Accessing KNX Devices using USB/KNX Interfaces for Remote Monitoring and Storing Sensor Data

J. A. Nazabal[1], F. Falcone[1], C. Fernández-Valdivielso[1],
S. C. Mukhopadhyay[2] and I. R. Matias[1]

[1]*Electrical and Electronic Engineering Department, Public University of Navarra, Campus de Arrosadia 31006, Pamplona, Spain*
[2]*School of Engineering and Advanced Technology, Massey University, Palmerston North, New Zealand*
juanantonio.nazabal@unavarra.es

Abstract

Nowadays is very common for homes or buildings in general to have some kind of automation system for the efficient use of energy and for common comfort matters like illumination and blinds & shutters controlling, among others. It can easily add to these systems different kind of sensors for different uses like: monitoring building structure health, monitoring elder people behavior, etc… In this work we have developed a system that access data from connected KNX sensor devices and sends it remotely to a MySQL server using IP packets.

Keywords: *KNX, USB, Smart Homes, Home Automation, Sensor Monitoring, MySQL*

1. Introduction

Nowadays the presence of home automation and environmental control systems in homes and in public buildings has been increased. These systems are capable of automating a home through energy management, safety, welfare and communication allowing a more efficient use of energy available and therefore contribute greatly to the sustainable development of our society. There are several different home automation technologies around the world for choosing, with their own advantages and disadvantages. In this work we have used KNX technology [1-3]. It is a worldwide standard for home and building control with different transmission mediums. KNX devices are provided by a large number of manufacturers, offering more functionality within the home automation system and the standardization guarantees that the different manufacturer ́s products may be connected together. This means there are no limitations to a single manufacturer and if a manufacturer ceases to trade or offer a particular product, the same or similar products are available from other manufacturers.

There are several ways for accessing KNX bus data, like: serial, Universal Serial Bus (USB) [4], Internet Protocol (IP) interfaces, *etc*….

In this work we are going to develop a Java application for accessing KNX sensor data via a USB/KNX interface [5] and store it remotely in a MySQL [6] server. For testing the application, a small KNX network has been installed consisting of a USB / KNX Interface and a 4 analog input module with various generic sensors.

2. Description

The application that has been developed in this project for accessing KNX sensor data has been programmed in Java and implements the KNX standard Application Note 037/02 Rev. 4 [7].

The USB / KNX interfaces are constructed following the KNX standard. They use Human Interface Device (HID) [8][9] Class for exchanging KNX data frames. The HID Class consists primarily of devices that are used by humans to control the operation of computer systems. The KNX Association does not require USB certification for KNX USB Interface devices working together with the KNX system tools. Nevertheless, such devices shall fulfill the USB specification version 1.1.

The data is transferred in packets called Reports limited to a maximum length of 64 octets. The corresponding frame structure is show in Figure 1. The Report ID field value for KNX data exchange is always fixed to '01h'.

KNX HID Report Header				**KNX HID Report Body**
ReportID	PacketInfo		Datalength	Data
	Sequence Number	Packet Type		
1 Octet	1 Octet		1 Octet	Maximum 61 Octets

Figure 1. HID Report Frame Structure

The Report ID allows the HID Class host driver to distinguish between different incoming data by examining this transfer prefix. The sequence number field is reserved for future use and the packet type field indicates if the packet is start & end packet, partial packet, start & partial packet or partial & end packet.

The data field (KNX HID Report Body) consists of the KNX USB Transfer Header and the KNX USB Transfer Body. An example for an "L_Data_Request" in Common EMI (cEMI) format is shown in Figure 2.

KNX HID Report Body							
KNX USB Transfer Protocol Header						**KNX USB Transfer Protocol Body**	
Protocol Version	Header Length	Body Length	Protocol ID	EMI ID	Manufacturer Code	EMI Message Code	Data (cEMI/EMI1/EMI2)
1 Octet	1 Octet	2 Octet	1 Octet	1 Octet	2 Octet	1 Octet	Max. 52 Octets

Figure 2. KNX HID Report Body Structure

The protocol version information shall state the revision of the KNX USB Transfer Protocol used in the frame. The Header Length shall be the number of octets of the KNX USB Transfer Protocol Header. The only valid protocol version value nowadays is '0' and the corresponding header length size, 8 bytes.

