Lantronix AVL FIRMWARE RELEASE

VERSION: avl_3.3.0_rc15

BIOS version: 3.0.2
Official release date: 10/02/2019
Affecting deliveries from: 08/21/2019
List of firmware files: avl_3.3.0_rc15_20190919.frp
avl_3.3.0_rc15-Ze4114daf.zip
avl_3.3.0_rc15_20190919.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

NOTE: 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.

DOCUMENTATION:

Filename	Description
AVL_PFAL_Commands_Set.pdf	Lists and describes all PFAL commands supported by this firmware release.

SUMMARY OF CHANGES:

Version	Description	Created by	Date (M/D/Y)
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_3.3.0_rc15***" and is intended for use as a reference when updating an AVL device to version "***avl_3.3.0_rc15***".

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,"url",80** on the device.

DON'T switch off the AVL device while it reboots after the firmware update. The duration of the reboot after the firmware update may take approx. 45 seconds.

Before upgrading the firmware on the device, it is strongly recommended to upload and backup 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.

NOTE 1: The latest FW 3.3.0_rc15 is for FOX3-2G/3G/4G as well as BOLERO40 series.

NOTE 2: This firmware version (3.3.0_rc15) currently does not support Sleep=Ring and DOZE on BOLERO40 series.

NOTE 3: Doze mode and Sleep=Ring on BOLERO40 series are in preparation for the next AVL firmware release. In future Sleep=RING works only in SIMSLOT2 (the upper SLOT) on BOLERO40 series.

3) Firmware Installation Notes

The installation package consists of firmware in three different formats *.txt, *.frp and *.zip. 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.	

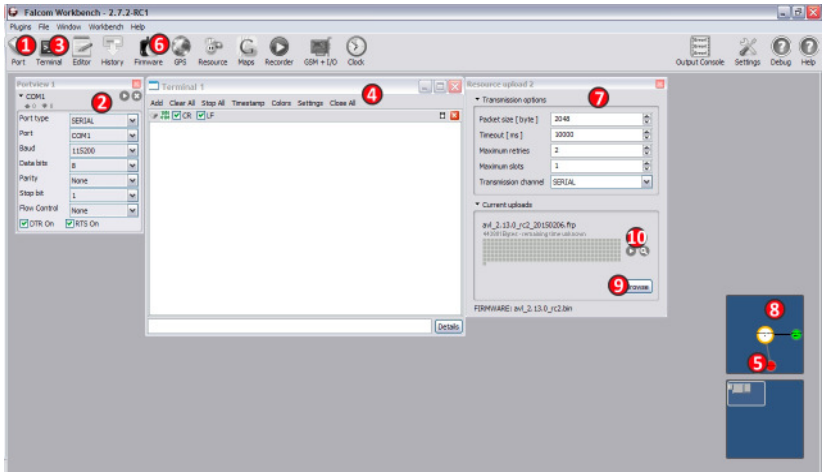
4) Prerequisites concerning the PC

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

1. Download "**avl_3.3.0_rc15.zip**" and extract the file you downloaded into a temporary folder on your PC.
2. Run the "**workbench**" software.



(a) 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.
12. Click **Help** (11) button for more details about the workbench software.

6) New and Modified Functions

NEW FEATURES:

This is a list of new features in the current firmware release.

✓ Added new PFAL commands/configuration/event/states/dynamic variables.

PFAL Commands	
MSG.Event[,<interface>],<"text">	Generates interface event with text, if the interface parameter is omitted sending event to the current one. Optional <port>: Serial0 Serial port 0 Serial1 Serial port 1 USB USB port User User interface TCP TCP interface <"text">: Any text can be specified
Sys.Can.DTCO.FMS.Enable	Enable tachograph communication on the 1st CAN port (main port)
Sys.Can.DTCO.FMS.Disable	Disable tachograph communication on the 1st CAN port (main port)
Sys.Can.DTCO.SendAPDU,<TA>,"bytes"	It transmit n bytes b0...bn to target with address <TA> b0 ... bn are coded as space separated hex values 0x<nn> e.g.: $pfal,Sys.Can.DTCO.SendAPDU,0xee,"0x10 0x7e"
Sys.Can.Timeout,<timeout_s>	CAN timeout in s for the 1st CAN port
Alarm.Call,<index>	Executes actions on the specified alarm index in range 0 -99 or if you are using the premium feature ALARM it ranges from 0-249.
GPS.Nav.HoldPosition=<mode>	Holds the current GPS position at this location to avoid GPS position jumps around from your real parking location. <mode> settings: 0 Deactivates holding of GPS position. 1 Activates holding of GPS position
Msg.Send.RawFlashBuffer,<protocols>,<"text">	Stores the specified protocols and/or user text in raw format to the nonvolatile TcpFlashBuffer inside the device <protocols> settings: Specifies the protocols to be sent to the buffer <"text"> Up to 1450 chars.
Msg.Send.RawTCPBuffer,<protocols>,<"text">	Stores the specified protocols and/or user text in the raw format to the temporary TcpBuffer inside the device. <protocols> settings: Specifies the protocols to be sent to the buffer <"text"> Up to 1450 chars.
Msg.Send.RawTCP,<protocols>,<"text">	Sends the specified protocols and/or user text in the raw format to the connected server <protocols> settings: Specifies the protocols to be sent to the buffer <"text"> Up to 1450 chars.

