Lantronix AVL FIRMWARE RELEASE

VERSION: avl_3.5.0_rc8

BIOS version: 3.0.4

Official release date: 02.05.2019 (M/D/Y)

List of firmware files: avl_3.5.0_rc8_20200204.frp
avl_3.5.0_rc8-Z0d67e2eb.zip
avl_3.5.0_rc8_20200204.txt

Hardware compatibility: This firmware applies to the following LANTRONIX products with Cortex processor:

Devices	Hardware Revisions	Supported firmware versions	Notes
FOX3-2G Series	13,15,17,19,20,21	avl_3.x.x (only)	1) Use the PFAL command `$PFAL,MSG.Version.HardwareRev` to get shown the hardware revision of your AVL device. The device responses with (second line shows the hardware version): $<MSG.Version.HardwareRev> $11-NUCHB $SUCCESS 2) The hardware revision is also printed on the product label, located on the back panel of the device. In the Serial Number (S/N) field there are 3 digits in parenthesis, for example, 60148(9XX)50600014, and the number "XX" is the hardware revision of the device. If the number is "11", it means that the hardware revision is 11.
FOX3-3G Series	06,11,13,15,17,19,20,21	avl_3.x.x (only)	
FOX3-3G-BID*			
FOX3-4G Series	All	avl_3.x.x (only)	
BOLERO40 Series	All	avl_3.x.x (only)	

* On request

IMPORTANT

- This firmware version is ONLY for the LANTRONIX products explicitly Mentioned above! Do not try to update other LANTRONIX products with this firmware, otherwise, you will not be able to operate your device anymore.
- Before updating the new firmware on your FOX3 or Bolero series, it is strongly recommended to back up the configuration with the command **$PFAL,CNF.Backup**
- Before upgrading the firmware on your FOX3 or Bolero series, it is recommended to upload and back up all history data on your server (if needed) and finally delete this data on the device.

NOTE

- If FOX3-3G-BLE devices with older firmware versions (e.g. 3.0.0_xx) are upgraded to this new firmware version (3.3.0_xx), please contact LANTRONIX to receive the BLE activation codes and continue to use this feature without additional costs.
- The latest FW 3.5.0_rc8 is for FOX3-2G/3G/4G with the CORTEX CPU as well as BOLERO40 series.
- Sleep=Ring on BOLERO40 series works only by using SIM-SLOT2 (the upper SLOT) on BOLERO40 series

DOCUMENTATION:

The following document(s) is (are) provided on https://www.lantronix.com// as part of the AVL firmware release "avl_3.5.0_rc8".

Filename	Description
PFAL Command Reference	Lists and describes all PFAL commands supported by this firmware release.

Version	Description	Created by	Date (M/D/Y)
3.5.0.0	Firmware release “avl_3.5.0_rc8”	Lantronix	02/05/2020
3.4.0.0	Firmware release “avl_3.4.0_rc8”	Lantronix	12/03/2019
3.3.0.0	Firmware release “avl_3.3.0_rc15”	Lantronix	10/02/2019
3.2.0.3	Firmware release "avl_3.2.0_rc39"	FALCOM	07/04/2019
3.1.0.2	Firmware release "avl_3.1.0_rc33"	FALCOM	11/09/2018
3.1.0.1	Firmware release "avl_3.1.0_rc20"	FALCOM	05/15/2018

1) Preface

This release note describes the new functionalities of the firmware release "***avl_3.5.0_rc8***" and is intended for use as a reference when updating an AVL device to version "***avl_3.5.0_rc8***".

2) Important Notes

The firmware file with extension "***.frp**" is for the update through the **Workbench** and for the update remotely OTA (RUpdate). The firmware file with extension "***.txt**" is for the update through **terminal emulators** (e.g.: Hyperterminal, PComm Pro). The firmware file with extension "***.zip**" is for the WebUpdate. To update the firmware with the extension "***.frp**", please use the **Workbench** version **2.6.2_RC7** or higher. To update the firmware with the extension "***.txt**" you can use any **terminal emulator** (example: Hyper terminal, Pcomm Pro). To initiate a WebUpdate use the command **$PFAL,SYS.WebUpdate.Start,"u rl”,80** on the device. DON'T switch off the AVL device while it reboots after the firmwa re update. The duration of the reboot after the firmware update may take approx. 45 seconds.

3) Firmware Installation Notes

The installation package consists of firmware in three different formats *.frp and *.zip. and *.txt. You can choose whether you want to update the firmware via following interfaces:

Interfaces	File	Description	References
RS-232 PORT	*.frp	This is primarily intended for updating one device first, to ensure the process completes properly before rolling the update to a group of other devices. Use " **Workbench** " and update the "*.frp"-file via the serial port.	
WEB-SERVER	*.zip	This is a perfect solution when multiple deployed AVL devices need updating. The firmware file is located in your web-server and you send to the AVL device the URL of a web server you have set up for downloading over-the-air the firmware file.	
Remote with Workbench	*.frp	This solution lets you update the firmware remotely on several AVL devices. More details can be found in the online help in the Workbench software.	
TCP-SERVER	*.frp	This solution lets you update the firmware remotely on several AVL devices.	
Terminal SW	*.txt	You can upload the firmware with the extension *.txt serially over a terminal SW such as PComm Lite, Tera Term, etc.	

4) Prerequisites concerning the PC

A 32/64-bit-WINDOWS operating system (Windows XP, Vista, 7) or Linux is running on your PC and about 50 MByte free space on your hard disk is required. The RS-232 interface must be configured with the following parameters:

- Baud rate: 115200
- Data Bits: 8
- Parity: None
- Stopbits: 1
- Flow Control: None

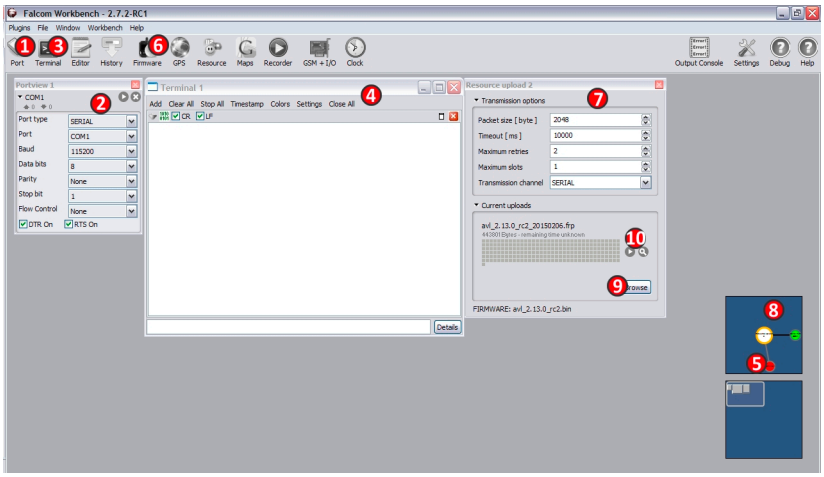
5) Firmware Update Process

These instructions are specific to updating your LANTRONIX AVL device via COM interface (Serial Port).

