT16 GenericApp_ProcessEvent( byte task_id, UINT16 events );
107
108 #if defined(ZDO_COORDINATOR)
109   /*
110   * Request for sequence(s) event handler
111   */
112   extern void GenericApp_unicastMsg(serialSysAppMsg_t *pkt );
113   extern void AskForAdValue(uint8 channel, uint8 resolution);//DAWIJK
114 #else
115   extern void AnswerWithADValue(uint8 channel, uint8 resolution);//DAWIJK
116   extern void sendBuffered( uint8 length , uint8 *msg );
117 #endif
118
119 extern void FlipLed(unsigned int);//DAWIJK
120 /*********************************************************************
121 *********************************************************************/
122
123 #ifdef __cplusplus
124 }
125 #endif
126
127 #endif // GENERICAPP H
```

D.5 GenericApp.c

```
/*********************************************************************************
  Filename:       GenericApp.c
  Revised:        February 11 2010
  Revision:       1

  Description:    Modified genericApp for creating a WSN using ZigBee and
          GPRS, with data retrieval.

*********************************************************************************/

/*********************************************************************************
 * INCLUDES
 *********************************************************************************/
#include "OSAL.h"
#include "AF.h"
#include "aps_groups.h"
#include "ZDApp.h"
#include "ZDObject.h"
#include "ZDProfile.h"

#include "GenericApp.h"
#include "DebugTrace.h"

#if !defined( WIN32 )
  #include "OnBoard.h"
#endif

/* HAL */
#include "hal_lcd.h"
#include "hal_led.h"
#include "hal_key.h"
#include "hal_uart.h"
#include "hal_adc.h"

/*********************************************************************************
 * MACROS
 *********************************************************************************/
/*********************************************************************************
 * CONSTANTS
 *********************************************************************************/
/*********************************************************************************
 * TYPEDEFS
 *********************************************************************************/
/*********************************************************************************
 * GLOBAL VARIABLES
 *********************************************************************************/

// This list should be filled with Application specific Cluster IDs.
const cId_t GenericApp_ClusterList[GENERICAPP_MAX_CLUSTERS] =
{
  GENERICAPP_FLASH_CLUSTERID,
  GENERICAPP_AD_ANSWER_CLUSTERID,
  GENERICAPP_AD_QUESTION_CLUSTERID,
  GENERICAPP_REQUEST_SEQ_CLUSTERID
};

const SimpleDescriptionFormat_t GenericApp_SimpleDesc =
{
  GENERICAPP_ENDPOINT,     // int Endpoint;
  GENERICAPP_PROFID,       // uint16 AppProfId[2];
  GENERICAPP_DEVICEID,     // uint16 AppDeviceId[2];
  GENERICAPP_DEVICE_VERSION, // int AppDevVer:4;
  GENERICAPP_FLAGS,        // int AppFlags:4;
  GENERICAPP_MAX_CLUSTERS, // byte AppNumInClusters;
  (cId_t *)GenericApp_ClusterList, // byte *pAppInClusterList;
  GENERICAPP_MAX_CLUSTERS, // byte AppNumInClusters;
  (cId_t *)GenericApp_ClusterList // byte *pAppInClusterList;
};

```

