LEIBnix Reference Manual

Version 01
This document is related to LEIBnix v1.019

General

The EIB (european installation bus) bus system is well suited for building automation, such as controlling light and temperature, and collecting sensor signals. But also control applications, like burglar alarms and many comfort functions can be realized - but users are restricted to use commercial available devices, these are often expensive and not flexible.

The project LEIBnix aims to provide an open solution for controlling EIB devices using an embedded Linux platform - the UNC20 controller. But of course, the LEIBnix software application can easily be used on any Linux distribution and requires only a few changes to adopt to other hardware.

LEIBnix Hardware Features

The UNC20 was chosen because it provides a cost-efficient, small and powerful plattform. A home controller shall be reliable all time, consume only a minimum of power, avoid head dissipation, so it requires no fan or cooling elements and emits no noise. The module comes with 10 MB ethernet interface, ideal for configuring, monitoring, debugging and much more. For more details see www.fsforth.de

Our UNC20 comes with 16 MB memory, way enough to store all temporary data, configuration and log files. We use one serial port to connect to a BIM113-2.1 module which is gateway to the EIB bus. Since the UNC20 operates on 3.3V and for safety reasons, this serial interface uses opto-couplers.

We want our home controller to do things based on date and time, so we added a PCF8563 chip to the I^2C bus with a gold cap capacitor.

I also connected a 20x4 LCD display to the UNC BasBoard connector to display some informations. You can use almost any display compatible to the Hitachi 44780 controller command set.

LEIBnix Software Features

This list is - of course - subject to changes

- a configuration file is used to register all EIB messages
- EIB messages can be sent using a time schedule (based on exact time, sunrise or sunset)
- simple rules can be set: if (msg1) wait (x seconds) then send (msg2) - this can be used to control blindfolds
- some EIB messages can be registered for monitoring, using the UNC web server
- use any web browser to watch the schedule and current status, or control some functions
- LEIBnix provides an EIB proxy server - a display or visualization tool can use this data to display the current building status
- a FIFO/pipe is provided to allow a cgi script or other processes to send EIB messages
- the signal SIGUSR1 is used to force a reload of the user config file

... and many more ideas ...

Software Reference Manual

Although I prefer to use C++ for encapsulation wherever I can, there is still some C code. I try to keep the code somehow well structured and use some coding conventions of which I think they make sense: like using type prefixes etc. - but nothing is perfect and you'll forget to fix things when they work.

The development environment that comes with the UNC is excellent - you need only a little knowledge about Linux to build your UNC applications, just follow the documention.

The main module is LEIBNIX.CPP, each module compiles to an object file, all are linked into one application LEIBnix.

LEIBnix Main Module

The main module consists of an initialization part and a loop with 10 ms cycle time using usleep() command - the time is not critical but shall be close to 10 ms because some othe functions rely on this.

The application can be terminated by pressing a button on the UNC BASBOARD, but I usually simply kill the process without noticing any side effects so far.

Initialization

- initialize the LCD display
- read config file
- update HTML files for web server
- register the signal handler for SIGUSR1
- check/create the FIFO
- establish connection to the EIB via the BCU
- start any static plug-ins
- start proxy server

Establish connection to EIB

We are using a BIM113 to communicate with the EIB - so it should behave like a transparent gateway, giving us all (unconnected) messages from the bus and allow us to send messages. It is currently not possible to listen to 'connected' communication, that is used when a device is programmed.

A 'type' resistor is used to put the BIM into a mode for serial communication, this uses the FT-1.2 protocol to encapsulate the EIB messages.

"Talking EIB", a message is called "group address" but compared to other bus systems it is simply some sort of dataframe with source and destination information and some data bytes.

The function open_ft12() opens the serial port for 19200-EVEN-8-1 communication.

Next step is to establish a signal handler for the serial port. I was looking for some asynchronous method to process incoming serial data and this turned out well. Whenever serial data is available, the system calls to receiver_FT12() and we can unwrap the FT-1.2 messages.

Finally we need to initialize the BIM/BCU by sending three commands:

- send a RESET command to the BCU
- send a PEI_SWITCH command, allowing us to directly transmit EIB messages through the BCU
- set length of address table to 0, so the BCU accepts all incoming group addresses

when the BCU acknowledges all three steps with a ACK (=0x5E) the BCU is ready.

Read Config File

I'm using a regular text file for configuration - I was thinking about using XML but the overhead seemed to big, but this may come in future.