(a) Download the firmware file and Workbench software needed from the following hyperlinks.

1. https://www.lantronix.com/products/workbench/#tab-docs-downloads
2. https://www.lantronix.com/products/fox3-series/#tab-docs-downloads
3. Download "**avl_3.5.0_rc8.zip**" and extract the file you downloaded into a temporary folder on your PC.
4. Run the "**workbench**" software. If this software is still not installed on your PC, download it first

and start the installation.



(b) Begin the firmware update process (refer to the fig. above).

1. Connect the AVL device to your PC either directly using the programming cable or the corresponding evaluation board.
2. Do **NOT** update the firmware version 3.x.x on FOX3-2G/3G/4G devices with an older processor. The firmware version 3.x.x is **ONLY** for FOX3-2G/3G/4G and BOLERO40 devices with the **CORTEX (CT)** processor. Please verify the hardware revision from the table "**Hardware compatibility"** above and make sure you are upgrading a FOX3-2G/3G/4G device with CORTEX processor. LANTRONIX takes no liability and no responsibility for any cases, firmware versions have been flashed wrongly nor will LANTRONIX cover any costs associated with this happening.
3. Click **Port** (1) icon, select the COM port settings from the **PortView1** (2) and click the **Play** button next to the text "COM.." to open the selected COM port.
4. Click **Terminal** (3) icon, select the **TerminalView** 1 (4) and go to the **ConnectionView** (5) and connect it to the **Serial Port COM1**.
5. Click **Firmware** (6) icon, select "SERIAL" from the **Transmission Options** (7), go to **ConnectionView** (8) and connect it to the **Serial Port COM1**.
6. Click **Browse** (9) button and select the firmware file as "*.frp" from the temporary folder where the firmware was expanded.
7. Click **Play** (10) button to start the firmware update. This button appears only if the firmware file has already been selected.
8. Wait until the update process completes. While the update is running, do not send any command to the device and do not manually reboot it until the device restarts itself.
9. After the update process successfully completes, a success message will appear. Click "OK" button to restart the AVL device.
10. After device restarts and configuring the unit, you can execute the command **$PFAL,Cnf.Backup** to save the user configuration as factory settings. If the AVL device was

already configured, you can execute the same command after the firmware update to save the user configuration as factory settings.

11. LANTRONIX recommends that you update one device first, to ensure the process completes properly before rolling the update to a group of other devices.

6) New and Modified Functions

NEW FEATURES:

This is a list of new features since the latest released firmware.

- ✓ Added new PFAL commands/configuration/event/
- ✓ Added new functions

Key Features:

- ✓ **TCP**: 2nd TCP channel with the same performance of the primary TCP channel
- ✓ **Whitelist**: activate/deactivate a section of the whitelist
- ✓ **CAN variables**: Increasing the number of variables to 150
- ✓ **Configuration Backup**: Check valid BACKUP marker after device startup
- ✓ **LUA**: Extension in LUA functions (saving more than 1 LUA script for different profiles)
- ✓ **Security**: TLS 1.2 Support
  - The TLS feature is only available if the premium setting AES is activated
  - Forbid secure connection without AES_TCP feature set
- ✓ **CAN**: CANOpen Support
- ✓ **GSM**: Stability fixes for Fox3-4G-CATM1
- ✓ **Tracking**: CellLocate Feature
- ✓ **LUA:** Implement LUA runtime statistics
- ✓ **FMS:** Driver-ID can be readout completely with simultaneous multiple source addresses.

PFAL Commands: $PFAL,<Command>	
<Command>	
TCP.Client.SetCertificate	Set certificate used by TLS library. The certificate must be sent after the command and the transmission is finished by "<CR><LF>"
TCP.Client.ShowCertificate	Shows the used TLS certificate
TCP.Client**2**.SetCertificate	Set certificate used by TLS library The certificate must be sent after the command and the transmission is finished by "<CR><LF>"
TCP.Client2.ShowCertificate	Shows the used TLS certificate
TCP.Client2.Connect	Opens a 2nd TCP connection
TCP.Client2.Disconnect	Closes the 2nd TCP connection
TCP.Client2.State	Shows the state of the 2nd TCP connection
TCP.Client2.Send,<protocols>,<"text">	Sends out a TCP packet from the 2nd TCP connection
TCP.Client2.ClearSendBuffer	Clears the outgoing TCP buffer for the 2nd TCP connection
TCP.Client2.FlushSendBuffer	Flushes the outgoing TCP buffer for the 2nd TCP connection
TCP.Client2.TxKey=<"key">	AES: Encrypts the outgoing TCP packet on the 2nd TCP connection
TCP.Client2.RxKey=<"key">	AES: Decrypts the incoming TCP packet on the 2nd TCP connection
MSG.Event[,<Serial0|Serial1|USB|TCP|**TCP2**|User>],<"text">	Generate interface event with text, if the interface parameter is omitted sending event to the current one. <Implemented for the 2nd TCP channel TCP2>
SYS.Lua.Start[,<"script.lua">]	Load specific LUA script
SYS.Lua.Clear[,<"script.lua">]	Delete specific LUA script
SYS.Lua.Info[,<"script.lua">]	Comment of specific LUA script
SYS.Lua.Write[,<"script.lua">]	Write specific LUA script. The data must send after the command and the transmission is finished by <CR><LF>
CNF.Load,<"/sys/file.txt">	PFAL command with which the existing config can be overwritten with a file from flash
SYS.CAN.CANopen.enable[,<hex_node_id>]	Enable CANopen on the main interface (8-pin connector). Node ID configurable is with <hex_node_id>- Hex number 0 .. 7f