```
70 // This is the Endpoint/Interface description. It is defined here, but
71 // filled-in in GenericApp_Init(). Another way to go would be to fill
72 // in the structure here and make it a "const" (in code space). The
73 // way it's defined in this sample app it is define in RAM.
74 endPointDesc_t GenericApp_epDesc;
75
76 /*********************************************************************************
77  * EXTERNAL VARIABLES
78 *********************************************************************************/
79
80 /*********************************************************************************
81  * EXTERNAL FUNCTIONS
82 *********************************************************************************/
83
84 /*********************************************************************************
85  * LOCAL VARIABLES
86 *********************************************************************************/
87 byte GenericApp_TaskID; // Task ID for internal task/event processing
88                         // This variable will be received when
89                         // GenericApp_Init() is called.
90 devStates_t GenericApp_NwkState;
91
92
93 byte GenericApp_TransID; // This is the unique message ID (counter)
94
95 afAddrType_t GenericApp_Flash_DstAddr;
96
97 aps_Group_t GenericApp_Group;
98
99 uint16 msgSequence = 0; // MJA
100
101 #if defined(ZDO_COORDINATOR)
102   // Save msg for node
103   serialSysAppMsg_t pendingMsg[ MAX_SEQ_RESPONDS ];//[MAX_ENDPOINTS];
104   uint8 pendingMsgSize = 0;
105 #else
106   dataSample storedSamples[ END_D_MAX_BUFFER ];
107   uint16 dataSampleNrStored = 0;
108   uint16 dataSampleIndex = 0;
109   uint16 dataSampleLastSent = 65534;
110 #endif
111
112 /*********************************************************************************
113  * LOCAL FUNCTIONS
114  */
115 void GenericApp_ProcessZDOMsgs( zdoIncomingMsg_t *inMsg );
116 void GenericApp_MessageMSGCB( afIncomingMSGPacket_t *pckt );
117 /*********************************************************************************
118  * NETWORK LAYER CALLBACKS
119  */
120
121 /*********************************************************************************
122  * PUBLIC FUNCTIONS
123  */
124
125 /*********************************************************************************
126  * @fn      GenericApp_Init
127  *
128  * @brief   Initialization function for the Generic App Task.
129  *          This is called during initialization and should contain
130  *          any application specific initialization (ie. hardware
131  *          initialization/setup, table initialization, power up
132  *          notificaiton ... ).
133  *
134  * @param  task_id - the ID assigned by OSAL. This ID should be
135  *                   used to send messages and set timers.
136  *
137  * @return  none
138 *********************************************************************************/
139 void GenericApp_Init( byte task_id )
140 {
141   GenericApp_TaskID = task_id;
```

```
142   GenericApp_NwkState = DEV_INIT;
143   GenericApp_TransID = 0;
144
145   // Setup for the periodic message's destination address
146   GenericApp_Flash_DstAddr.addrMode = (afAddrMode_t)afAddrGroup;
147   GenericApp_Flash_DstAddr.endPoint = GENERICAPP_ENDPOINT;
148   GenericApp_Flash_DstAddr.addr.shortAddr = GENERICAPP_FLASH_GROUP;
149
150
151
152   // Fill out the endpoint description.
153   GenericApp_epDesc.endPoint = GENERICAPP_ENDPOINT;
154   GenericApp_epDesc.task_id = &GenericApp_TaskID;
155   GenericApp_epDesc.simpleDesc
156             = (SimpleDescriptionFormat_t *)&GenericApp_SimpleDesc;
157   GenericApp_epDesc.latencyReq = noLatencyReqs;
158
159   // Register the endpoint description with the AF
160   afRegister( &GenericApp_epDesc );
161
162 #if defined(ZDO_COORDINATOR)
163   // Register for all key events - This app will handle all key events
164   RegisterForKeys( GenericApp_TaskID );
165 #endif
166
167    // By default, all devices start out in Group 1
168   GenericApp_Group.ID = 0x0001;
169   osal_memcpy( GenericApp_Group.name, "Group 1", 7 );
170   aps_AddGroup( GENERICAPP_ENDPOINT, &GenericApp_Group );
171
172
173   // Send message
174 #if !defined(ZDO_COORDINATOR)
175    osal_start_timerEx( GenericApp_TaskID,
176                                GENERICAPP_SEND_MSG_EVT,
177                                GENERICAPP_SEND_MSG_TIMEOUT );
178 #endif
179
180    // System online
181    FlipLed( LED_MSG_INIT );
182 }
183
184 /*********************************************************************************
185  * @fn      GenericApp_ProcessEvent
186  *
187  * @brief   Generic Application Task event processor.  This function
188  *          is called to process all events for the task.  Events
189  *          include timers, messages and any other user defined events.
190  *
191  * @param  task_id  - The OSAL assigned task ID.
192  * @param  events - events to process.  This is a bit map and can
193  *                  contain more than one event.
194  *
195  * @return  none
196  *********************************************************************************/
197 UINT16 GenericApp_ProcessEvent( byte task_id, UINT16 events )
198 {
199   afIncomingMSGPacket_t *MSGpkt;
200   //afDataConfirm_t *afDataConfirm;
201
202   // Data Confirmation message fields
203   if ( events & SYS_EVENT_MSG ) {
204     MSGpkt = (afIncomingMSGPacket_t *)osal_msg_receive( GenericApp_TaskID );
205     while ( MSGpkt )
206     {
207       switch ( MSGpkt->hdr.event )
208       {
209     #if defined(ZDO_COORDINATOR)
210       case SERIAL_SYS_APP_MSG:
211         // Callback by serial.c
212         GenericApp_unicastMsg((serialSysAppMsg_t *)MSGpkt);
213         break;
```