Example of Config File

The default name of the config file is EIBCONFIG.INI and looks like this:

```
[Header]
version = 3.00

[GroupAddrs]
10/0/0   = Room1 set Dimmer       |+|*|off|on|
10/0/2   = Room1 status Dimmer    |+|*|=off|=on|
10/1/0   = Ventilation            |-|*|off|on|
2/2/4    = Main Door              |+|*|=closed|=open|
2/2/5    = Main Door              |+|*|unlocked|locked|

[System]
SYSTEMENABLE = 8/0/0
SYSTEMSTATE  = 8/0/1

[Schedule]
SETTIME |  05:00:00 | 31/7/255 | 1 | MDMDFSS | reload config
SETTIME |  01:30:00 | 10/1/0   | 1 | MDMDFSS | ventilation on
SETTIME |  04:00:00 | 10/1/0   | 0 | MDMDFSS | ventilation off
SUNDOWN | +01:30:01 |  3/2/9   | 1 | MDMDFSS | blindfolds down

[Rules]
10/7/1 = 0 | 12 | 10/7/2 = 0 | stop blindfold after 12 secs

[StatusObjs]
2/2/4   |grn|red| main door open-close
2/2/5   |red|grn| main door is locked
2/1/1   |grn|ylo| windows 1st floor
```

The function ReadConfig() opens the config file and reads it line by line. According to the section header, each line is parsed using a sscanf() call and the information is the stored into tables. I decided to use static tables because the UNC has no MMU and I assume that the config file's size does not change too much.

Most entries use a '=' or ' |' character to separate parameter fields, see examples above.

[Header]

this section is currently not used, the version tag is just for convenience

[GroupAddrs]

The purpose of this section is to register all group addresses. Format of the entries is like this

1. group address
   EIB group address in the form 'a/b/c'

2. description
   description of the group address as text

3. logging flag
   selects logging for incoming EIB messages

'+' enables logging
'-' disables logging

4. data type
   in some cases it is necessary to know the data type for further processing - this list is not yet complete

'*'	is the default value and interpretes data as 1 bit
'BIT'	selects an 1 bit data type
'B1'	selects an 1 byte data type
'B2F'	selects a 2 byte float data type (=EIS5)

5. off-state-text
   text description in case the data value equals 0 or FALSE

6. on-state-text
   text description in case the data value equals 1 or TRUE

[Schedule]

This section is like a time table to control EIB devices.

1. time-base

'SETTIME'	treats the following time as exact time
'SUNRISE'	the following time is a shift to the sunrise time
'SUNDOWN'	the following time is a shift to the sunset time

1. time
   Any time in the format hh:mm:ss (hh is 00..24)
   In case SUNRISE or SUNSET is selected, a leading + or - sign indicates the shift.

3. group address
   EIB group address in the form 'a/b/c'

4. value
   data value attached to the message

5. weekday schedule
   Selects individual weekdays, requires 7 characters starting on monday thru sunday. Use a '-' character to disable this event for a specific weekday. The software looks only for the '-' so you can use your language format for day abbreviation:

Examples	MDMDFSS	all days	(german)
	MTWTFSS	all days	(english)
	LMMJVSD	all days	(french)
	-----SS	weekend schedule	

6. comment
   can be any comment, helping you to remember this event

REMARKS
There is (so far) one special group address (31/7/255 = 0xFFFF) that is used to reload the config file once a day. The idea behind the schedule is to make a plan for one day and simply work through the list, so we need to recalculate the schedule once a day to apply users changes and also to make the sunrise/sunset work. A good idea to do this is in the early morning hours.

[rules]

This section is designed for logic operations, at this time it is fairly simple. It allows the user to specify a rule like this:

when (groupaddr && value) then wait (seconds) then send (groupaddrs / value)

This is useful when you want to control your blindfolds to move down for just 10 seconds when you press the button - but I'm sure there are more ideas.

[statusobjs]

The idea behind the StatusObjs is monitoring - a way to create a web document allowing me to have a quick overview of the house, if everything is green (or yellow) I can go to bed. The web document is accessed by any browser.

1. group address

   EIB group address in the form 'a/b/c'

2. off-state-color

   color code in case the data value is 0 or FALSE
   values are GRN, YLO, RED and BLU

3. on-state-color

   color code in case the data value is 1 or TRUE
   values are GRN, YLO, RED and BLU

4. comment

   can be any comment, helping you to remember this event

REMARKS

I had some trouble with the UNC's web server because I wanted to use CSS but this creates some problems, so there is some work left to do.

[system]

This section provides some system specific controls.

SYSTEMENABLE

Any other EIB device (eg. a button or display) can send this message to toggle the online/offline state of LEIBnix. 'Offline' means, that LEIBnix is not processing any schedules and rules.

SYSTEMSTATE

LEIBnix sends this message to indicate whether it is online or offline, one can use this message as status feedback (eg. to display LEIBnix' current status on an EIB display unit).

Software Description

Data Tables

LEIBnix uses some internal tables to store data

- 'event' stores the list of scheduled events. The entries are usually read from the config file, these are static events. In case a rule is activated, a dynamic event is created. The main loop checks cyclic the list of events and whenever an event is done, it is marked as done.
- 'text' stores the list of group addresses, this information is used for the HTML documents etc.
- 'rule' stores the list of rules - whenever a group message is received, the system checks for a matching rule and handles the message.
- 'status' stores the list of status objects - these group addresses are monitored and written to a HTML document. When LEIBnix is started, these group addresses are requested by sending ObjValueRead messages on the bus. To avoid overloading the bus, each request is sent after 1 sec pause.