SYS.CAN.CANopen.disable	Disable CANopen on the main interface (8-pin connector)
SYS.CANB.CANopen.enable[,<hex_node_id>]	Enable CANopen on IOBOX-CAN or IOBOX-WLAN Node ID configurable is with <hex_node_id>- Hex number 0 .. 7f
SYS.CANB.CANopen.disable	Disable CANopen on IOBOX-CAN or IOBOX-WLAN
SYS.CAN.CANopen.cmd,"<CIA309-3 gateway command>" or SYS.CANB.CANopen.cmd,"<CIA309-3 gateway command>"	$CANopen <CIA309-3 gateway answer> <CIA309-3 gateway command>: [[net] node] r[ead] <multiplexor> <datatype> <datatype>: b,i8,i16,i32,u8,u16,u32,r32,vs,os,us,t,td,d,i24,r64,i40,i48,i56,i64,u24,u40,u48,u56,u64 [[net] node] w[rite] <multiplexor> <datatype> <data> <datatype>: b,i8,i16,i32,u8,u16,u32,r32,vs,os,us,t,td,d,i24,r64,i40,i48,i56,i64,u24,u40,u48,u56,u64 [[net] node] start [[net] node] stop [[net] node] preop[erational] [[net] node] reset node [[net] node] reset comm[unication] [net] set sdo_timeout <ms> [net] set node <value> [net] set sdo_block 0|1 [[net] node] set rpdo <nr> <COB> <tx-type> <nr-of-data> <map-obj1>[..<map-obj16>] <tx-type> ::= "event"
Configuration parameters: $PFAL,CNF.Set,<Config>	
<Config>	
DEVICE.COMM.TCP.CLIENT2=<input_mode>,<output_mode>	
TCP.CLIENT2.CONNECT=<auto>,<ip_addr>,<port>	Specifies a 2nd TCP port setup
TCP.CLIENT2.ALTERNATIVE=<auto>,<ip_addr>,<port>,<timeout>	Specify an alternative TCP port for the 2nd TCP connection
TCP.CLIENT2.SENDMODE=<mode>[,<buflevel>]	Sendmode setting (mode:0..2) and TCP buffer level for the 2nd TCP connection. Only one TCP channel can be configured with SENDMODE=2. TCP.CLIENT.SENDMODE=2 or TCP.CLIENT2.SENDMODE=2
TCP.CLIENT2.PING=<active>,<time>	Ping message configuration for the 2nd TCP connection
TCP.CLIENT2.TIMEOUT=<timeout>,<retry_wait>	Connect and reconnect timeouts for the 2nd TCP connection
TCP.CLIENT2.LOGIN.EXT=<text>	Login extension information for the 2nd TCP connection
TCP.CLIENT2.LOGIN=<login>[,<security>]	Login and security settings for the 2nd TCP connection (security is 0:Disabled, 1:AES-EBC, 2:AES-CBC, 3:TLS)
TCP.CLIENT2.DNS.TIMEOUT=<timeout>	Timeout for DNS cache for the 2nd TCP connection (0 disables caching)

TCP.CLIENT2.CERT=<On|Off|Optional>	Using of certificates for TLS for the 2nd TCP connection
WHITELIST.ACTIVE=<first>,<last>	Activates part of the white list using the indices. Example: PFAL,cnf.set,WHITELIST.ACTIVE=0,3 The above example means the first 4 whitelist entries are now active.
Events/States	
GPS.Nav.eCellLocate	Get celllocate data (Lat/Lon) from µBlox server NOTE: the Celllocate works only when no GNSS is available
TCP.Client2.eConnecting	"Connecting" event for the 2nd TCP connection
TCP.Client2.eConnected	"Connected" event for the 2nd TCP connection
TCP.Client2.eDisconnecting	"Disconnecting" event for the 2nd TCP connection
TCP.Client2.eDisconnected	"Disconnected" event for the 2nd TCP connection
TCP.Client2.sConnecting	"Connecting" state for the 2nd TCP connection
TCP.Client2.sConnected	"Connected" state for the 2nd TCP connection
TCP.Client2.sDisconnecting	"Disconnecting" state for the 2nd TCP connection
TCP.Client2.sDisconnected	"Disconnected" state for the 2nd TCP connection
TCP.Client2.sIdle	"Idle" state for the 2nd TCP connection
SYS.eCO.PDO="pdo <PDOnum> <nvals> <val1> ... <valn>"	CANopen event. <PDOnum> - index of PDO entry <nvals> - number of values
Dynamic entries	
&(CO.pdo)	CANopen: shows the latest PDO event content
&(CanERR)	It shows what type of CANbus error has been generated, if any (please refer to Can.error=<cond>[,<cond>] on page 7 in the RelNotes_Firmware_AVL_3.2.0_rev6.pdf)
&(TCPClient2Online)	Reports the status (0 or 1) of the 2nd TCP connection. 1: Connected 2: Disconnected
&(TCPClient2)	Reports the status of the 2nd TCP connection in textual form (connecting/Connected/Disconnecting/Disconnected)
&(TCP2Text)	Outputs the last received TCP packet on the 2nd TCP connection.

NEW FEATURES FOR FOX3-3G-BLE:

This section presents what is new since the latest released firmware regarding the FOX3-3G-BLE.

- ✓ See “key features”, “Improvements” and “Bug fixes”.

NEW FEATURES FOR FOX3-3G-BID:

This section presents what is new in the firmware release regarding the product FOX3-3G-BID.

- ✓ See “key features”, “Improvements” and “Bug fixes”.

NEW FEATURES FOR IOBOX-MINI:

This section presents what is new in the firmware release regarding the IOBOX-MINI.

- ✓ See “key features”, “Improvements” and “Bug fixes”.

NEW FEATURES FOR NFC Reader:

This section presents what is new since the latest released firmware regarding the Near Field Communication (NFC) reader.

- ✓ None

NEW FEATURES FOR BOLERO40 Series:

This section presents what is new since the latest released firmware regarding the BOLERO40.

- ✓ See “key features”, “Improvements” and “Bug fixes”

NEW FEATURES FOR IOBOX-CAN:

This section presents what is new since the latest released firmware regarding the IOBOX-CAN.

- ✓ CANopen feature for CANB (see the CANopen commands in the previous table)

NEW FEATURES FOR IOBOX-WLAN:

This section presents what is new since the latest released firmware regarding the IOBOX-WLAN.

- ✓ CANopen feature for CANB since IOBOX-WLAN supports CANB (see the CANopen commands in the previous table)

NEW PREMIUM-FEATURES

This section presents what is new since the latest released firmware regarding the PREMIUM-FEATURES.

- ✓ TLS 1.2 Support
- ✓ The TLS feature is only available if the premium setting AES is activated

NEW FEATURES FOR LUA:

- ✓ PFAL & LUA commands/Constants/event/states. The “Application-Notes” on how to get started with Lua is in preparation.