```
214     #endif
215         case AF_INCOMING_MSG_CMD:
216           GenericApp_MessageMSGCB( MSGpkt );
217           break;
218
219         case ZDO_STATE_CHANGE:
220           GenericApp_NwkState = (devStates_t)(MSGpkt->hdr.status);
221           if (GenericApp_NwkState == DEV_END_DEVICE)
222           {
223             // Start sending "the" message in a regular interval.
224             osal_start_timerEx( GenericApp_TaskID,
225                                 GENERICAPP_SEND_MSG_EVT,
226                                 GENERICAPP_SEND_MSG_TIMEOUT );
227
228       #if !defined(ZDO_COORDINATOR)
229         // Send all data samples
230         sendBuffered( dataSampleNrStored, NULL );
231
232       #endif
233           }
234           break;
235
236         default:
237           break;
238       }
239
240       // Release the memory
241       osal_msg_deallocate( (uint8 *)MSGpkt );
242
243       // Next
244       MSGpkt = (afIncomingMSGPacket_t *)osal_msg_receive( GenericApp_TaskID );
245     }
246
247     // return unprocessed events
248     return (events ^ SYS_EVENT_MSG);
249   }
250
251   // Send a message out - This event is generated by a timer
252   // (setup in GenericApp_Init()).
253   if ( events & GENERICAPP_SEND_MSG_EVT )
254   {
255
256     // Create periodic event
257 #if !defined(ZDO_COORDINATOR)
258       AnswerWithADValue(0,4);//resolution 4 <=> 14 bits
259
260     // Setup to send message again
261     osal_start_timerEx( GenericApp_TaskID,
262                         GENERICAPP_SEND_MSG_EVT,
263                         GENERICAPP_SEND_MSG_TIMEOUT + (osal_rand() & 0x00FF));
264 #endif
265
266     // return unprocessed events
267     return (events ^ GENERICAPP_SEND_MSG_EVT);
268   }
269
270   // Discard unknown events
271   return 0;
272 }
273
274 /*********************************************************************************
275  * LOCAL FUNCTIONS
276 *********************************************************************************/
277
278 /*********************************************************************************
279  * @fn      GenericApp_MessageMSGCB
280  *
281  * @brief   Data message processor callback.  This function processes
282  *          any incoming data - probably from other devices.  So, based
283  *          on cluster ID, perform the intended action.
284  *
285  * @param  none
```