Msg.ClearBuffer,<interface>	Clears the input/output buffer of the specified port. <port> settings: Serial0 Serial port 0 Serial1 Serial port 1 USB USB port TCP TCP port
SYS.Device.HealthState	Shows battery and lifetime statistics
SYS.can.UpdateIoBox,"<filename>"	Updates IOBOX-CAN from file system
Configuration parameters	
DEVICE.CAN.DTCOFMS.STARTUP=<mode>,<port>	Enables or disables reading of tachograph data via FMS interface on the specified CAN port <mode> off Disable reading of tachograph data. on Enables reading of tachograph data. <port> 0 Enables reading of DTCO on 1st CAN. 1 Enables reading of DTCO on IOBOX-CAN
DEVICE.CAN.FMS.STARTUP=<format>,<port>	Defines the frame format to enable reading of FMS messages on the specified CAN port. <format> Std Enables the 11-bit identifier (CAN2.0A). Ext Enables the 29-bit identifier (CAN2.0B) <port> 0 Enables reading of DTCO on 1st CAN. 1 Enables reading of DTCO on IOBOX-CAN
DEVICE.CAN.STARTUP<port>=<on_off>,<baud>,<mode>	<can_port> 0 Enables reading on 1st CAN interface (main port). 1 Enables reading on 2nd CANB interface (IOBOX-CAN) <on_off> off Disable CAN functionality on the specified port. on Enables CAN functionality on the specified port. <baud> 10K -CAN interface operates at 10 Kbits/s. 20K -CAN interface operates at 20 Kbits/s. 33K3 -CAN interface operates at 33.3 Kbits/s. 50K -CAN interface operates at 50 Kbits/s. 83K3 -CAN interface operates at 83.3 Kbits/s. 95K2 -CAN interface operates at 95.2 Kbits/s . 100k -CAN interface operates at 100 Kbits/s . 125K -CAN interface operates at 125 Kbits/s. 250K -CAN interface operates at 250 Kbits/s. 500K -CAN interface operates at 500 Kbits/s. 666K6 -CAN interface operates at 666.6 Kbits/s 800K -CAN interface operates at 800 Kbits/s. 1M -CAN interface operates at 1024 Kbits/s. <mode> RO Read Only mode (Silent mode). RW Read Write mode (Running mode) Some CAN interfaces require the ACK. Thus, RW is needed. LB Loop back mode, for self-test function SLB Loop back combined with silent mode, for self-test function
DEVICE.DTCO.D8=B0,<format>	Enables IN4 (PIN 8) on the IOBOX-CAN for reading tachograph data via D8 interface.

	<format> settings VDO VDO data format. SRE Stoneridge data format.
DEVICE.DTCO.D8=off	Disables the IN4 (PIN 8) on the IOBOX-CAN for reading tachograph data via D8 interface
DEVICE.CAN.ERR.EVENTS=<mode>	Activates error CAN events. Default: off
Events/States	
Can.error=<cond>[,<cond>]	This event occurs when the received error from CAN bus matches the user specified condition(s) <cond> settings: **STUFF** - More than 5 equal bits in a sequence have occurred **FORM** - A fixed format part of a received frame has the wrong format **ACK** - Sent message was not acknowledged by another node **BIT1** - Bit 1 error (monitored bus value was dominant) **BIT0** - Bit 0 error (monitored bus value was recessive) **CRC** - The CRC check sum was incorrect in the message received **WARNING_RX** - TX error counter (TEC) reached warning level (>96) **WARNING_TX** - RX error counter (REC) reached warning level (>96) **PASSIVE** - CAN "error passive" occurred **BUS_OFF** - CAN "bus off" error occurred **OVERRUN_RX** - Overrun in RX queue or hardware occurred **OVERRUN_TX** - Overrun in TX queue occurred **ARBITRATION_LOST** - Arbitration lost **PHY_FAULT** - General failure of physical layer detected (if supported by hardware) **PHY_H** - Fault on CAN-H detected (Low Speed CAN) **PHY_L** - Fault on CAN-L detected (Low Speed CAN)
Can.eStat=<mode>	This event is occurred when the 1st CAN bus (on the main port) goes idle or active accordingly. <mode> settings: idle - CAN bus is in idle mode (CAN stops sending data for the user specified time out). active - CAN bus is in active mode(CAN starts sending data for the user specified time out) Example: $pfal,config.set,AL10=Sys.Can.estat=idle:Msg.Send.Serial0,0,"CAN idle" $pfal,config.set,AL11=Sys.Can.estat=active:Msg.Send.Serial0,0,"CAN active" $pfal,sys.can.Timeout,10
SYS.Iobox.eReset[=<cond>[...<cond>]]	This event is occurred when the IOBOX-CAN/MINI/WLAN makes software reset. Optional <cond> settings: "PINRST" 0x04 PIN reset