SYS.Lua.Start[,<"script.lua">]	Load specific LUA script
SYS.Lua.Clear[,<"script.lua">]	Delete specific LUA script
CNF.Load,<"/sys/file.txt">	PFAL command with which the existing config can be overwritten with a file from flash

PFAL commands	
SYS.Lua.Start[,<"script.lua">]	Load specific LUA script
SYS.Lua.Clear[,<"script.lua">]	Delete specific LUA script
CNF.Load,<"/sys/file.txt">	PFAL command with which the existing config can be overwritten with a file from flash.
SYS.Lua.Start	Starts the Lua script loaded to the device. To automate starting the LUA script, an alarm configuration line is needed: $PFAL,CNF.Set,AL1=Sys.Device.eStart:SYS.Lua.start
SYS.Lua.Stop	Stops a running the Lua script loaded to the device
SYS.Lua.Dump	Reads the source code of that Lua script available on the device
SYS.Lua.Lock,<"password">	Locks the Lua script with a password from reading
SYS.Lua.Unlock,<"password">	Unlocks the Lua script
SYS.Lua.Dump[,<"password">]	Reads the source code of that Lua script available on the device that is locked with a password
SYS.Lua.Clear	Clears the Lua script available on the device
SYS.LUA.Info	Shows the header of installed script
SYS.LUA.Event,<id>,<"text">	Generates custom events for LUA
LUA Commands	
os.sleep(millies)	Suspends the execution of the current thread until the time-out interval in milliseconds elapses
os.trace("format", args)	LUA debug output (“DBG.EN=1” must enabled)
PFAL command request	
bState, sResult := avl.pfal("command")	Reads the state and the result of the execution of the PFAL command that has been defined in the “command” field
PFAL alarm request	
socket:close([force:0..1])	Close socket (force to close immediately)
ev := avl.event(timeout)	When an event happens in the device, the FOX3 creates an event type, puts details into it and passes it to the Lua. The “ev” reads that event type. To read the type and data of that event use the one of the event listed under “Event Requests”. For example: ev = **avl**.event(1000) if ev ~= nil then **if** ev.type == **ALARM_SYS_BLE_TAGDATA** **then** ble_data = ev.u_string os.trace("DATA = [%s]", ble_data); end; end;
ev := avl.setevent(id[,"text"])	Send event back to LUA, this is identical to the command “SYS.LUA.Event,<id>,<"text">” when executed
avl.useevent(type[,OnOff])	Mask/Unmask LUA event types (i.e ALARM_SYS_BLE_TAGDATA,ALARM_SYS_CANMSG)

LUA Event Requests	
ev := [ev.type ev.time ev.idx ev.u_value ev.u_string ev.u_starttype ev.u_startreason ev.u_recvdata ev.u_recvlen ev.u_ipadress ev.u_opid ev.u_opname ev.u_callid ev.u_smsnum ev.u_smstext ev.u_msgid ev.u_msgtype ev.u_msglen ev.u_msgdata]	The “ev” reads the type and data of event // values of “ev.u_xxx” fields depending on the event type // integer event type // integer timestamp // integer subindex // integer value type // string value type // integer starttype // integer startreason // string recvdata buffer // integer recvlen length // string ipaddress // integer operator id // string operator name // string caller name // string SMS number // string SMS text // CAN msg id // CAN msg type // CAN msg length // CAN msg data
LUA EVENTS / Notification	
ALARM_SYS_CO_PDO_RECEIVED	LUA event for CANopen PDOs. Example: if e.type == ALARM_SYS_CO_PDO_RECEIVED then os.trace("CANopen PDO data \"%s\" (%d bytes) (%d ms)", e.u_recvdata, e.u_recvlen, t)
ALARM_SYS_DEVICE_WAKEUP	This event is created after the device is woken up from a sleep mode
ALARM_SYS_DEVICE_START	This event is created after the device has been successfully started up
ALARM_SYS_DEVICE_SHUTDOWN	This event is created before the device is being shut down (turned off or go sleeping)
ALARM_SYS_DEVICE_OVERVOLTAGE	This event is created when the device detects overvoltage on the input power supply
ALARM_SYS_TIMER	This event is created whenever a Timer runs out.
ALARM_SYS_TRIGGER	This event is created whenever a Trigger changes its state
ALARM_SYS_COUNTER	This event is created whenever a Counter changes its state
ALARM_SYS_nvCOUNTER	This event is created whenever a nvCounter changes its state
ALARM_SYS_ERROR	This event is created whenever a system error is detected
ALARM_SYS_USEREVENT0	This event is created whenever a user event 0 to 9 is detected accordingly
ALARM_SYS_USEREVENT1	
ALARM_SYS_USEREVENT2	
ALARM_SYS_USEREVENT3	
ALARM_SYS_USEREVENT4	
ALARM_SYS_USEREVENT5	

ALARM_SYS_USEREVENT6	
ALARM_SYS_USEREVENT7	
ALARM_SYS_USEREVENT8	
ALARM_SYS_USEREVENT9	
ALARM_SYS_SERIALDATA0	This event is created whenever the device detects incoming data on the serial port 0, 1 accordingly
ALARM_SYS_SERIALDATA1	
ALARM_SYS_USBDATA	This event is created whenever the device detects incoming data on the USB port
ALARM_SYS_BLE_TAGDATA	This event is created whenever the device detects Manufacture Specific Data advertised from the scanned Bluetooth Low Energy beacons
ALARM_SYS_CAN	This event is called whenever the device detects incoming data from the CAN interface
ALARM_SYS_TIMESYNC	This event is created whenever the device detects time synchronization
ALARM_SYS_OBDII_DTC	This event is created whenever the device detects incoming data from the OBDII DTC interface
ALARM_SYS_OBDII	This event is created whenever the device detects incoming data from the OBDII
ALARM_SYS_FMS_VAR	This event is created whenever the device detects incoming data from the FMS VAR
ALARM_SYS_J1939_VAR	This event is created whenever the device detects incoming data from the J1939 VAR
ALARM_SYS_FMS	This event is created whenever the device detects incoming data from the FMS interface
ALARM_SYS_J1939	This event is created whenever the device detects incoming data from the J1939 interface
ALARM_SYS_1WIRE_REGISTER	This event is created whenever a 1-Wire device is connected and registered to the 1-Wire interface of the FOX3-2G/3G/4G
ALARM_SYS_1WIRE_RELEASE	This event is created whenever a 1-Wire device is released from the 1-Wire interface of the FOX3-2G/3G/4G
ALARM_SYS_BAT_LOWBAT	This event is created whenever the internal battery gets low
ALARM_SYS_BAT_CHARGE	This event is created whenever the internal battery starts charging process.
ALARM_SYS_POWER_DETECTED	This event is created whenever a connection to an external power supply is detected
ALARM_SYS_POWER_DROPPED	This event is created whenever the external power supply is dropped
ALARM_SYS_NFC_DETECTED	This event is created whenever the external NFC reader detects/reads a NFC tag
ALARM_SYS_NFC_RELEASED	This event is created whenever a connected NFC reader loses the attached NFC TAG
ALARM_SYS_BLE_REGISTER	This event is created whenever the device detects a BLE tag during scanning
ALARM_SYS_BLE_RELEASE	This event is created whenever the device loses a detected BLE tag after scanning ends