```
286  *
287  * @return  none
288 ******************************************************************************************/
289 void GenericApp_MessageMSGCB( afIncomingMSGPacket_t *pkt ) {
290   uint16 DataLength;
291   uint8 index = 0;
292   uint8 appDataLen=0;
293   uint8 *appData;
294
295
296   switch ( pkt->clusterId ) {
297 #if !defined(ZDO_COORDINATOR)
298   case GENERICAPP_REQUEST_SEQ_CLUSTERID:
299     // Get from local database
300     sendBuffered( pkt->cmd.DataLength, pkt->cmd.Data );
301     break;
302 #endif
303
304 #if defined(ZDO_COORDINATOR)
305     case GENERICAPP_AD_ANSWER_CLUSTERID:
306     // sample data received
307
308     // First, send data back
309
310     for( uint8 i=0; i < pendingMsgSize; i++ ){
311       if( pendingMsg[ i ].address16 == pkt->srcAddr.addr.shortAddr ){
312           index = i;
313         appDataLen = pendingMsg[i].appDataLen;
314         break;
315       }
316     }
317
318     if(appDataLen > 0){// maybe 0x04
319       if ( AF_DataRequest( &pkt->srcAddr, // Destination address pointer
320                 &GenericApp_epDesc, // Endpoint Descriptor pointer of the sending endpoint
321                 GENERICAPP_REQUEST_SEQ_CLUSTERID, // Request for seq
322                 pendingMsg[ index ].appDataLen,
323                 &pendingMsg[ index ].appData[0],
324                 &GenericApp_TransID,
325                 AF_DISCV_ROUTE, // Should always be included
326               AF_DEFAULT_RADIUS ) == afStatus_SUCCESS ){ // Maximum number of hops
327       // Successfully requested to be sent.
328         // reset variables
329         pendingMsg[ index ].address16 = 0;
330         pendingMsg[ index ].appDataLen = 0;
331       }else{
332       // Error occurred in request to send.
333       }
334     }
335
336
337     // Data
338     DataLength = pkt->cmd.DataLength;
339
340     // *** Send procedure *** //
341     // sync
342     uint8 serialSynch[2] = { SERIAL_SYNC1, SERIAL_SYNC2 };
343     HalUARTWrite( SERIAL_PORT, serialSynch, (2) );
344
345     // Length of package
346     uint8 dataLen[DATA_LEN_LEN] ={ SERIAL_HEADER_LEN + DataLength};
347     HalUARTWrite( SERIAL_PORT, dataLen, DATA_LEN_LEN );
348
349     // MAC of Coor
350     uint8 ieeeAddrCoor[ MAC_ADDR_LEN ] = {0};
351     uint8* ieeetemp = NLME_GetExtAddr();
352     for( uint8 i = 0; i<MAC_ADDR_LEN; i++){
353       ieeeAddrCoor[i] = ieeetemp[i];
354     }
355     HalUARTWrite( SERIAL_PORT, ieeeAddrCoor, MAC_ADDR_LEN );
356
357     // Number of end-devices in this message
```

