LANTRONIX®

Application Note:

Using Lua Scripts for FOX Series and BOLERO40 Series

Part Number APP-0117
Revision F September 2024

Intellectual Property

Contacts

Lantronix, Inc.
48 Discovery, Suite 250
Irvine, CA 92618, USA

Toll Free: 800-526-8766
Phone: 949-453-3990
Fax: 949-453-3995

Technical Support
Online: www.lantronix.com/technical-support

Sales Offices
For a current list of our domestic and international sales offices, go to the Lantronix web site at www.lantronix.com/about/contact

Disclaimer

Revision History

Date	Rev.	Comments
September 2021	A	Initial document
May 2023	B	Updated to Firmware Release AVL_3.16.0_rc9, which includes the following: Added Lua functions dofile() and loadfile().

		Updated to Firmware Release AVL_3.17.0_rc5, which includes the following:
		Added Modbus commands,
		modbus_register reg :=[ ]
		res = avl.modbus_query()
		t := avl.modbus_data([t])
		t, addr := avl.modbus_register("<slave>:<LE|BE>,<reg>:<fmt>");

July 2023	C	Corrected syntax of commands for Lua Start, Lua Stop and Lua Dump
November 2023	D	Updated to Firmware Release AVL_3.20.0.0, which includes the following: - Added Lua Events for Percepxion - Added Lua States for Percepxion
May 2024	E	Replaced mention of FOX3-2G/3G/4G series with FOX series
September 2024	F	Updated make_script.sh to include .zip and .gz output formats Added details about using the make_script.sh file to convert the .lua file into .frp, .zip, and .gz archive files

For the latest revision of this product document, please check our online documentation at www.lantronix.com/support/documentation.

Contents

Deploying Lua Scripts

This Application note describes how to deploy Lua scripts on FOX series and BOLERO40 series devices. It shows how to load a Lua file onto a device using Lantronix Workbench and run a script.

This document assumes that you have the prerequisite hardware and software tools installed and configured for use and know how to configure and execute PFAL commands on the FOX series and BOLERO40 series devices. The example in this document uses a Windows 10/11, 64-bit environment, but you can also use Linux or Mac OS.

Prerequisites

You will need the following tools to deploy Lua script:

- The Lantronix FOX series or BOLERO40 series Promotion Kit with AVL firmware version 3.2 or greater
- Lantronix Workbench software
- PC Windows, Linux, or MacOS computer
- IDE with full support for Lua
- Lua, version 5.2.4 or greater (https://www.lua.org/download.html)
- Bash and zip software

Development Setup

As part of development setup, install the following components:

- Install a bash and zip
    - On Windows, you can use Cygwin (https://www.cygwin.com/) or WSL (https://docs.microsoft.com/en-us/windows/wsl/install-win10)
    - On Linux it is built-in or you can add via 'apt-get install bash unzip'.
    - On MacOS, it is built-in.
- Install Lantronix Workbench software on your PC.
    - https://www.lantronix.com/products/workbench/
- Install the IDE of your choice, preferably with built-in Lua Highlight and/or CodeCheck support. Lantronix recommends:
    - IntelliJ (https://www.jetbrains.com/idea/) CE Edition is free for use.
        - Download and Install IntelliJ.
        - Start IntelliJ.
        - Go to File/Settings/Plugins -> Browse repositories -> Type "LUA"
        - Select "Lua language integration for IntelliJ" click install
        - Checkout this repository File/new/Project from Version Control/git
    - Eclipse (https://www.eclipse.org)
        - Download the latest stable version
        - Install Eclipse locally. To install, go to Help > Eclipse Marketplace type "LUA."
        - Install "Lua Development Tools. To install, go to Help > Eclipse Marketplace type "TM Terminal."
        - Install "TM Terminal 4.0"
        - Checkout this repository file/new/project from Version control/git

Tracker/Hardware Setup

To install and set up the tracker, install the following components first:

- Set up tracker/promotion kit.
- Connect tracker via USB or serial to your PC (can be done via TCP).