ALARM_SYS_BLE_CONNECTED	This event is created once a connection is established between the FOX3-3G-BLE as a peripherals and one central device (such as a mobile phone)
ALARM_SYS_BLE_DISCONNECTED	This event is called once the FOX3-3G-BLE is disconnected from the central device (such as a mobile phone)
ALARM_SYS_BLE_SCANEND	This event is created once the FOX3-3G-BLE has ended a scan session for BLE sensors
ALARM_SYS_WLAN_CONNECTING	This event is created when the IOBOX-WLAN is trying to connect to one of 5 wireless access points
ALARM_SYS_WLAN_CONNECTED	This event is created once the IOBOX-WLAN is connected to one of 5 wireless access points
ALARM_SYS_WLAN_DISCONNECTED	This event is created once the IOBOX-WLAN is disconnected from one of 5 wireless access points
ALARM_SYS_WLAN_RECEIVED	This event is created whenever the IOBOX-WLAN receives data from one of 5 wireless access points
ALARM_SYS_WLAN_TCP_CONNECTED	This event is created once a connection is established between the device and remote server over one of 5 wireless access points
ALARM_SYS_WLAN_TCP_DISCONNECTED	This event is created once the device is disconnected from the remote server over one of 5 wireless access points
IO	
ALARM_IO_IN	This event is created whenever a device input/output signal changes its state
ALARM_IO_MOTION_MOVING	This event is created once the device detects moving (IO.Motion.eMoving) based on pre-defined threshold.
ALARM_IO_MOTION_STANDING	This event is created once the device detects standing (IO.Motion.eStanding) based on pre-defined threshold.
ALARM_IO_MOTION_FORCE	This event is created once the pre-configured force acceleration (IO.Motion.eForce) is exceeded.
ALARM_IO_MOTION_3DFORCE	This event is created once the device exceeds the configured force acceleration in one direction (IO.Motion.e3DForce)
ALARM_IO_MOTION_CRASH	Not supported (Event from external motion sensor)
ALARM_IO_MOTION_INTERNAL	Not supported (Event from external motion sensor)
ALARM_IO_MOTION_EXTERNAL	Not supported (Event from external motion sensor)
ALARM_IO_BEARING	This event is created once the device detects moving (IO.Motion.eBearing) based on pre-defined threshold.
GPS	
ALARM_GPS_NAV_FIX	This event is called once the device gets a valid GNSS fix
ALARM_GPS_NAV_HEADING	This event is created once the device detects changes in heading for more than the specified heading tolerance (GPS.Nav.eChangeHeading).
ALARM_GPS_NAV_HEADING2	This event is created once the device detects changes in heading2 for more than the specified heading2 tolerance (GPS.Nav.eChangeHeading2).
ALARM_GPS_GEOFENCE	This event is created once the device detects in/out of one of pre-configured geofences.

ALARM_GPS_AREA	This event is created once the device detects in/out of one of pre-configured areas.
ALARM_GPS_MULTI_GEOFENCE	This event is created once the device detects in/out of one of pre-configured multi-geofences
ALARM_GPS_WAYPOINT_GEOFENCE	This event is created once the device leaves the corridor of preconfigured waypoints.
ALARM_GPS_JAMMING	This event is created once the GPS jamming is detected
ALARM_GPS_ANT_PLUGGED	This event is created once an external GPS antenna is plugged/connected
ALARM_GPS_ANT_UNPLUGGED	This event is created once an external GPS antenna is unplugged/disconnected
GSM	
ALARM_GSM_OPFOUND	This event is created once a GSM network operator is found
ALARM_GSM_OPLOST	This event is created when the GSM network operator is lost
ALARM_GSM_CELLCHANGE	This event is created whenever a GSM cell is changed
ALARM_GSM_CBM	This event is created whenever new cell broadcast message is received
ALARM_GSM_SIMLOST	This event is created whenever a SIM-Card is no longer presents
ALARM_GSM_MCCCHANGE	This event is created whenever a mobile country code is changed
ALARM_GSM_JAMMING	This event is created whenever GSM jamming is detected
ALARM_GSM_VOICECALL_INCOMING_RING	This event is created when an incoming voice call is received
ALARM_GSM_VOICECALL_RING_STOPPED	This event is created when the device stops ringing
ALARM_GSM_VOICECALL_OUTGOING_DIAL	This event is created when an outgoing voice call is dialed
ALARM_GSM_VOICECALL_CALL_ESTABLISHED	This event is created when an outgoing voice call is established
ALARM_GSM_VOICECALL_CALL_FINISHED	This event is created when an outgoing voice call is finished
ALARM_GSM_SMS_INCOMING	This event is created when an SMS is received
ALARM_GSM_SMS_SENT	This event is created when an SMS is sent
ALARM_GSM_GPRS_CONNECTING	This event is created when device starts connecting to GPRS services
ALARM_GSM_GPRS_CONNECTED	This event is created when the device is attached to GPRS services
ALARM_GSM_GPRS_DISCONNECTING	This event is created when the device stars disconnecting from GPRS services
ALARM_GSM_GPRS_DISCONNECTED	This event is created when the device is successfully detached from GPRS services
TCP	
ALARM_TCP_CLIENT_CONNECTING	This event is created when device starts connecting to a TCP server
ALARM_TCP_CLIENT_CONNECTED	This event is created when device is connected to the TCP server
ALARM_TCP_CLIENT_PACKETSENT	This event is created when a TCP packet is sent
ALARM_TCP_CLIENT_PINGSENT	This event is created when a TCP ping is sent
ALARM_TCP_CLIENT_RECEIVED	This event is created when data is received from the TCP server