```
358     uint8 numberDevices[ SERIAL_END_QUANTITY_LEN ] = {0,1}; // STATIC!!
359     HalUARTWrite( SERIAL_PORT, numberDevices, SERIAL_END_QUANTITY_LEN );
360
361     // Per end-device
362     uint8 ieeeAddrEND[ MAC_ADDR_LEN ] = {0};
363     uint16 numberDevices16 = BUILD_UINT16(numberDevices[1],numberDevices[0]);
364     for( uint16 numberDev=0; numberDev < numberDevices16 ; numberDev++){
365
366       // Lookup extended address (if known))
367       uint8 ieeeAddrENDLookup[ MAC_ADDR_LEN ] = {0};
368       if( APSME_LookupExtAddr( pkt->srcAddr.addr.shortAddr, ieeeAddrENDLookup ) ){
369         for(uint8 i=0; i<MAC_ADDR_LEN; i++ ) {
370           ieeeAddrEND[ i ] = ieeeAddrENDLookup[ MAC_ADDR_LEN - i - 1 ];
371         }
372       }else{
373         // Get short address
374         ieeeAddrEND[ MAC_ADDR_LEN - 1] = HI_UINT16(pkt->srcAddr.addr.shortAddr);
375         ieeeAddrEND[ MAC_ADDR_LEN - 2] = LO_UINT16(pkt->srcAddr.addr.shortAddr);
376       }
377       HalUARTWrite( SERIAL_PORT, ieeeAddrEND, MAC_ADDR_LEN );
378
379       // Data package of end-device
380       HalUARTWrite( SERIAL_PORT, pkt->cmd.Data, DataLength );
381     }
382
383     // Write to DISPLAY
384     uint8 line1[16] = {0};
385     uint8 j=0;
386     for(uint8 i=0; i<8;i++){
387       _ltoa( ieeeAddrEND[i], &line1[j], 10 );
388       if(!ieeeAddrEND[i]){
389         line1[j] = '0';
390         line1[j+1]= '0';
391       }
392       j=j+2;
393
394     }
395
396     uint8 line2[16];
397     for(uint8 i=0; i<16;i++){
398       line2[i]=' ';
399     }
400
401     _ltoa( BUILD_UINT16(pkt->cmd.Data[SERIAL_TIMESTAMP_LEN+1], pkt->cmd.Data[
           SERIAL_TIMESTAMP_LEN]), &line2[0], 10 );
402     if(line2[1] == 0 )line2[1] = ' ';
403     if(line2[2] == 0 )line2[2] = ' ';
404     if(line2[3] == 0 )line2[3] = ' ';
405     if(line2[4] == 0 )line2[4] = ' ';
406     if(line2[5] == 0 )line2[5] = ' ';
407
408      line2[6] = '-';
409      line2[7] = '>';
410      _ltoa( BUILD_UINT16(pkt->cmd.Data[DataLength-1], pkt->cmd.Data[DataLength-2]), &line2[9], 10 );
411
412      HalLcdWriteString((char*)line1, HAL_LCD_LINE_1);
413      HalLcdWriteString((char*)line2, HAL_LCD_LINE_2);
414
415   break;
416 #endif //ZDO COORDINATOR
417   }
418 }
419
420 #if defined(ZDO_COORDINATOR)
421 /*********************************************************************************************
422  * @fn      GenericApp_unicastMsg
423  *
424  * @brief   Msg for node, sent by GPRS module
425  *
426  * @param  The message
427  *
428  * @return  none
```

```
429 *********************************************************************************/
430 void GenericApp_unicastMsg(serialSysAppMsg_t *pkt ){
431   // STORE MESSAGE IN GENERICAPP.C UNTIL NODE IS UP
432   uint8 index;
433   for( index = 0; index < pendingMsgSize; index++ ){
434     if( pendingMsg[ index ].address16 == pkt->address16 ){
435       break;
436     }
437   }
438
439   // Prevent overflow
440   index = index % MAX_SEQ_RESPONDS;
441
442   // Save message
443   pendingMsg[ index ].address16 = pkt->address16;
444   pendingMsg[ index ].appDataLen = pkt->appDataLen;
445   for( uint8 i=0; i < pkt->appDataLen; i++ ){
446     pendingMsg[ index ].appData[ i ] = pkt->appData[ i ];
447   }
448
449   pendingMsgSize++;
450   if( pendingMsgSize >= MAX_SEQ_RESPONDS ){
451   pendingMsgSize = MAX_SEQ_RESPONDS - 1;
452   }
453 }
454 #endif
455
456 #if !defined(ZDO_COORDINATOR)
457 /*********************************************************************************
458 * Send sequence to coordinator is reachable
459 *********************************************************************************/
460 void AnswerWithADValue(uint8 channel, uint8 resolution){
461   P1SEL &= 254 ;//1111 1110 //Port 1 pin 1 to GPIO
462   P1DIR |= 1; //0000 0001 //Port 1 pin 1 to output
463   P1_1 = 1;
464
465   // Take sample
466   uint8 adValueLen = 2;
467   uint16 adValue = HalAdcRead(channel, resolution);
468   uint32 sysTime32 = osal_GetSystemClock(); // time of sample
469
470   // If the same as previous data sample, ignore.
471   uint16 previousValue = BUILD_UINT16( storedSamples[dataSampleIndex - 1 % END_D_MAX_BUFFER].
        buffer[END_DEVICE_buflen - 2 ],
472                       storedSamples[dataSampleIndex - 1 % END_D_MAX_BUFFER].buffer[
                            END_DEVICE_buflen - 1 ] );
473   // If no difference, do not save
474   if( previousValue == adValue ){
475     return;
476   }
477   uint8 samples = 1;
478
479   uint8 buffer[ END_DEVICE_HEADER_LEN + END_D_MAX_SEQ_LEN ];
480   uint8  bufferFill  = 0; //END_DEVICE_HEADER_LEN + END_DEVICE_buflen;
481
482   uint16 sensorType = BUILD_UINT16(resolution, channel);
483
484   // Add timestamp at latest point
485    bufferFill  = SERIAL_TIMESTAMP_LEN;
486
487   // Sequence
488   buffer[ bufferFill++] = HI_UINT16(msgSequence);
489   buffer[ bufferFill++] = LO_UINT16(msgSequence);
490
491
492   // Samples
493   for( int i=0; i<SERIAL_SAMPLE_QUANTITY_LEN; i++ ){
494     buffer[ bufferFill++] = (uint8)(((samples) >>((SERIAL_SAMPLE_QUANTITY_LEN - i - 1) * 8)) & 0
          x00FF);
495   }
496
497   // For every sample
```