	"PORRST" 0x08 POR/PDR reset "SFTRST" 0x10 Software reset "IWDGRST" 0x20 Independent watchdog reset "WWDGRST" 0x40 Window watchdog reset "LPWRRST" 0x80 Low-power reset
SYS.Device.eStart=Reset,WatchdogIobox	This event is occurred when IOBOX-CAN/MINI/WLAN watchdog causes a device reset.
SYS.IoBox.eLost	This event is occurred when IOBOX-MINI/CAN/WLAN is disconnected.
SYS.1Wire.sAvailable=whitelist	Compares the id of 1-Wire devices with the values of whitelist and sets this state to true as long as the comparison matches.
SYS.eCan.DTCO.Confirm	True if the incoming APDU message has data.
SYS.eCan.DTCO.Incoming	This event is generated whenever an APDU message has been sent completely to the tachograph. This is just a confirmation event.
SYS.sCan.DTCO.Confirm	This event is generated whenever a new incoming APDU message is received $PFAL,CNF.Set,AL1=Sys.eCan.DTCO.Incoming:Msg.Send.Serial0,0,"dtco: &(Can.Dtco.Incoming)"
Dynamic entries	
&(RAT)	It is used to report the GSM radio access technology. 0: GSM (2G) 1: GSM COMPACT 2: UTRAN 3: GSM with EDGE availability 4: UTRAN with HSDPA availability 5: UTRAN with HSUPA availability 6: UTRAN with HSDPA and HSUPA availability 7: LTE
&(UnixTime2)	It is used to report the number of seconds since 1.1.1970 UTC (UNIX epoch time)
&(UserEventText)	It is used to report the user text causes the event
&(Can.Dtco.Incoming)	Used to report the incoming data from the tachograph in the format "<SA> <TA> <xx>...<xx>" <SA> hexadecimal source address <TA> hexadecimal target address <xx> hexadecimal data bytes

NEW FEATURES FOR FOX3-3G-BLE:

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

✓ Added new PFAL commands/configuration/event/states/dynamic variables for BLE tags/sensors:

PFAL Commands	
SYS.BLE.ListDev[=<filter>]	List of active beacons with corresponding content. Optional <filter> settings: Name Show the list with names. MAC Show the list with MACs.

	UUID Show the list with UUIDs. RSSI Show the list with RSSIs.
SYS.BLE.ListAdd[=<filter>]	List of added beacons with corresponding content Optional <filter> settings: Name Show the list with names. MAC Show the list with MACs. UUID Show the list with UUIDs. RSSI Show the list with RSSIs.
SYS.BLE.ListRel[=<filter>]	List of released beacons with corresponding content. Optional <filter> settings: Name Show the list with names. MAC Show the list with MACs. UUID Show the list with UUIDs.
Dynamic entries	
&(BLE.ListDev[:<filter>])	It is used to report the list of active beacons Optional <filter> settings: Name Show the list with names. MAC Show the list with MACs. UUID Show the list with UUIDs. RSSI Show the list with RSSIs.
&(BLE.ListAdd[:<filter>])	It is used to report the list of added beacons Optional <filter> settings: Name Show the list with names. MAC Show the list with MACs. UUID Show the list with UUIDs. RSSI Show the list with RSSIs.
&(BLE.ListRel[:<filter>])	It is used to report the list of released beacons Optional <filter> settings: Name Show the list with names. MAC Show the list with MACs. UUID Show the list with UUIDs.
&(BLE.Name[:index])	It is used to report the Name of active beacon. Optional <index> settings: 0…n The indexed tag during scanning.
&(BLE.MAC[:index])	It is used to report the MAC of active beacon Optional <index> settings: 0…n The indexed tag during scanning.
&(BLE.UUID[:index])	It is used to report the UUID of active beacon Optional <index> settings: 0…n The indexed tag during scanning.
&(BLE.Major[:index])	It is used to report the Major of active beacon Optional <index> settings: 0…n The indexed tag during scanning.
&(BLE.Minor[:index])	It is used to report the Minor of active beacon Optional <index> settings: 0…n The indexed tag during scanning.
&(BLE.relName[:index])	It is used to report the name of released beacon Optional <index> settings: 0…n The indexed tag during scanning.
&(BLE.relMAC[:index])	It is used to report the MAC of released beacon Optional <index> settings: 0…n The indexed tag during scanning.

&(BLE.relUUID[:index])	It is used to report the UUID of released beacon Optional <index> settings: 0…n The indexed tag during scanning.
&(BLE.relMajor[:index])	It is used to report the Major of released beacon Optional <index> settings: 0…n The indexed tag during scanning.
&(BLE.relMinor[:index])	It is used to report the Minor of released beacon Optional <index> settings: 0…n The indexed tag during scanning.

NEW FEATURES FOR FOX3-3G-BID:

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

✓ None

NEW FEATURES FOR IOBOX-MINI:

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

✓ None

NEW FEATURES FOR NFC Reader:

This section presents what is new in the firmware release regarding the Near Field Communication (NFC) reader.