The Body Length shall be the number of octets of the KNX USB Transfer Protocol Body. Because an interface device connecting a PC with a field bus via an USB link can not only transfer KNX frames but also other protocols, the field Protocol ID in the header shall be used as the main protocol separator. For using KNX tunneling, the value of this field is 01h and for 0Fh Bus Access Server Feature protocol, 0Fh.

For a KNX Tunnel, the External Message Interface (EMI) ID field shall represent the EMI format used in the KNX USB Transfer Protocol Body. An EMI ID value of 01h indicates that the EMI format used is EMI1, 02h is used for EMI2 and finally 03h for cEMI format.

In protocol version '0', Protocol ID shall always be present. Value '0000h' shall be used for transmission of frames that fully comply with the standardized field bus protocol, indicated with Protocol ID octet. In case of a KNX Link Layer Tunnel, this field shall be set to '0000h'.

If not fully complying with the standard indicated in the Protocol ID field, then the manufacturer code field of the KNX USB Transfer Protocol Header shall filled in with the manufacturer's KNX member ID.

When using KNX tunneling, KNX frames are tunneled on the USB link using one of the EMI formats. The time-out for a KNX tunneling is 1 s. In that time interval, the KNX USB Interface Device shall be able to receive a tunneling frame, transmit it on the KNX medium and send the local confirmation back.

For accessing USB device features Bus Access Server Feature protocol is used and the corresponding frame format is shown in Figure 3.

KNX HID Report Header			**KNX HID Report Body**						
Report Identifier	Packet Info	Packet Length	USB KNX Transfer Protocol Header						KNX HID Transfer Protocol Body
			Protocol Version	Header Length	Body Length	Protocol Identifier	Service Identifier	Manufacturer Code	Feature Identifier
1 Octet	1 Octet	1 Octet	1 Octet	1 Octet	2 Octet	1 Octet	1 Octet	2 Octet	1 Octet

Figure 3. Device Feature Service Frame (Example)

Depending on the Service Identifier field value, the frame can be a device feature query (01h), a device feature query's response (02h), a device feature set (03h) or info about a device feature (04h).

There are many device features but the most useful ones and that have been implemented in this work are shown in Table 1.

For exchanging KNX HID Report frames the java application JAVAHIDAPI [10] library has been used.

This is JNI wrapper around C/C++ HIDAPI library which allows an application to interface with USB and Bluetooth HID-Class devices on Windows, Linux, and Mac OS X. As this is a wrapper of the C++ library, the installation of Microsoft .Net Framework is needed.

Table 1. Device Features Overview

Feature Identifier	Feature Name	Description	Data Length
01h	Supported EMI type	Getting the supported EMI type(s)	2 octets (B_{16})
03h	Bus connection status	Getting and informing on the bus connection status	1 bit (B_1)
05h	Active EMI type	Getting and Setting the EMI type to use.	1 octet (N_8)

The first step is to list all the HID devices connected to the system using the "HIDManager.getInstance().listDevices()" method. From that list, we need to select the USB /KNX interface device that we want to use and open it using "HIDManager.getInstance().openByPath" method. Once the device is correctly opened we can start reading and receiving HID reports.

Every time a sensor sends new data through the KNX bus, we will receive a new KNX HID report. Once we have the report, we need to parse it for extracting EMI [11] frame (EMI1 / EMI2 / cEMI). Finally, when we have the EMI frame and depending of the data type presented, we need to decode it for getting the final sensor data. Sometimes the KNX devices are programmed for sensing data only on demand. In that case, an EMI frame asking for data must be encapsulated into a KNX HID report and sent to the sensor device.

When the KNX HID report has been totally parsed and the corresponding sensor data obtained, it is time of processing it. We can simply send it via UDP datagrams or TCP segments over IP packets for processing elsewhere or store them remotely in a data server for further analysis. The second option is the one that we are going to use, storing the data into a MySQL server.

For accessing MySQL in Java MySQL JDBC driver has been used. First we need to create a new instance of the driver using "Class.forName("com.mysql.jdbc.Driver").newInstance()" method and then connect to the server using "DriverManager.getConnection". Once we are connected, a new statement must be created using "createStatement()" method for executing MySQL queries.