HTML documents

LEIBnix creates two HTML documents (by default they are placed into /ram).

The file EIBSTATUS.HTM contains general informations and the current status of the event table.

The file EIBOBJS.HTM contains the table of status objects **.**

Plug-Ins

I started to develop some of my own EIB devices, based on BIM modules, so I wanted to add some special functions to LEIBnix to support those devices. What I really like is a way to dynamically load a module, a little like a DLL works - but at first, I started with a static plug-in.

Any input on this idea is appreciated.

The test16 plug-in is used to request a data value from my temperature module every 30 seconds, convert the value (16 bit float) into a EIS5 value and send it over the bus, so a display module can inform about the temperature.

EIB message flow

The idea behind LEIBnix is, that most people want to control their EIB based on messages that are received or transmitted.

As described above, the BCU/BIM is programmed to receive all group addresses that appear on the bus, then the message is received by the serial interface. This activates the signal handler, reading the buffer content and decoding the FT 1.2 message format. When a complete and valid message (=group address) is received, the LEIBnix acknowledges this to the BCU and starts to dispatch the message.

Dispatching includes these actions

- if the logging flag is set, a log entry is added to the log file
- if a matching rule exists, a dynamic event is scheduled
- any plug-in is called to process the message
- if a matching status obj exists, the data table and and state flag is updated

IMPORTANT : incoming messages are processed in the process context of the signal handler and you must not transmit any EIB message out of this context. The preferred way to send a message based on any incoming message is to set a flag and let the main loop check the flag and then transmit a message. I cannot explain this in detail - but it seems to make sense. In any other case, the system crashed (at least the UNC did).

As explained above, outgoing messages are sent from the main loop. I added some helper routined to make message transmission easier:

- SendEIBmsg() send a group address and 1 bit data value
- ObjValueWrite() is similar, but allows to select a data type
- ObjValueRead() sends a read request for a group address to the bus
- SendFTmsg() encapsulates a EIB message into a FT 1.2 frame and sends it to the BCU

SunCalc

Trying to control anything based on sunrise or sunset is usually solved by some EIB light sensor - a solution I liked much better is a calculation based on longitude and latitude.

The class CSunRiseSet encapsulates all computation routines and delivers only the necessary informations to the LEIBnix application

- ::SetDSTzone() sets the difference to GMT (germany is +1)
- ::GetDayOfYear() gets the julian day of year index (1..366)
- ::GetSundown() gets the sundown time based on location and day index
- ::GetSunrise() gets the sunrise time based on location and day index
- ::GetBeginDST() gets begin of daylight saving time (last sunday in march)
- ::GetEndDST() gets end of daylight saving time (last sunday in october)

EIBPIPE

I was looking for a way to have a button on the LEIBnix' web page that allows me to switch devices on or off. A solution is to put a cgi application behind the button and connect cgi and LEIBnix using a fifo, this is /ram/EIBPIPE.

This job is not completed but works nice so far - I'm going to clean up the command format some day.

I recently added this enOcean receiver to the LEIBnix and it turned out, that it works great to use the EIBPIPE for this interface too.

These commands are currently supported

- SetGA..... transmits a group address (16 bit hex) and value , example "SetGA5600v1"
- ReConfig reload config file
- SystemEnable. set LEIBnix online/offline , example "SystemEnable0" or "SystemEnable1"

EIB frame decoding

I use predefined EIB message frames and simply change the appropriate fields - so how does this work ? The format of an EIB message is described in many documents (so I'n not going to repeat this here in all details) - a very helpful document is the 'BCU2 helpfile' with a table describing all EIB frame formats.

FT1.2 issues

Here is some background information about the FT 1.2 protocol.

First, the serial interface needs to be configured for 19200 Baud / EVEN Parity / 8 Bits / 1 Stopbit

Every communication across FT 1.2 needs to be acknowledged using ACK = 0x5E character. The EIB specification defines a timeout of 36 bit times = 1.875 ms. This is usually no problem for any kernel driver or embedded device - but a user level application may guarantee this.

When a BCU send a message and this is not acknowledged, the BCU repeates the message up to 3 times, and the 'repeat flag' in the EIB Control Field is set to '1'. So whenever you see every message 4 times, your acknowledge does not work.

Each FT 1.2 message consists of a 4 byte header : 0x68 - length - length - 0x68

The next byte contains the FT 1.2 Control Field. When sending data to the BCU, initialize this field with 0x53 and toggle the Frame Counter Bit (bit 5) on each message.

The FT 1.2 checksum field is simply the byte sum of the FT 1.2 control field and all message bytes. the message is terminated with a 0x16 character.

What's next ?

Allthough my LEIBnix is running stable since 2004, there is some work to do, cleaning the code and adding new features.

As far as it concers my version of LEIBnix I'm going to stay with the UNC20 platform - meaning that the code shall be small and lean - this may be different for a PC-based LEIBnix.