✓ Added new PFAL commands/configuration/event/states/dynamic variables

PFAL Commands	
SYS.NFC.play,"<tone><tone>....<tone>"	This command is used to turn on and play specific tones on the buzzer on the NFC reader. Following settings are available: <tone>: <note>|<pause>[<shift>][<oktave>][/[<duration>[.][<art>] <note> - <pause> - _ <shift> - #|- <oktave> - 0|1|2|3|4|5|6|7|8 <duration> - 1|2|4|8|16 <art> - l|s|ss # - increase - - decrease (b) . - dotted duration l - legato: tone duration until the next tone without pause s - staccato: shortened tone duration ss - staccatissiomo: strongly shortened tone duration Valid tones C0 to C8 Standard octave = 3 Standard duration 1/4 non legato Example: $PFAL,SYS.NFC.play,"cdefgahc4" $PFAL,SYS.NFC.play,"c4 d4 e4 f4 g4 a4 h4 c5" $PFAL,SYS.NFC.play,"c4,d4,e4,f4,g4,a4,h4,c5" $PFAL,SYS.NFC.play,"c4/8 d4/8 e4/8 f4/8 g4/8 a4/8 h4/8 c5/8"

	$PFAL,SYS.NFC.play,"c4/8ss d4/8ss e4/8ss f4/8ss g4/8ss a4/8ss h4/8ss c5/8ss" $PFAL,SYS.NFC.play,"c5d5e5f5g5a5h5c6" $PFAL,SYS.NFC.play,"a4/8l f#5/l a4/8l d5/l a4/8l f#5/l a4/8l d5/l" $PFAL,SYS.NFC.play,"a4/16l d5/4.l a4/16l d5/16l a4/16l d5/16l a4/16l d5/4.l"

NEW FEATURES FOR BOLERO40 Series:

This section presents what is new in the firmware release regarding the BOLERO40.

✓ None

NEW FEATURES FOR IOBOX-CAN:

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

✓ Added new events for IOBOX-CAN

PFAL Commands	
Sys.CanB.DTCO.FMS.Enable	Enable tachograph communication on the 2nd CAN port (IOBOX-CAN)
Sys.CanB.DTCO.FMS.Disable	Disable tachograph communication on the 2nd CAN port (IOBOX-CAN)
Sys.CanB.Timeout,<timeout_s>	CAN timeout in s for the 2nd CAN port (IOBOX-CAN)
EVENTS	
CanB.eStat=<mode>	This event is occurred when the 2nd CAN bus (on the IOBOX-CAN) goes idle or active accordingly. <mode> settings: idle - CAN bus is in idle mode (CAN stops sending data for the user specified time out). active - CAN bus is in active mode(CAN starts sending data for the user specified time out) Example: $pfal,config.set,AL10=Sys.CanB.estat=idle:Msg.Send.Serial0,0,"CAN idle" $pfal,config.set,AL11=Sys.CanB.estat=active:Msg.Send.Serial0,0,"CAN active" $pfal,Sys.CanB.Timeout,10

NEW FEATURES FOR IOBOX-WLAN:

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

✓ Added new PFAL commands/configuration/event/states/dynamic variables.

PFAL Commands	
WLAN.Connect,<id>	Connects to the wireless access point specified in the profile id. Up to 5 profiles can be defined. <id> = Ranges from 0 to 4
Configuration parameters	
WLAN.NET<id>.SSID=<ssid>	Defines the SSID of the network (max. 32 characters) to connect to the WLAN access point. <id> Ranges from 0 to 4 <ssid> SSID of the access points with a max. of 32 characters.

WLAN.NET<id>.TYPE=dhcp	Defines the type DHCP of connection to the WLAN access point. <id> = Ranges from 0 to 4
WLAN.NET<id>.TYPE=static,<own_ip>,<netmask>,<gatewayIP>,<dnsIP>	Defines the settings for connecting to the wireless access point with a static IP address. <id> Ranges from 0 to 4 <own_ip> Specifies the static IP address set on that wireless access point <netmask> Specifies the network mask of that wireless access point <gatewayIP> Specifies the gateway IP of that wireless access point <dnsIP> Specifies the network IP address of that DNS wireless access point
WLAN.NET<id>.PSK=<psk>	Specifies the (PSK) Pre-Shared-Key (max. 63 characters) of that wireless access point <id> Ranges from 0 to 4 <psk> Sets the pre-shared key (PSK) of the WLAN network
WLAN.NET<id>.IP=<ip>	Specifies the IP address of the remote server to connect to after establishing a WLAN connection. <id> Ranges from 0 to 4 <ip> IP address in dotted-four-byte ("xxx.xxx.xxx.xxx") format
WLAN.NET<id>.PORT=<port>	Specifies the Port number of the remote server to connect to after establishing a WLAN connection. <id> Ranges from 0 to 4 <port> Specifies the port number that is used to send the data
WLAN.NET<id>.SECURITY=<type>	Specifies the security type of connection for that wireless access point. <id> = Ranges from 0 to 4 <type> settings WEP Wired Equivalent Privacy. WPA Wireless Protected Access. WPA2 Second version of the WPA standard. WPA2_MIX WPA2 mixed mode. NONE No security.
WLAN.MODE=<mode>	Defines the connection mode to the WLAN network. <mode> = Ranges from 0 to 4 <mode> settings: off Automatic connection mode is turned off. Scan Automatic connection mode after the scan. Connect<id> Automatic connection mode with profile <id>.
WLAN.RSSIMIN	Defines the minimum RSSI (Received Signal Strength Indication) level to be able to reliably connect to that wireless access points
Events/States	
SYS.WLAN.TCP.ePingSent	This event is occurred when a ping is sent to connected TCP server over the wireless access point.