Sensor data is stored in two different tables. The first table, named "sensorconfig", stores the sensor identifier, name, description, units and the conversion factor for converting registered voltage value into real data. The second table, named "sensordata", stores sensor data. Each time a new data frame is received, the sensor's identifier (same as in table "sensorconfig"), time of arrival and data value are stored in a new entry.

3. Results

For testing the developed Java application, a small KNX network has been installed. It consists of a USB / KNX interface model 2130 USB REG by Jung and a 4 analog input device model 2214 REG A by Jung. It has 4 analog input channels and is highly configurable, an important feature that makes sensor connection easier. The analog input can be evaluated using voltage signals (0 ... 1 V DC or 0 ... 10 V DC) or current signals (0 ... 20 mA DC or 4 ... 20 mA DC). Every input channel can independently be configured for sending data periodically or after a change is detected. For detecting these changes, every channel has two different threshold levels with hysteresis, for avoiding oscillations when the measured value

is near the threshold value. Figure 4 shows the connection diagram of all the devices implicated in the testing network.

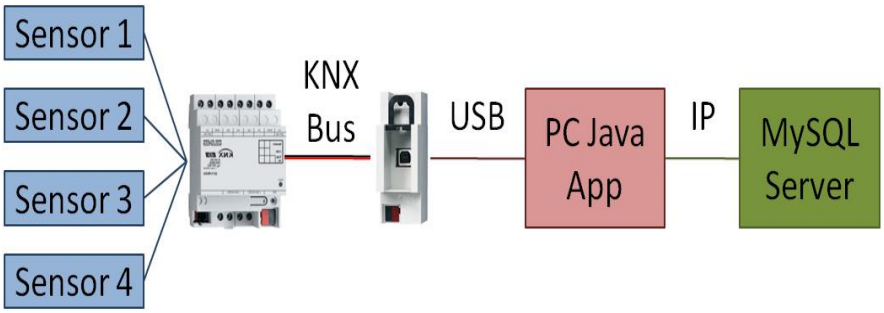

Figure 4. Test Network Connection Diagram

The USB / KNX Interface device is connected to the KNX bus and to a personal computer USB port. This computer has an Ethernet connector for accessing internet and a windows XP system, with JRE and Microsoft .Net Framework installed. The Java application developed in this work runs in this computer and access a remote MySQL server using IP protocol.

Figure 5 shows some MySQL server and Java application exchanged packets captured with the Wireshark [12] network protocol analyzer tool. In the screenshot it can be appreciated the IP addresses of the Control Unit (172.18.70.6) and the MySQL server (172.18.70.45). It also shows that 3306 port and MySQL protocol are used. Finally, if focusing on the right-bottom section of the figure, the content of one of the MySQL insert query entry can be seen.

Figure 6 shows a screen capture of the MySQL stored data. On the left side, a screenshot of the "sensorconfig" table is shown. It can be appreciated how the system consist in a temperature sensor, a magnetic proximity sensor for determining if a door is closed or not (0 VDC: Closed, 5 VDC: Opened) and a Boolean sensor with indicates door's status (0: Closed, 1: Opened) which value is stored on demand or when the door status changes.