ALARM_TCP_CLIENT_DISCONNECTING	This event is created when device starts disconnecting from the TCP server
ALARM_TCP_CLIENT_DISCONNECTED	This event is created when device is disconnected from the TCP server
ALARM_TCP_CLIENT_BUFFER_EMPTY	This event is created once the TCP buffer is emptied
ALARM_TCP_CLIENT_FLASHBUFFER_EMPTY	This event is created once the flash buffer is emptied
ALARM_TCP_CLIENT2_CONNECTING	This event is created when device starts connecting to a TCP server
ALARM_TCP_CLIENT2_CONNECTED	This event is created when device is connected to the TCP server
ALARM_TCP_CLIENT2_PACKETSENT	This event is created when a TCP ping is sent
ALARM_TCP_CLIENT2_PINGSENT	This event is created when a TCP ping is sent
ALARM_TCP_CLIENT2_RECEIVED	This event is created when data is received from the TCP server
ALARM_TCP_CLIENT2_DISCONNECTING	This event is created when device starts disconnecting from the TCP server
ALARM_TCP_CLIENT2_DISCONNECTED	This event is created when device is disconnected from the TCP server
ALARM_TCP_CLIENT2_FLASHBUFFER_EMPTY	This event is created once the flash buffer is emptied
ALARM_TCP_CLIENT2_BUFFER_EMPTY	This event is created once the TCP buffer is emptied
ALARM_SYS_CO_PDO_RECEIVED	
ALARM_TCP_SMTP_SENT	This event is created once an email is sent
ALARM_TCP_SMTP_FAILED	This event is created when sending email failed
ALARM_TCP_UDP_RECEIVED	This event is created when receiving data via UDP
FILE	
ALARM_FILE_AVAILABLE	This event is created when file is available
ECODRIVE	
ALARM_ECODRIVE_START	These events are created when the ecodrive is started/stopped/on harsh-turn/-brake/-accelerate
ALARM_ECODRIVE_STOP	
ALARM_ECODRIVE_TURN	
ALARM_ECODRIVE_BRAKE	
ALARM_ECODRIVE_ACCELERATE	
BLUEID	
ALARM_BLUEID_CMD	These events are created when BLUEID gets command, data or tickets
ALARM_BLUEID_DATA	
ALARM_BLUEID_TICKETS	
TYPE	
ALARM_TYPE_INTERNAL	User specific event types for LUA (i.e timer or user events)
LUA	
ALARM_SYS_LUA_START	These events are created when Lua is started or stopped
ALARM_SYS_LUA_STOP	

DTCO	
ALARM_SYS_DTCO_CONFIRM	Confirmation that the message has been sent completely
ALARM_SYS_DTCO_INCOMING	Indication that the requested message has got incoming data
CAN	
ALARM_SYS_CANMSG	This event is created when contents of this CAN message is changed
TCP Socket	
NET_TCP	Socket is used for a TCP connection
NET_UDP	Socket is used for a UDP connection
ALARM_TCP_SOCKET_IFUP	Socket interface is up
ALARM_TCP_SOCKET_IFDOWN	Socket interface is down
ALARM_TCP_SOCKET_CONNECTED	Socket interface is connected
ALARM_TCP_SOCKET_DISCONNECTED	Socket interface is disconnected
ALARM_TCP_SOCKET_RECV	Socket interface has received data
ALARM_TCP_SOCKET_SENT	Socket interface has sent data
IOBOX	
ALARM_SYS_IOBOX_LOST	This event is created when a connection to the IOBOX-MIN/CAN or WLAN is lost
PFAL state request	
state := avl.state(type[,index])	When a state changes in the device, the FOX3 creates a state type, puts details into it and passes it to the Lua. The "state" reads that state type. To read the type and data of that state use the one of the state types listed under "State Requests". For example: st = **avl**.event(1000) if st ~= nil then **if** st.type == STATE_SYS_BLE_CONNECTED **then** ble_data = st.u_string os.trace("DATA = [%s]", ble_data); end; end;
State Requests	
state := [state.type state.idx state.u_bool state.u_value state.u_string state.u_starttype state.u_startreason state.u_opid state.u_opname]	Reads the type and the data assigned to that state // values of "state.u_xxx" fields depends on the state type // integer state type // integer subindex // boolean value type // integer value type // string value type // integer starttype // integer startreason // integer operator id // string operator name
STATES / Notifications	
STATE_SYS_DEVICE_START	Value of the PFAL SYS.Device.sStart state
STATE_SYS_TIMER	Value of the PFAL SYS.Timer.s<id> state

STATE_SYS_TRIGGER	Value of the PFAL SYS.Trigger.s <id> state
STATE_SYS_COUNTER	Value of the PFAL SYS.Counter.s <id> state
STATE_SYS_nvCOUNTER	Value of the PFAL SYS.NVCounter.s <id> state
STATE_SYS_CAN	Value of the PFAL SYS.sCan state
STATE_SYS_BAT_VOLTAGE	Value of the PFAL SYS.Bat.sVoltage state
STATE_SYS_BAT_CHARGE	Value of the PFAL SYS.Bat.sCharge state
STATE_SYS_BAT_MODE	Value of the PFAL SYS.Bat.sMode state
STATE_SYS_POWER_VOLTAGE	Value of the PFAL SYS.Power.sVoltage state
STATE_SYS_1WIRE_REGISTER	Value of the PFAL SYS.Power.sRegister state
STATE_SYS_NFC_DETECTED	Value of the PFAL SYS.NFC.sDetected state
STATE_SYS_BLE_CONNECTED	Value of the PFAL SYS.BLE.sConnected state
STATE_SYS_WLAN_CONNECTED	Value of the PFAL SYS.WLAN.sConnected state
STATE_SYS_WLAN_DISCONNECTED	Value of the PFAL SYS.WLAN.sDisconnected state
STATE_SYS_WLAN_TCP_CONNECTED	Value of the PFAL SYS.WLAN.sTCPConnected state
STATE_SYS_WLAN_TCP_DISCONNECTED	Value of the PFAL SYS.WLAN.sTCPDisconnected state
IO	
STATE_IO_IN	Value of the PFAL IO.IN.s<id> state
STATE_IO_ANA	Value of the PFAL IO.ANA.s<id> state
STATE_IO_PULSECNT	Value of the PFAL IO.PulseCount.s<id> state
STATE_IO_MOTION_MOVING	Value of the PFAL IO.Motion.sMoving state
STATE_IO_MOTION_STANDING	Value of the PFAL IO.Motion.sStanding state
GPS	
STATE_GPS_NAV_FIX	Value of the PFAL GPS.Nav.sFix state
STATE_GPS_NAV_SPEED	Value of the PFAL GPS.Nav.sSpeed state
STATE_GPS_NAV_POSITION	Value of the PFAL GPS.Nav.sPosition state
STATE_GPS_NAV_DIST	Value of the PFAL GPS.Nav.sDist state
STATE_GPS_NAV_DELTASPEED	Value of the PFAL GPS.Nav.sDeltaSpeed state
STATE_GPS_HISTORY_DIST	Value of the PFAL GPS.History.sDist state
STATE_GPS_AREA	Value of the PFAL GPS.Area.s<id> state
STATE_GPS_GEOFENCE	Value of the PFAL GPS.Geofence.s<id> state
STATE_GPS_MULTI_GEOFENCE	Value of the PFAL GPS.MultiGeofence.s<id> state
STATE_GPS_WAYPOINT_GEOFENCE	Value of the PFAL GPS.WPGF.s<id> state
GSM	
STATE_GSM_OPVALID	Value of the PFAL GSM.sOpValid state
STATE_GSM_HOME	Value of the PFAL GSM.sNoRoaming state
STATE_GSM_ROAMING	Value of the PFAL GSM.sRoaming state
STATE_GSM_VOICECALL_READY_FOR_CALL	Value of the PFAL GSM.Voicecall.sReady state
STATE_GSM_VOICECALL_INCOMING_RING	Value of the PFAL GSM.Voicecall.sIncoming state
TATE_GSM_VOICECALL_NUMBER_OF_RINGS	Value of the PFAL GSM.Voicecall.sRingCounter state