Activate Lua Premium Feature

To activate the Lua premium feature, see the application note, "Activation of Premium-Features".

To verify which premium features are active, execute the PFAL command:

```
$PFAL,msg.feature
```

Example output:

```
$<MSG.Feature>
$IndexedHistory: inactive
$AES_TCP: inactive
$LUA: active (never expires)
$<end>
```

Using Lua Scripts

Lua Sample Scripts

The example in this application note uses the following sample script files:

- `make_script.sh`
- `averagetemp.lua`

Description

- `make_script.sh` converts the Lua file into the following files:
  - .frp archive file (only for uploads via serial line)
  - .zip archive file (only for uploads via Percepxion)
  - .gz archive file (only for uploads via HTTP/S)
- `averagetemp.lua` measures the internal temperature of the tracker and outputs the average temperature every 10 seconds.

These files are provided in *Appendix: Sample Scripts*.

To use the sample files in your own test, copy the script file content into a plain text editor and save with the appropriate file extension.

Load Lua on Device

Before you deploy a Lua file:

1. Set up the tracker and development environment.
2. Make sure the Lua premium feature is active.

Deploy a Lua File on Device

This example demonstrates how to deploy a Lua script to the tracker using the sample script files.

To deploy a Lua file on a device:

1. Copy `make_script.sh` and `averagetemp.lua` to the cygwin home directory. Both files should be in the same directory. On Linux and Mac, use bash and zip to perform the task.
2. Using cygwin, call `make_script.sh`.

   ```
   $ ./make_script.sh averagetemp.lua
   ```

   - This converts the averagetemp.lua file into the following archive files:`averagetemp.frp` (only for uploads via serial line)
   - `averagetemp.zip` (only for updates via Percepxion and HTTP/S)
   - `averagetemp.gz` (only for updates via HTTP/S)

3. Use Lantronix Workbench to connect to the tracker and upload `averagetemp.frp` to the tracker.

   Note:

   - *To deploy the .zip archive file use Percepxion (for instructions refer to Percepxion online help) or perform WebUpdate (for instructions refer to PFAL Command Reference)*
   - *To deploy the .gz archive file perform WebUpdate*

4. Run the file loaded on the device. To run it manually, execute the following PFAL commands:

   ```
   $PFAL,SYS.LUA.Start
   ```

   This command starts the Lua script.

   ```
   $PFAL,SYS.LUA.Stop
   ```

   This command stops the Lua script.

   ```
   $PFAL,SYS.LUA.Dump
   ```

   This command lists the Lua script source code.

   Note: To automate starting Lua when the device starts, add the following command to the startup configuration:

   ```
   $PFAL,CNF.Set,AL0=SYS.DEVICE.eSTART:SYS.LUA.Start
   ```

5. View average temps being reported and displayed in the Lantronix Workbench window.

   Note: *The sample Lua script writes the average temps to the serial channel as defined in the following line of code, but it can be defined in the script file to send it to the TCP server or to other channel.*

   ```
   avl.pfal(string.format("MSG.Send.Rawserial0,0,\"Average Temperature Is
   %s\r\n\"",ave))
   ```

Activate Debug Output

Activate debug output to find errors in code.

Use the corresponding PFAL command to activate the debug output on the preferred channel.

- To activate the debug output of the serial0 interface on the 8-pin connector, use

  ```
  $PFAL,CNF.Set,DBG.EN=1 or
  ```

```
$PFAL,CNF.Set,DBG.EN=1,serial0
```

- To activate the debug output of the serial1 interface on the 6-pin connector, use

```
$PFAL,CNF.Set,DBG.EN=1,Serial1
```

- To activate the debug output of the USB interface, use

```
$PFAL,CNF.Set,DBG.EN=1,USB
```

- To disable the debug output, use

```
$PFAL,CNF.Set,DBG.EN=0,<interface>
<interface>: 0, 1, USB
```

Reference

The following tables list commands, events, and states that you can reference in Lua scripts as additional features in the FOX3 and BOLERO 40 series devices once the Lua Premium feature is activated.