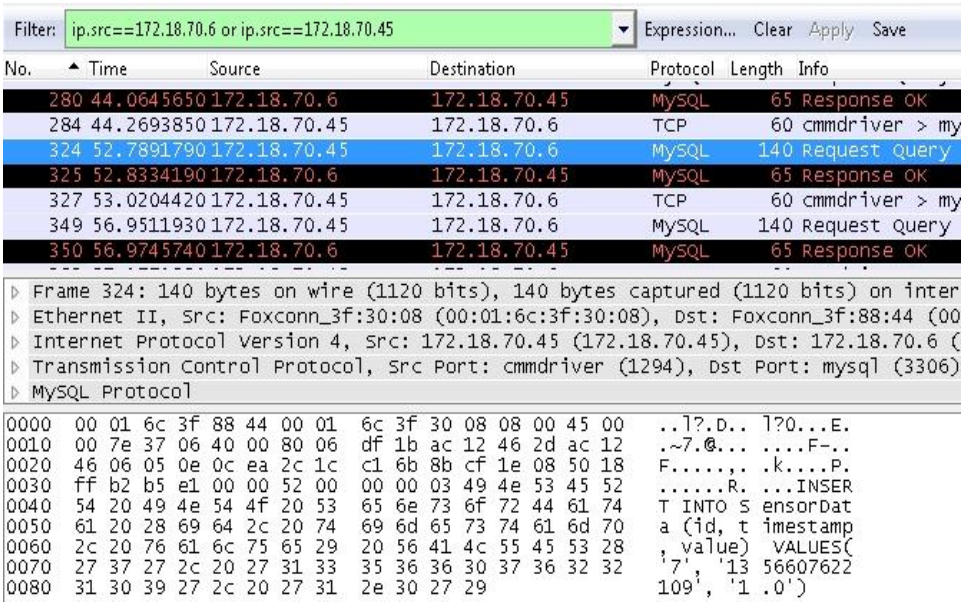

Figure 5. Wireshark IP Traffic Capture

On the right side of the figure, a screenshot of a portion of the "sensordata" table is shown. The timestamp field stores the sensor frame time of arrival, represented by the number of milliseconds elapsed between the arrival time and January 1, 1970 00:00:00.000 GMT (Gregorian).

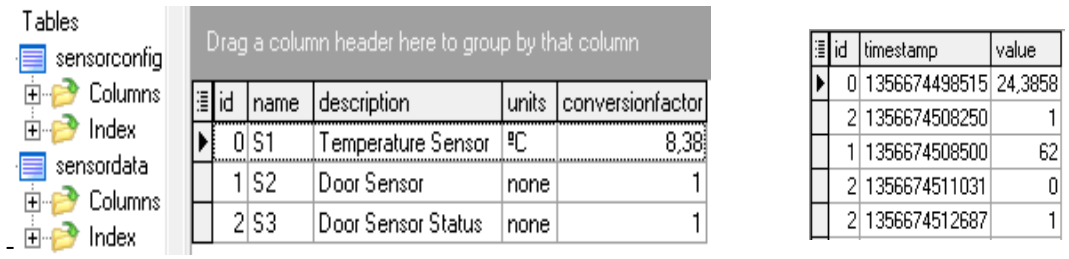

Figure 6. Screenshot of the MySQL Stored Data

4. Conclusions

In this work a KNX sensor remote monitoring system has been developed using a USB / KNX Interface. A temperature and a magnetic proximity sensor have been used as an example but they can be exchanged by any kind of analog sensor. Once the sensor data has been acquired, in this work it has been stored in a remote MySQL server but it could be sent to a remote host via UDP or TCP protocol for remote monitoring and processing.

It is important to mention that sensor data is exchanged in plaintext, with no encryption. This must be a problem because the data travels around the Internet and can be potentially accessed by unwanted recipients. If the security of the exchanged data is critical, an extra security mechanism as some kind of encryption must be provided.

The main drawback of this system is that a KNX USB interface device only supports one connection at the same time.

Acknowledgements

This work was supported by the Spanish Economy and Competitivity Ministry AIB2010NZ-00328.

References

[1] Cenelec EN50090, "Home and Building Electronic System (HBES)", **(2005)**.
[2] CEN EN 13321-1, "Open Data Communication in Building Automation, Controls and Building Management", HBES, **(2006)**.
[3] ISO/IEC 1454-3, "Home Electronic System (HSE) Architecture", **(2006)**.
[4] Universal Serial Bus Specification Revision 1.1, Compaq Computer Corporation, Intel Corporation, Microsoft Corporation, NEC Corporation, **(1998)** September 23.
[5] H. W. Werntges, J. Neumann and V. Vinarski, "Controlling EIB/KNX devices from Linux using USB", Konnex Scientific Conference, ISTI Institute Pisa, Italy, **(2005)**.
[6] MySQL, http://www.mysql.com/.
[7] "KNX on USB Protocol Specification & KNX USB Interface Device Requirements", Application Note 037/02 Rev. 4, **(2003)**.
[8] Device Class Definition for Human Interface Devices, V 1.11, http://www.usb.org/developers/devclass_docs/HID1_11.pdf.
[9] HID Usage Tables, V 1.11, http://www.usb.org/developers/devclass_docs/HID1_11.pdf.
[10] Javahidapi, http://code.google.com/p/javahidapi/.
[11] Cenelec EN 50090, "External Message Interface", Chapter 3/6/3, **(2009)**.
[12] Wireshark, http://www.wireshark.org/.