STATE_GSM_VOICECALL_OUTGOING_DIAL	Value of the PFAL GSM.Voicecall.sOutgoing state
STATE_GSM_VOICECALL_INSIDE	Value of the PFAL GSM.Voicecall.sInside state
STATE_GSM_GPRS_CONNECTING	Value of the PFAL GSM.GPRS.sConnecting state
STATE_GSM_GPRS_CONNECTED	Value of the PFAL GSM.GPRS.sConnected state
STATE_GSM_GPRS_DISCONNECTING	Value of the PFAL GSM.GPRS.sDisconnecting state
STATE_GSM_GPRS_DISCONNECTED	Value of the PFAL GSM.GPRS.sDisconnected state
TCP	
STATE_TCP_CLIENT_IDLE	Value of the PFAL TCP.Client.sIdle state
STATE_TCP_CLIENT_CONNECTING	Value of the PFAL TCP.Client.sConnecting state
STATE_TCP_CLIENT_CONNECTED	Value of the PFAL TCP.Client.sConnected state
STATE_TCP_CLIENT_DISCONECTING	Value of the PFAL TCP.Client.sdisconnecting state
STATE_TCP_CLIENT_DISCONECTED	Value of the PFAL TCP.Client.sDisconnected state
STATE_TCP_CLIENT2_IDLE	Value of the PFAL TCP.Client2.sIdle state
STATE_TCP_CLIENT2_CONNECTING	Value of the PFAL TCP.Client2.sConnecting state
STATE_TCP_CLIENT2_CONNECTED	Value of the PFAL TCP.Client2.sConnected state
STATE_TCP_CLIENT2_DISCONECTING	Value of the PFAL TCP.Client2.sdisconnecting state
STATE_TCP_CLIENT2_DISCONECTED	Value of the PFAL TCP.Client2.sDisconnected state
ECODRIVE	
STATE_ECODRIVE_START	Value ecodrive state is started
STATE_ECODRIVE_STOP	Value ecodrive state is stopped
STATE_ECODRIVE_SPEED1	Value ecocdrive has speed limit1
STATE_ECODRIVE_SPEED2	Value ecocdrive has speed limit2
STATE_ECODRIVE_SPEED3	Value ecocdrive has speed limit3
GSM	
GSM_DISABLED	Value GSM state is disable
GSM_SLEEP	Value GSM state is sleep
GSM_IDLE	Value GSM state is idle
GSM_INIT_BASE	Value GSM state is initializing base commands
GSM_INIT_MAIN	Value GSM state is initializing main commands
GSM_INIT_NET	Value GSM state is initializing GPRS commands
GSM_VERSION	Value GSM state is checking cellular version
GSM_IMSI_CHECK	Value GSM state is checking IMSI number
GSM_SMS_CHECK	Value GSM state is checking SMS activity
READY_FOR_CALL	Value GSM is ready for call
INCOMING_VOICE_CALL	Value GSM has incoming voice call
INCOMING_DATA_CALL	Value GSM has incoming data call
INCOMING_FAX_CALL	Value GSM has incoming fax call
OUTGOING_VOICE_CALL	Value GSM has outgoing voice call
INSIDE_VOICE_CALL	Value GSM is inside voice call

TMER	
TIMER_ERASED	Timer is cleared
TIMER_INACTIVE	Timer is inactive
TIMER_PAUSED	Timer is paused
TIMER_RUNNING	Timer is running
String formating with dynamic entries	
sResult := avl.format("format", args)	Reads the formatted “args” that has been defined in the “args” field
PFAL variables	
sResult := avl.version()	Reads the firmware version
sResult := avl.device()	Reads the device name
iResult := avl.timer(index)	Reads the timer index
iResult := avl.trigger(index)	Reads the trigger index
iResult := avl.counter(index)	Reads the counter index
iResult := avl.nvcounter(index)	Reads the nvcounter index
PFAL file transfer	
len := avl.file_upload(buffer)	Reads the length of the file
GPS state and data	
sValue := avl.gps_version()	Reads the GPS firmware version
tResult := avl.gps_data()	Reads the current GPS data
tResult := avl.gps_sats()	Reads the GPS satellites in use
GPS data	
record := [lat lon alt speed course ecef_x ecef_y ecef_z dop time fix]	Reads the GPS values listed within the [] square brackets // Latitude (degree) // Longitude (degree) // Altitude (meter) // speed (m/s) // course (degree) // ECEF-X (meter) // ECEF-Y (meter) // ECEF-Z (meter) // pdop value // time (seconds) // fix (boolean)