Lua Commands

PFAL commands	
SYS.Lua.Start[,<"script.lua">]	Loads and starts a specific Lua script
SYS.Lua.Clear[,<"script.lua">]	Deletes a specific Lua script
SYS.Lua.Info[,<"script.lua">]	Comment of a specific Lua script
SYS.Lua.Write[,<"script.lua">]	Writes a specific Lua script
SYS.Lua.Start	Starts the Lua script loaded into 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 into 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.Event,<id>,<"text">	Generates custom events for the Lua.
LUA Commands / PFAL command request	
os.sleep(millies)	Suspends the execution of the current thread until the time-out interval in milliseconds elapses.
os.trace("format", args)	It outputs the “args” information if debug “DBG.EN=1” is enabled.
avl.useevent(type[,OnOff])	Unmask/Mask LUA events/constant types
count := avl.i2c_read(addr, register, data)	Read data from I2C devices

count := avl.i2c_write(addr, register, data)	Write data from I2C devices
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 Modbus Commands	

modbus_register reg := [
"valid"
"value"
"format"
"word0"
"word1"
"word2"
"word3"
]
	The polled Modbus register data.
	// validity flag
	// value of the register
	// printed register value
	// register word 0
	// register word 1
	// register word 2
	// register word 3

res = avl.modbus_query()	Query non-periodically ModBus devices.
t := avl.modbus_data([t])	Get the polled ModBus register values.
t, addr := avl.modbus_register("<slave>:<LE|BE>,<reg>:<fmt>");	Read a ModBus device register.
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;`

Lua Events

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	

ALARM_SYS_SERIALDATA1	This event is created whenever the device detects incoming 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_BLE_SCANEND	This event is created once the FOX3-3G-BLE has ended a scan session for BLE sensors
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_BLEDATA	This event is created whenever the device receives data from a BLE slave during a BLE connection.
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 FOX device
ALARM_SYS_1WIRE_RELEASE	This event is created whenever a 1-Wire device is released from the 1-Wire interface of the FOX device
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_WLAN_CONNECTING	This event is created when the WLAN module is trying to connect to one of 5 wireless access points

ALARM_SYS_WLAN_CONNECTED	This event is created once the WLAN module is connected to one of 5 wireless access points
ALARM_SYS_WLAN_DISCONNECTED	This event is created once the WLAN module is disconnected from one of 5 wireless access points
ALARM_SYS_WLAN_RECEIVED	This event is created whenever the WLAN module 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_HISTORY_TAUT	Not supported (Event used in GPS history download)
ALARM_GPS_HISTORY_PUSH_FINISH	Not supported (Event used in GPS history download)
ALARM_GPS_JAMMING	This event is called 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 no 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
LARM_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 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 stars 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 packet 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	This event occurs when a CANopen PDO event is 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
ALARM_MQTT_CLIENT_CONNECTING	This event is created when device starts connecting to a MQTT server
ALARM_MQTT_CLIENT_CONNECTED	This event is created when device is connected to the MQTT server
ALARM_MQTT_CLIENT_PACKETSENT	This event is created when a TCP packet is sent
ALARM_MQTT_CLIENT_PINGSENT	This event is created when a TCP ping is sent
ALARM_MQTT_CLIENT_DISCONNECTING	This event is created when device starts disconnecting from the MQTT server
ALARM_MQTT_CLIENT_DISCONNECTED	This event is created when device is disconnected from the MQTT server
ALARM_MQTT_CLIENT_FLASHBUFFER_EMPTY	This event is created once the flash buffer is emptied
ALARM_MQTT_CLIENT_BUFFER_EMPTY	This event is created once the message buffer is emptied
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	
CAN	
ALARM_SYS_CANMSG	This event is created when contents of this CAN message is changed
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
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
PERCEPXION	
ALARM_PX_CLIENT_STARTED	This event is created when PX MQTT client is started.
ALARM_PX_CLIENT_STOPPED	This event is created when PX MQTT client is stopped.
ALARM_PX_CLIENT_CAP_NEG_STARTED	This event is created when PX client starts capability negotiation.
ALARM_PX_CLIENT_CAP_NEG_COMPLETED	This event is created when PX client completes capability negotiation.
ALARM_PX_CLIENT_MQTT_RECEIVED	This event is created when PX MQTT client gets a subscription.
ALARM_PX_CLIENT_MQTT_CONNECTED	This event is created when PX MQTT client is connected to the server.
ALARM_PX_CLIENT_MQTT_DISCONNECTED	This event is created when PX MQTT client is disconnected from the server.
ALARM_PX_CLIENT_REGISTERED	This event is created when PX client is registered on the server.
ALARM_PX_CLIENT_PUBLISHED	This event is created when PX client publishes telemetry data.
ALARM_PX_CLIENT_UPDATES_AVAILABLE	This event is created when PX client gets available updates.
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;`