```
498   for( int sampleI=0; sampleI<samples; sampleI++ ){
499     // Timestamp of measured data
500     for( int i=0; i<SERIAL_TIMESTAMP_LEN && i<4; i++ ){
501       buffer[ bufferFill++] = (uint8)(((sysTime32) >>((SERIAL_TIMESTAMP_LEN - i - 1) * 8)) & 0x00FF);
502     }
503
504     // Sensor type
505     for( int i=0; i<SERIAL_TYPE_LEN; i++ ){
506       buffer[ bufferFill++] = (uint8)(((sensorType) >>((SERIAL_TYPE_LEN - i - 1) * 8)) & 0x00FF);
507     }
508
509     // Data length
510     for( int i=0; i<SERIAL_SAMPLE_LEN; i++ ){
511       buffer[ bufferFill++] = (uint8)(((adValueLen) >>((SERIAL_SAMPLE_LEN - i - 1) * 8)) & 0x00FF);
512     }
513
514     // Data
515     for( int i=0; i<adValueLen; i++ ){
516       buffer[ bufferFill++] = (uint8)(((adValue) >>((adValueLen - i - 1) * 8)) & 0x00FF);
517     }
518   }
519
520   /*
521   header:
522     TimeOfSent     4
523     Sequence       2
524     NumberOfSamples   2
525
526   n * NumberOfSamples
527   - TimeOfMeasurement  4
528     - SensorType   2
529     - DataLength   2
530     - Data         2
531   */
532
533   // SAVE IN LOCAL DATABASE
534   storedSamples[ dataSampleIndex ].msgSequence = msgSequence;
535   storedSamples[ dataSampleIndex ].appDataLen = END_DEVICE_buflen;
536
537   // Discard END_DEVICE_HEADER_LEN
538   for( uint8 i = 0; i < END_DEVICE_buflen; i++ ){
539     storedSamples[ dataSampleIndex ].buffer[ i ] = buffer[ END_DEVICE_HEADER_LEN + i ];
540   }
541   // List fill
542   dataSampleNrStored++;
543   if( dataSampleNrStored > END_D_MAX_BUFFER ){
544     dataSampleNrStored = END_D_MAX_BUFFER;
545   }
546
547   // Update index of newest seq.
548   dataSampleIndex = (dataSampleIndex + 1) % END_D_MAX_BUFFER;
549
550   // Timestamp data sent at latest point possible
551   uint32 sysTime32Send = osal_GetSystemClock(); // returns milliseconds
552   for( uint8 i=0; i < SERIAL_TIMESTAMP_LEN && i < 4; i++ ){
553     buffer[i] = (uint8)(((sysTime32Send) >>((SERIAL_TIMESTAMP_LEN - i - 1) * 8)) & 0x00FF);
554   }
555
556   if ( (GenericApp_NwkState == DEV_END_DEVICE) ){
557   // Only send if connected
558     if ( AF_DataRequest( &GenericApp_Flash_DstAddr, &GenericApp_epDesc,
559               GENERICAPP_AD_ANSWER_CLUSTERID,
560               //bufferFill,
561               END_DEVICE_HEADER_LEN + END_DEVICE_buflen,
562               buffer,
563               &GenericApp_TransID,
564               AF_DISCV_ROUTE,
565           AF_DEFAULT_RADIUS ) == afStatus_SUCCESS )
566     {
567     // Successfully requested to be sent.
568
569     dataSampleLastSent = msgSequence;
```