GPS satellites record	
record := [gps_num gps_sat1 .. gps_sat12 gls_num gls_sat1 .. gls_sat12]	Reads the GPS values listed within the [] square brackets // Number of GPS satellites // Dump of satellite data // "SatID,Elevation,Azimuth,AvgCNo,Used" // Number of GLS satellites // Dump of satellite data // "SatID,Elevation,Azimuth,AvgCNo,Used"
GSM state and data	
sValue := avl.gsm_version()	Reads the GSM firmware version
tResult := avl.gsm_data()	Reads the current GSM data
sValue := avl.gsm_imei()	Reads the IMEI of the device
sValue := avl.gsm_imsi()	Reads the IMSI of the SIM card
sValue := avl.gsm_iccid()	Reads the ICCID of the SIM card
GSM data	
record := [state csq creg cpas lac cellid opid opname callstate callnumber]	Reads the GSM values listed within the [] square brackets // GSM state // CSQ value // CREG value // CPAS value // local area code // cell id // operator id // operator name (string) // call state **// caller number (string)**
Motion data	
tResult := avl.motion_data()	Reads the motion data
Motion data	
record := [val_x val_y val_z min_x min_y min_z max_x max_y max_z nsum_x nsum_y nsum_z]	Reads the motion values listed within the [] square brackets // Current X acceleration // Current Y acceleration // Current Z acceleration // Min. X acceleration in <g_coe> interval // Min. Y acceleration // Min. Z acceleration // Max. X acceleration in <g_coe> interval // Max. Y acceleration // Max. Z acceleration // Normal X gravitation in <g_coe> interval // Normal Y gravitation **// Normal Z gravitation**

Timer variable	
timer := avl.tick(interval, event_type);	
timer:start([time])	Restarts a timer or start a timer with a new interval
timer:stop()	Stops the timer
timer:single()	Restarts a single timer
timer:cyclic()	Restarts a cyclic timer
iResult := timer:id()	Reads the timer event type
iResult := timer:interval()	Reads the timer interval time
iResult := timer:elapsed()	Reads the timer elapsed time
Socket interface	
socket := net.create_socket([type, param]) socket:connect(<"IP"|"URL">, port) socket:close([force:0..1]) ocket:flush() socket:hold() socket:unhold() tVal := socket:unsent() tVal := socket:ttl([ttl]) tVal := socket:bufsize([bytes]) tBytes := socket:send(data) data, tBytes := socket:recv() tIP, tPort := socket:getaddr() tIP, tPort := socket:getpeer() tIP := net.dns_resolve("URL") socket:on(<"connection"|"disconnection"|"sent"|"receive">, function())	TCP socket implementation for LUA // Create a socket // Connect the socket to peer // Close the socket // Flush the socket // Hold the socket // Unhold the socket // Unsent socket data // Set/Read TTL value // Set/Read buffer size // Send data to socket // Read data from socket // Socket address // Peer address // Resolve URL // Socket callbacks
LUA I2C access	
count := avl.i2c_read(addr, register, data)	Read data from I2C devices connected to FOX3-2G/3G/4G
count := avl.i2c_write(addr, register, data)	Write data to I2C devices connected to FOX3-2G/3G/4G
avl.i2c_reset()	Reset the I2C bus
LUA DTCO-commands	
tBytes = dtco.iso_send(TA, strData)	Sends requests to the specified address: tBytes - count of transmitted bytes TA - target address strData - string variable
tData, tBytes, SA := dtco.iso_recv()	Reads the data the tachograph has transmitted on request: tData - received data tBytes - cound received bytes SA - source address

LUA file access	
file := io.open(filename [, mode]) io.lines (filename) io.read(...) io.write(...) io.type (file) io.flush(file) io.close(file) file:read(···) file:write(···) file:lines() file:flush() file:close() file:seek ([whence] [, offset]) os.remove(name) os.rename(oldname, newname) os.mkdir(name)	Working with files // Open a file // Read one line from file // Read data from file // Write data to file // Type of file // Flush written data // Close file // File operations // Remove a file on disk // Rename a file on disk // Make a directory
LUA library	
os.clock(), os.date(), os.time(), os.difftime(), os.exit(), os.execute(), os.getenv(), os.setenv(), os.sleep(), os.setlocale()	Documentation for LUA under <https://www.lua.org/manual/5.2> or <https://www.lua.org/pil/contents.html>
coroutine.create(), coroutine.resume(), coroutine.running(), coroutine.status(), coroutine.wrap(), coroutine.yield()	
string.byte(), string.char(), string.dump(), string.find(), string.format(), string.gmatch(), string.gsub(), string.len(), string.lower(), string.match(), string.rep(), string.reverse(), string.sub(), string.upper(), string.replace()	
table.concat(), table.insert(), table.pack(), table.unpack(), table.remove(), table.sort()	
math.abs(), math.acos(), math.asin(), math.atan2(), math.atan(), math.ceil(), math.cosh(), math.cos(), math.deg(), math.exp(), math.floor(), math.fmod(), math.frexp(), math.ldexp(), math.log(), math.max(), math.min(), math.modf(), math.pow(), math.rad(), math.random(), math.randomseed(), math.sinh(), math.sin(), math.sqrt(), math.tanh(), math.tan()	
bit32.arshift(), bit32.band(), bit32.bnot(), bit32.bor(), bit32.bxor(), bit32.btest(), bit32.extract(), bit32.lrotate(), bit32.lshift(), bit32.replace(), bit32.rrotate(), bit32.rshift()	

NEW FEATURES FOR CAN_CPC:

This section presents what is new since the latest released firmware regarding the Premium-Feature CAN_CPC.

- ✓ None

REMOVED/NOT SUPPORTED:

This section presents what has been removed/not supported since the latest released firmware.

- ✓ The TCP.SERVICE which has limited functions has been removed and replaced by TCP.CLIENT2

CHANGES:

This section presents what has been changed since the latest released firmware.

- ✓ Disable uFOTA for SARA-R4 to prevent any connection impact when using private APN.

IMPROVEMENTS:

This section presents what has been improved in this firmware release.

- ✓ Changed the thread priority of LUA and communication thread to get a better balance between both
- ✓ PDP context optimization for a special case.
- ✓ Optimization in GSM initialization when no SIM is inserted (Devices with SARA R412)
- ✓ IOBOX-CAN (V1.3):
  - * find routine in software filter optimized
  - * Interrupt routines and priorities optimized
- ✓ Bios: Bios-Communication
- ✓ The Driver-ID via FMS can be read completely with simultaneous multiple source addresses.

BUGS FIXED:

This is a list of internal problems that were fixed in the current firmware release.

- ✓ Check time format for **SYS.SetTime** to prevent setting invalid time
- ✓ CANopen command parser for "**set rpdo**" fixed
- ✓ Fixed SMS text length for **GSM.SMS.eIncoming**
- ✓ Fixed **TCP.Storage.AddRecord**

KNOWN ISSUES:

This is a list of known issues in the current firmware release.