Lua States

State Requests	

state := [
state.type
state.idx
state.u_bool
state.u_value
state.u_string
state.u_starttype
	Reads the type and the data assigned to that state
	// values of type “state.u_xxx” fields depending on the state type
	// integer state type
	// integer subindex
	// boolean value type
	// integer value type
	// string value type

state.u_startreason
state.u_opid
state.u_opname
]
	// 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_DISCONNECTING	Value of the PFAL TCP.Client2.sdisconnecting state
STATE_TCP_CLIENT2_DISCONNECTED	Value of the PFAL TCP.Client2.sDisconnected state
STATE_MQTT_CLIENT_IDLE	Value of the PFAL TCP.MQTT.sIdle state
STATE_MQTT_CLIENT_CONNECTING	Value of the PFAL TCP.MQTT.sConnecting state
STATE_MQTT_CLIENT_CONNECTED	Value of the PFAL TCP.MQTT.sConnected state
STATE_MQTT_CLIENT_DISCONNECTING	Value of the PFAL TCP.MQTT.sdisconnecting state
STATE_MQTT_CLIENT_DISCONNECTED	Value of the PFAL TCP.MQTT.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
PERCEPXION	
STATE_PX_CLIENT_STARTED	Value of PFAL PX.client.sstarted state
STATE_PX_CLIENT_STOPPED	Value of PFAL PX.client.sstopped state
STATE_PX_CLIENT_CAP_NEG_STARTED	Value of PFAL PX.client.cap.neg.sstarted state
STATE_PX_CLIENT_CAP_NEG_COMPLETED	Value of PFAL PX.client.cap.neg.scompleted state
STATE_PX_CLIENT_MQTT_RECEIVED	Value of PFAL PX.MQTT.sreceived state
STATE_PX_CLIENT_MQTT_CONNECTED	Value of PFAL PX.MQTT.sconnected state
STATE_PX_CLIENT_MQTT_DISCONNECTED	Value of PFAL PX.MQTT.sdisconnected sate
STATE_PX_CLIENT_REGISTERED	Value of PFAL PX.client.sregistered state
STATE_PX_CLIENT_PUBLISHED	Value of PFAL PX.client.spublished state
STATE_PX_CLIENT_UPDATES_AVAILABLE	Value of PFAL PX.client.updates.savailable state
PFAL file transfer	
len := avl.file_upload(buffer)	Reads the length of the file
Format string 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
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
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
Motion data	
tResult := avl.motion_data()	Reads the motion data
Filesystem access	
file := io.open(filename [, mode]	This function opens a file, in the mode specified in the string mode. It returns a new file handle, or, in case of errors, nil plus an error message. The mode string can be any of the following: "r": read mode (the default); "w": write mode; "a": append mode; "r+": update mode, all previous data is preserved; "w+": update mode, all previous data is erased; "a+": append update mode, previous data is preserved, writing is only allowed at the end of file. The mode string can also have a 'b' at the end, which is needed in some systems to open the file in binary mode.
io.lines (filename)	Opens the given file name in read mode and returns an iterator function that works like file:lines(···) over the opened file. When the iterator function detects the end of file, it returns nil (to finish the loop) and automatically closes the file. The call io.lines() (with no file name) is equivalent to io.input():lines(); that is, it iterates over the lines of the default input file. In this case it does not close the file when the loop ends. In case of errors this function raises the error, instead of returning an error code.
io.read(...)	Equivalent to file:read(). Without a file, reads from the default input file.
io.write(...)	Equivalent to file:write(). Without a file, writes to the default output file.
io.type(file)	Checks whether file is a valid file handle. Returns the string "file" if obj is an open file handle, "closed file" if obj is a closed file handle, or nil if obj is not a file handle.
io.flush(file)	Equivalent to file:flush(). Without a file, closes the default output file.