```
570
571     } else {
572     // Error occurred in request to send.
573     }
574     // Wait for responds
575     FlipLed( LED_MSG_SEND );
576   }
577
578   // Update sequence number
579   msgSequence++;
580 }
581
582 /*********************************************************************************************
583 * Sends the buffered sequences
584 *********************************************************************************************/
585 void sendBuffered( uint8 length, uint8 *msg ){
586   // If none stored, discard.
587   if( dataSampleNrStored < 1 || length < 1 ){
588   return;
589   }
590
591   // indexes to request
592   uint8 bufFil = 0;
593   uint8 bufMax = END_D_MAX_BUFFER / 10;
594   uint8 bufferIndex[ END_D_MAX_BUFFER / 10 ];
595   if( msg == NULL ){
596   //Send all after dataSampleLastSent
597   for( uint8 i=0; i < dataSampleNrStored; i++){
598     if( storedSamples[i].msgSequence > dataSampleLastSent ){
599       bufferIndex[ bufFil ] = i;
600       bufFil++;
601       if( bufFil == bufMax ){
602         break;
603       }
604     }
605   }
606   }else{
607   // Prevent overflow
608   if( ( length / 2 ) > bufMax ){
609     length = bufMax * 2;
610   }
611
612   // Send the onces requested
613   for( uint8 i=0; i < dataSampleNrStored && bufFil < ( length / 2 ); i++){
614     for( uint8 j=0; j < length; j+=2){
615     uint16 requestedSeq = BUILD_UINT16( msg[j+1], msg[j] );
616     if( storedSamples[i].msgSequence == requestedSeq ){
617       bufferIndex[ bufFil ] = i;
618       bufFil++;
619       }
620     }
621   }
622   }
623
624   // Send all requested sequences that are known
625   for(uint8 index = 0; index < bufFil; index++){
626     dataSample *tempData = &storedSamples[ bufferIndex[ index ] ];
627
628     // Save into buffer large enough for header + buf
629     uint8 buffer[ END_DEVICE_HEADER_LEN + END_DEVICE_buflen ];
630     for( uint8 i = 0; i < END_DEVICE_buflen; i++ ){
631       buffer[ END_DEVICE_HEADER_LEN + i ] = tempData->buffer[ i ] ;
632     }
633
634     // Save Sequence in buffer
635     buffer[ SERIAL_TIMESTAMP_LEN + 0 ] = HI_UINT16( tempData->msgSequence );
636     buffer[ SERIAL_TIMESTAMP_LEN + 1 ] = LO_UINT16( tempData->msgSequence );
637
638     // Save number of samples. (STATIC 1)
639     buffer[ SERIAL_TIMESTAMP_LEN + SERIAL_SEQ_LEN + 0 ] = 0;
640     buffer[ SERIAL_TIMESTAMP_LEN + SERIAL_SEQ_LEN + 1 ] = 1;
641
```