NEW PREMIUM-FEATURES

This is a list of new PREMIUM-FEATURES in this firmware release.

- ✓ LUA
- ✓ CAN_CPC

NEW FEATURES FOR LUA:

This is a list of new PREMIUM-FEATURES in this firmware release.

✓ PFAL & LUA commands/Constants/event/states. An application note on how to get started with Lua is in preparation.

PFAL commands	
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	

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_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

ALARM_SYS_SERIALDATA1	data on the serial port 0, 1 accordingly
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 simcard is not longer present
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 dialled
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_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

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([flush])
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 in the firmware release regarding the Premium-Feature CAN_CPC.

✓ Added PFAL commands/configuration/event/states/dynamic variables.

Events	Description
SYS.eJ1939.TIRE_FAULT<comp><value>	This event is generated whenever the contents of the J1939.TIRE_FAULT variable changes. *Example: SYS.eJ1939.TIRE_FAULT=0 or SYS.eJ1939.TIRE_FAULT>0*
SYS.eJ1939.TIRE_STAT<comp><value>	This event is generated whenever the contents of the SYS.eJ1939.TIRE_STAT changes. *Example: SYS.eJ1939.TIRE_STAT=0 or SYS.eJ1939.TIRE_STAT>0*
SYS.eJ1939.TIRE_CPC_STAT_HEALTH<comp><value>	This event is generated whenever the contents of the SYS.eJ1939.TIRE_CPC_STAT_HEALTH variable changes. *Example: SYS.eJ1939.TIRE_HEALTH>=0*
SYS.eJ1939.TIRE_CPC_STAT_WEX<comp><value>	This event is generated whenever the contents of the SYS.eJ1939.TIRE_CPC_STAT_WEX variable changes. *Example: SYS.eJ1939.TIRE_CPC_STAT_WEX>=0*
SYS.eJ1939.TIRE_CPC_STAT_LEARN<comp><value>	This event is generated whenever the contents of the SYS.eJ1939.TIRE_CPC_STAT_LEARN variable changes. *Example: SYS.eJ1939.TIRE_CPC_STAT_LEARN>=0*
SYS.eJ1939.TIRE_CPC_TTM_STATE<comp><value>	This event is generated whenever the contents of the SYS.eJ1939.TIRE_CPC_TTM_STATE variable changes. *Example: SYS.eJ1939.TIRE_CPC_TTM_STATE>=0*
SYS.eJ1939.TIRE_CPC_TTM_ALARM<comp><value>	This event is generated whenever the contents of the SYS.eJ1939.TIRE_CPC_TTM_ALARM variable changes. *Example: SYS.eJ1939.TIRE_CPC_TTM_ALARM>=0*
SYS.eJ1939.TIRE_CPC_TTM_BAT<comp><value>	This event is generated whenever the contents of the SYS.eJ1939.TIRE_CPC_TTM_BAT variable changes. *Example: SYS.eJ1939.TIRE_CPC_TTM_BAT>=0*
SYS.eJ1939.TIRE_CPC_TTM_DEFECT<comp><value>	This event is generated whenever the contents of the SYS.eJ1939.TIRE_CPC_TTM_DEFECT variable changes. *Example: SYS.eJ1939.TIRE_CPC_TTM_DEFECT>=0*
SYS.eJ1939.TIRE_CPC_TTM_LOSE<comp><value>	This event is generated whenever the contents of the SYS.eJ1939.TIRE_CPC_TTM_LOSE variable changes. *Example: SYS.eJ1939.TIRE_CPC_TTM_LOSE>=0*