io.close(file)	Equivalent to file:close(). Without a file, closes the default output file.
file:read(···)	Reads the file file, according to the given formats, which specify what to read. For each format, the function returns a string (or a number) with the characters read, or nil if it cannot read data with the specified format. When called without formats, it uses a default format that reads the next line (see below). The available formats are: "*n": reads a number; this is the only format that returns a number instead of a string. "*a": reads the whole file, starting at the current position. On end of file, it returns the empty string. "*l": reads the next line skipping the end of line, returning nil on end of file. This is the default format. "*L": reads the next line keeping the end of line (if present), returning nil on end of file. number: reads a string with up to this number of bytes, returning nil on end of file. If number is zero, it reads nothing and returns an empty string, or nil on end of file.
file:write(···)	Writes the value of each of its arguments to file. The arguments must be strings or numbers. In case of success, this function returns file. Otherwise it returns nil plus a string describing the error.
file:lines()	Returns an iterator function that, each time it is called, reads the file according to the given formats. When no format is given, uses "*l" as a default. Unlike io.lines, this function does not close the file when the loop ends. In case of errors this function raises the error, instead of returning an error code.
file:flush()	Saves any written data to file.
file:close()	Closes file. Note that files are automatically closed when their handles are garbage collected, but that takes an unpredictable amount of time to happen.
file:seek([whence] [, offset])	Sets and gets the file position, measured from the beginning of the file, to the position given by offset plus a base specified by the string whence, as follows: "set": base is position 0 (beginning of the file); "cur": base is current position; "end": base is end of file; In case of success, seek returns the final file position, measured in bytes from the beginning of the file. If seek fails, it returns nil, plus a string describing the error. The default value for whence is "cur", and for offset is 0. Therefore, the call file:seek() returns the current file position, without changing it; the call file:seek("set") sets the position to the beginning of the file (and returns 0); and the call file:seek("end") sets the position to the end of the file, and returns its size.
dofile()	Executes a chunk of code stored in a file.
loadfile()	Loads a Lua chunk from a file, compiles the chunk and returns the compiled chunk as a function.

os.remove(name)	Remove the file given as "name".
os.rename(oldname, newname)	Rename file "oldname" to "newname".
os.mkdir(path)	Create the directory given as "path".
os.rmdir(path)	Remove the directory given as "path".
stat := os.stat(filename [, request|result])	Returns a table with file attributes corresponding to filename (or nil followed by an error message and a system-dependent error code in case of error). If the second optional argument is given and is a string, then only the value of the named attribute is returned (this use is equivalent to os.stat(file)[request]. But the table is not created and only one attribute is retrieved from the OS). If a table is passed as the second argument, it (result) is filled with attributes and returned instead of a new table. The attributes are described as follows; attribute mode is a string, all the others are numbers. dev, rdev - On Unix systems, this represents the device that the inode resides on. On Windows systems, represents the drive number of the disk containing the file. Ino - On Unix systems, this represents the inode number. On Windows systems this has no meaning mode. String - representing the associated protection mode (the values could be file, directory, or other). Nlink - Number of hard links to the file. Uid - User-id of owner (Unix only, always 0 on Windows) Gid - Group-id of owner (Unix only, always 0 on Windows) Access - Time of last access Modification - Time of last data modification Change - Time of last file status change Size - File size, in bytes Permissions - File permissions string
iter, dir_obj := os.dir (path)	Lua iterator over the entries of a given directory. Each time the iterator is called with dir_obj, it returns a directory entry's name as a string, or nil if there are no more entries. You can also iterate by calling dir_obj:next(), and explicitly close the directory before the iteration finished with dir_obj:close(). Raises an error if path is not a directory.