```
642     // Timestamp data sent at latest point possible
643     uint32 sysTime32Send = osal_GetSystemClock(); // returns milliseconds
644     for( uint8 j=0; j < SERIAL_TIMESTAMP_LEN && j < 4; j++ ){
645         buffer[j] = 0;
646       buffer[j] = (uint8)(((sysTime32Send) >>((SERIAL_TIMESTAMP_LEN - j - 1) * 8)) & 0x00FF);
647     }
648
649     if ( AF_DataRequest( &GenericApp_Flash_DstAddr, &GenericApp_epDesc,
650               GENERICAPP_AD_ANSWER_CLUSTERID,
651               END_DEVICE_HEADER_LEN + END_DEVICE_buflen,
652               buffer,
653               &GenericApp_TransID,
654               AF_DISCV_ROUTE,
655             AF_DEFAULT_RADIUS ) == afStatus_SUCCESS )
656     {
657     // Successfully requested to be sent.
658     }
659     else
660     {
661     // Error occurred in request to send.
662     }
663   }
664
665   // LED indicator
666   FlipLed( LED_MSG_REQ );
667 }
668 #endif
669
670 /*********************************************************************************************
671 * DAWIJK
672 * prototype LED blink
673 *********************************************************************************************/
674 void FlipLed( unsigned int format){
675
676   bool oldDesign = true;
677
678   if( oldDesign ){
679     P1SEL &= 239;//1110 1111 //Port 1 pin 4 to GPIO
680     P1DIR |= 16; //0001 0000 //Port 1 pin 4 to output
681     P1_4 = 0;
682     for (uint8 a =0; a < 10; a++){
683     for(unsigned int i =0; i < format; i++ ){
684       for(unsigned int z =0; z < format; z++ ){
685       asm("NOP"); \
686       asm("NOP"); \
687       }
688     }
689     P1_4 = !P1_4;
690     }
691     P1_4 = 0;
692
693   }else{
694
695     P0SEL &= 253;//1111 1101 //Port 1 pin 4 to GPIO
696     P0DIR |= 2; //0000 0010 //Port 1 pin 4 to output
697     P0_1 = 0;
698     for (uint8 a =0; a< 10 ;a++){
699     for(unsigned int i =0; i < format; i++ ){
700       for(unsigned int z =0; z < format; z++ ){
701       asm("NOP"); \
702       asm("NOP"); \
703       }
704     }
705     P0_1 = !P0_1;
706     }
707     P0_1 = 0;
708   }
709 }
710
711 /*********************************************************************
712 *********************************************************************/
```

E

Server application

E.1 MyServerSocket.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Net.Sockets;
6  using System.IO;
7
8  namespace MyServerSocket {
9    class Program {
10   static void Main(string[] args) {
11     bool mainFlag = true;
12     String file = "c:\\data.txt";
13     if (!File.Exists(file)){
14       File.Create(file);
15     }
16     StreamWriter sw = File.AppendText(file);
17
18     /* Open port 9876 */
19     TcpListener serverSocket = new TcpListener(9876);
20     TcpClient clientSocket = default(TcpClient);
21     serverSocket.Start();
22     Console.WriteLine(" >> Server Started");
23
24     clientSocket = serverSocket.AcceptTcpClient();
25     Console.WriteLine(" >> Accept connection from client");
26     while ( mainFlag ) {
27       try {
28         if ( ! clientSocket.Connected ) {
29           clientSocket = serverSocket.AcceptTcpClient();
30           Console.WriteLine(" >> Accept connection from client");
31         }
32         NetworkStream networkStream = clientSocket.GetStream();
33
34         int avail = clientSocket.Available;
35         byte[] bytesFrom = new byte[avail];
36         if (avail > 0) {
37           /* Read all available bytes */
38           networkStream.Read(bytesFrom, 0, avail);
39           string dataFromClient = System.Text.Encoding.ASCII.GetString(bytesFrom);
40
41           Console.WriteLine(" >> Data from client - " + dataFromClient);
42           sw.Write(dataFromClient);
43           sw.Flush();
44         }
45       } catch ( Exception ex ) {
46         Console.WriteLine( "Exception" + ex.ToString() );
47         clientSocket.Close();
48         serverSocket.Stop();
49
50         mainFlag = false;
51       }
52     }
53
54     /* Notify user that application exits */
55     Console.WriteLine(" >> exit");
56     Console.ReadLine();
57     }
58   }
59 }
```