	<comp>	<comp> Compares the contents of this variable with the used specified one and return a Boolean (True/False) representing the result of the comparison. It can be set to =, !=, <,>, <= or>=.
	<value>	<value> Specifies the value to be compared.
Dynamic Variables		
Tire Condition		
J1939.TIRE_TEMP ≙ PGN 0xFEF433		It reports, into the comma separated value format, the temperature of each wheel as follows. e.g. 00:21,01:22,10:20,11:20,12:20,13:21,20:21,21:22 00 = wheel position, 21 = temperature in °C.
J1939.TIRE2_NOMPRESS ≙ PGN 0xFEF433		It reports, into the comma separated value format, the nominative pressure in kPa of each tire as follows. e.g.: 00:790,01:795,10:790,11:795,12:790,13:795,13:790,13:795 00 = wheel position, 790 = nominative tire pressure in kPa.
J1939.TIRE2_PRESSURE ≙ PGN 0xFEF433		It reports, into the comma separated value format, the pressure of each tire as follows. e.g.: 00:4,01:4,10:249,11:4,12:4,13:4,20:249,21:4 10:790 - 00 = wheel position, 249 = tire pressure in kPa.
J1939.TIRE_PRESSURE ≙ PGN 0xFEF433		It reports, into the comma separated value format, the pressure of each tire as follows. e.g.: 00:4,01:4,10:248,11:4,12:4,13:4,20:248,21:4 10 = wheel position, 248 = tire pressure in kPa.
J1939.TIRE_FAULT ≙ PGN 0xFEF433		It reports, into the comma separated value format, the fault state of each tire as follows. e.g.: 00:0,01:0,10:0,11:1, 12:0,13:0,20:0,121:0 11 = wheel position, 0 = no fault; 1 = fault.
J1939.TIRE_PTD ≙ PGN 0xFEF433		It reports, into the comma separated value format, the pressure threshold detection of each tire as follows. e.g.: 00:4,01:2,10:2,11:2,12:4,13:2,20:2,21:2 00 = wheel position; 1 = Over pressure, 2 = No warning pressure, 3 = Under pressure, 4 = Extreme under pressure.
J1939.TIRE_PLOSS ≙ PGN 0xFEF433		It reports, into the comma separated value format, the pressure loss of each tire as follows. e.g.: 00:0,01:0,10:0,11:0,12:0,13:0,20:0,21:0 00 = wheel position; 1 = tire pressure loss in Pa/s (0.1 Pa/s per bit).
J1939.TIRE_SEN ≙ PNG 0xFEF433		It reports, into the comma separated value format, if the sensors are enabled as follows. e.g.: 00:1,01:1,10:1,11:1,12:1,13:1,20:1,21:1 00:1 - 00 = wheel position; 0 = Sensor disabled, 1 = Sensor enabled.
J1939.TIRE_STAT ≙ PGN 0xFEF433		It reports, into the comma separated value format, the status of each tire as follows. e.g.: 00:0,01:0,10:0,11:0,00:0,01:0,10:0,11:0 00 = wheel position; 0 = tire OK, 1 = tire leak.
J1939.TIRE_ETP ≙ PGN 0xFEF433		It reports, into the comma separated value format, the while position and if the extended tire pressure is supported as follows. e.g.: 00:1,01:1,10:1,11:1,20:1,21:1,30:1,31:1 00 = wheel position; 0 = Not using Extended Tire Pressure, 1 = Using Extended Tire Pressure, 10: Error, 11: Not available/Not supported
CPC System Configuration		
J1939.TIRE_CPC_CNF_NAXLE ≙ PGN 0xFF0033		It reports the number of axles for the tractor or trailer as follows. e.g.: 00:4 00 = tractor, 01 = trailer; 4 = Number of axles.
J1939.TIRE_CPC_CNF_NCTTM ≙ PGN 0xFF0033		It reports the number of TTMs for the tractor and trailer as follows. e.g.: 00:8 00 = tractor, 01 = trailer; 08 = Number of TTMs.

CPC System Status	
J1939.TIRE_CPC_STAT_HEALTH ≙ PGN 0xFF0133	It reports the status of CPC system for the tractor and trailer as follows. e.g.: 00:1 00 = tractor, 01 = trailer; 0 = OK; "No TTM mounted" NOT detected, 0 = "No TTM mounted" detected.
J1939.TIRE_CPC_NOTTM ≙ PGN 0xFF0133	It reports if no TTMs mounted for the tractor and trailer as follows. e.g.: 00:0 00 = tractor, 01 = trailer; 0 = OK "No TTM mounted" NOT detected, 0 = "No TTM mounted" detected.
J1939.TIRE_CPC_STAT_WEX ≙ PGN 0xFF0133	It reports, into the comma separated value format, the single wheel exchange as follows. e.g.: 00:n/a 00 = tractor, 01 = trailer; 0 = "Single wheel exchanged" not detected, 0 = "Single wheel exchanged" detected.
J1939.TIRE_CPC_STAT_LEARN ≙ PGN 0xFF0133	It reports, into the comma separated value format, the automatic trailer learning as follows. e.g.: 00:0 00 = tractor; 0: Automatic trailer learning ongoing; 1: Automatic trailer learning finished, known trailer found, 2: Automatic trailer learning finished, new trailer found, 3: Automatic trailer learning finished, no trailer found, 4: Feature not active.
CPC TTM Data	
J1939.TIRE_CPC_TTM_PRESSURE ≙ PGN 0xFF0233	It reports, into the comma separated value format, the TTM pressure 4.706 kPa/bit as follows. e.g.: 00:aa,01:0,02:0,03:0,04:0,05:0,06:0,07:0,08:0,09:0 00…1F = index range in hex used to identify a sensor id; 0 = Sensor defective or data not available; 01…FF = (aa -1) * 4.706 kPa/bit = 800 kPa, FF=Overflow.
J1939.TIRE_CPC_TTM_TEMP ≙ PGN 0xFF0233	It reports, into the comma separated value format, the TTM temperature 1 °C/bit -50 K offset as follows. e.g.: 00:22,01:22,02:22,03:22,04:22,05:22,06:22,07:22,08:22,09:22 00…1F = index range in hex used to identify a sensor id; 0 = Sensor defective or data not available; 01…FF = (4F -50K) = 45 °C, FF=Overflow.
J1939.TIRE_CPC_TTM_STATE ≙ PGN 0xFF0233	It reports, into the comma separated value format, the TTM state as follows. e.g.: 00:8,01:8,02:8,03:8,04:8,05:8,06:8,07:8,08:8,09:8 00…1F = index range in hex used to identify a sensor id; 00-FE = Don't care, FF = No TTM data since Power On
J1939.TIRE_CPC_TTM_ALARM ≙ PGN 0xFF0233	It reports, into the comma separated value format, the alarm and warning as follows. e.g.: 00:2,01:2,02:2,03:2,04:2,05:2,06:2,07:2,08:2,09:2 00…1F = index range in hex used to identify a sensor id; 0: OK, 1: Under-inflation warning, 2: Under-inflation alarm, 3: Tire leak alarm, 4: TTM mute, 5: Temperature warning, 8: TTM over temperature warning
J1939.TIRE_CPC_TTM_BAT ≙ PGN 0xFF0233	It reports the battery flag as follows. e.g.: n/a n/a = Battery flag.
J1939.TIRE_CPC_TTM_DEFECT ≙ PGN 0xFF0233	It reports the TTM defective as follows. e.g.: n/a 00 = wheel position; n/a = TTM defective.
J1939.TIRE_CPC_TTM_LOSE ≙ PGN 0xFF0233	It reports, into the comma separated value format, the loose TTM detection as follows. e.g.: 00:0,01:0,02:0,03:0,04:0,05:0,06:0,07:0,08:0,09:0 00…1F = index range in hex used to identify a sensor id; 0: OK, 1: TTM

	loose, 2: TTM turned
CPC Graphical Position Configuration	
J1939.TIRE_CPC_POS:H<n> ≙ PGN 0xFF0433	It reports, into the comma separated value format, the value according to the matrix (graphical position and tire location) and the index used as a reference/conjunction for matching the tire location, graphical position and ID of the sensors in hexadecimal as follow s. e.g.: 00:03,01:0b,02:43,03:4b,04:53,05:5b 00…1F = index range in hex used to identify a sensor id; 0b = Hexadecimal value to identify the graphical position of the sensors. Refer to the pic. 3 and Table 5 below.
J1939.TIRE_CPC_POS ≙ PGN 0xFF0433	It reports, into the comma separated value format, the value according to the matrix (graphical position and tire location) and the index used as a reference/conjunction for matching the tire location, graphical position and ID of the sensors in decimal as follows. e.g.: 00:3,01:11,02:67,03:75,04:83,05:91 00…1F = index range in hex used to identify a sensor id; 0b = Decimal value to identify the graphical position of the sensors. Refer to the pic. 3 and Table 5 below.
J1939.TIRE_CPC_LOC:H<n> ≙ PGN 0xFF0433	It reports, into the comma separated value format, the value according to the matrix (graphical position and tire location) and the index used as a reference/conjunction for matching the tire location, graphical position and ID of the sensors in hexadecimal as follows. e.g.: 00:00,01:01,02:10,03:11,04:20,05:21 00…1F = index range in hex used to identify a sensor id; 11 = Hexadecimal value to identify the tire location. Refer to the pic. 3 and Table 5 below.
J1939.TIRE_CPC_LOC ≙ PGN 0xFF0433	It reports, into the comma separated value format, the value according to the matrix (graphical position and tire location) and the index used as a reference/conjunction for matching the tire location, graphical position and ID of the sensors in decimal as follows. e.g.: 00:0,01:1,02:16,03:17,04:32,05:33 00…1F = index range in hex used to identify a sensor id; 17 = decimal value to identify the graphical position of the sensors. Refer to the pic. 3 and Table 5 below.
J1939.TIRE_CPC_TTM_ID ≙ PGN 0xFF0433	It reports, into the comma separated value format, the while position and the TTM ID of the sensor in decimal as follows. e.g.: 00:1835297152,01:1835297152,02:1821695488,03:1818171136,04:1825507328,05:1825000448,06:1821695360,07:1821695232 00…1F = index range in hex used to identify a sensor id; 1835297152=TTM ID in decimal value. Refer to the pic. 3 and Table 5 below.
J1939.TIRE_CPC_TTM_ID:H<n> ≙ PGN 0xFF0433	It reports, into the comma separated value format, the while position and the TTM ID of the sensor in <n> decimal digits as follows. e.g.: J1939.TIRE_CPC_TTM_ID:H8 00:6d646950,01:6d646948,02:6c94de3c,03:6c5f16e6,04:6ccf0811,05:6cc74bf2,06:6c94dd4e,07:6c94dd08 00…1F = index range in hex used to identify a sensor id; 6d646950=TTM ID in hexadecimal value. Refer to the pic. 3 and Table 5 below.
J1939.TIRE_CPC_X ≙ PGN 0xFF0433	It reports, into the carriage-return and line-feed, the collection of all CPC messages as follows. e.g.: J1939.TIRE_CPC_CNF_NAXLE:00:4 J1939.TIRE_CPC_CNF_NCTTM:00:8 J1939.TIRE_CPC_STAT_HEALTH:00:1 J1939.TIRE_CPC_STAT_WEX:n/a J1939.TIRE_CPC_STAT_LEARN:00:0 J1939.TIRE_CPC_TTM_PRESSURE:00:aa,01:0,02:0,03:0,04:0,05:0,06:0,

	07:0,08:0,09:0 J1939.TIRE_CPC_TTM_TEMP:00:22,01:22,02:22,03:22,04:22,05:22,06:22,07:22,08:22,09:22 J1939.TIRE_CPC_TTM_STATE:00:8,01:8,02:8,03:8,04:8,05:8,06:8,07:8,08:8,09:8 J1939.TIRE_CPC_TTM_ALARM:00:2,01:2,02:2,03:2,04:2,05:2,06:2,07:2,08:2,09:2 J1939.TIRE_CPC_TTM_BAT:n/a J1939.TIRE_CPC_TTM_DEFECT:n/a J1939.TIRE_CPC_TTM_LOSE:00:0,01:0,02:0,03:0,04:0,05:0,06:0,07:0,08:0,09:0 J1939.TIRE_CPC_POS:00:3,04:11,08:19,0c:27,10:67,14:75,18:83,1c:91 J1939.TIRE_CPC_LOC:00:0,04:1,08:16,0c:17,10:32,14:33,18:48,1c:49 J1939.TIRE_CPC_TTM_ID:00:1835297152,01:1835297152,02:1821695488,03:1818171136,04:1825507328,05:1825000448,06:1821695360,07:1821695232

TTM=Truck tire module "Sensor inside the tire"

REMOVED/NOT SUPPORTED:

This section presents what has been removed/not supported in this firmware release.

✓ SYS.BLE.eRegister=whitelist

CHANGES:

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

✓ Changed PFAL commands/configuration/event/states/dynamic variables

PFAL Commands	
Sys.Can.OBDII.Enable[,<format>]	Enables the OBD-II [and the format] for reading its messages on 1st CAN interface (on main port) Optional <format> settings: Std Enables the 11-bit identifier (CAN2.0A) Ext Enables the 29-bit identifier (CAN2.0B)
Sys.CanB.OBDII.Enable[,<format>]	Enables the OBD-II [and the format] for reading its messages on 2nd CAN interface (IOBOX-CAN) Optional <format> settings: Std Enables the 11-bit identifier (CAN2.0A) Ext Enables the 29-bit identifier (CAN2.0B)
MSG.Mode.<interface>=<output>,D=<msg_output_channel>	Changed the transparent data mode between channels: Supported <msg_output_channel> settings: 1 1st Serial port. 2 2nd Serial port. 4 USB port 10 TCP port.
Configuration parameters	
DEVICE.CAN.OBD.STARTUP=<format>,<port>	Enables the OBD-II [and the format] for reading its messages on the user specified CAN port <format> settings:

	Std Enables the 11-bit identifier (CAN2.0A) Ext Enables the 29-bit identifier (CAN2.0B) <port> 0 Enables reading on 1st CAN interface (on main port). 1 Enables reading on 2nd CANB interface (on IOBOX-CAN)

IMPROVEMENTS:

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

- ✓ AES key handling (Support AES128/192/256 keys).
- ✓ Added LUA support for flash filesystem.
- ✓ Read/Write data from I2C devices.
- ✓ BIOS 3.0.2: Optimized for BOLERO40 series.
- ✓ Improvements of the LUA net socket class.
- ✓ Double buffer handling in OBD2 and ISO-TP.
- ✓ Increase the “CAN message filter” on the main interface and IOBOX-CAN.

BUGS FIXED:

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

- ✓ The setting $PFAL,CNF.Set,DEVICE.PFAL.SEND.FORMAT=<...>.
- ✓ The event GSM.Voicecall.eIncoming" will be generated when a voice call comes in.
- ✓ Fixed flag Enable/Disable in the Dead-Reckoning function for FOX3-3G-DR.
- ✓ Fixed wrong hysteresis "Sys.Device.sleep=AiWu=<min>,<max>".
- ✓ Changed default for TCP.CLIENT.LOGIN=<parameter>.

KNOWN ISSUES:

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

Mantis ID	Priority	Status	Bug Summary
-	-	-	RFID-A-B-EXT & AVL device. When the AVL device goes into the doze low power mode, a Tag should be double passed within the reader magnetic field to detect the ID of that Tag on the AVL serial port. The first swipe is to wake up the AVL device from the doze mode and the second swipe is to detect the Tag and generate the corresponding event.
-	-	-	

References

This is a list of references used for updating this firmware release into one of the LANTRONIX AVL devices. The current version of the documents/software can be directly downloaded from the

LANTRONIX website:

Workbench Tool

https://www.lantronix.com/products/workbench/#tab-docs-downloads

WebUpdate

https://cdn.lantronix.com/wp-content/uploads/pdf/AppNote_WebUpdate_Howto.pdf

Remote Update with Workbench

https://cdn.lantronix.com/wp-content/uploads/pdf/AppNotes_Remote_Update_With_Workbench.pdf