FS directory object dir := [
dir:next()
dir:close()
]
	
	// Next entry from directory
	// Close directory

Direct CAN access	
result := avl.can_write(chan, ext, id, data)	Writes a message to the corresponding CAN interface. Returns 1 if sending of the CAN message was successfully. chan: CAN interface [0,1] ext: message type std/ext [0,1] id: message id to send data: message data to send
result := avl.can_read([table])	Reads a message from CAN interface. Returns a table filled with a CAN message or Nil if no data is available. If a table is passed as argument, it is filled with message data (table) and returned instead of a new table. The attributes are described as follows; attribute data is a string, all others are numbers.

	ch: The CAN interface the message is read from [0,1] ext: The type of the message std/ext [0,1] msg: The id of the message size: The length of the message data: The message data (0..8 bytes)
Socket interface	

socket := net.create_socket([type, param])
socket:connect(<"IP"|"URL">, port)
socket:close([flush])
socket:flush()
socket:hold()
socket:unhold()
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())
	
	- unhold the socket
	- Set/Read ttl value
	- Set/Read buffer size
	- Send data to socket
	- Read data from socket

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
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
..
	Reads the GPS values listed within the [] square brackets.
	// Number of GPS satellites
	// Dump of satellite data
	// "SatID,Elevation,Azimuth,AvgCNo,Used"

gps_sat12
gls_num
gls_sat1
..
gls_sat12
]
	// Number of GLS satellites
	// Dump of satellite data
	// "SatID,Elevation,Azimuth,AvgCNo,Used"

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	

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
	l// Normal Y gravitation
	// Normal Z gravitation

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/
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()	

Appendix: Sample Scripts

averagetemp.lua

```
--
-- Created by IntelliJ IDEA.
-- User: username
-- Date: 25.01.19
-- Time: 09:44
-- To change this template use File | Settings | File Templates.
--script ro read temperature every 10 sec

timer1 = avl.tick(10000,  1000)
timer1:cyclic()

storage = {}

function event (e)
    -- local t = os.clock() or ...
    local t = e.time
    local type = e.type

    -- Possible user events
    if type >= ALARM_TYPE_INTERNAL then
        type = type - ALARM_TYPE_INTERNAL
        if type == timer1:id() then
            os.trace("ser event %d \"%s\" (%d ms)", type, e.u_string, t)

            os.trace(avl.format("Temperature is &(Temp)"))
            local currentTemp = tonumber(avl.format("&(Temp)"))
            table.insert(storage,currentTemp)
            printTableAvg(storage)
        end
    end
end

function printTableAvg (t)
    local elements = 0
    local sum = 0
    local ave = 0

    for k,v in pairs( t ) do
        sum = sum + v
        elements = elements + 1
    end
    ave = sum / elements
    os.trace("Average Temperature Is %.2f", ave)
     avl.pfal(string.format("MSG.Send.Rawserial0,0,\"Average Temperature Is
%s\r\n\"",ave))

end

while 1 do
    local ev = avl.event(10000)
   -- x = x + 1
```

```
    if (ev == nil) then
--        loop ()
    else
        event(ev)
    end
end
```

make_script.sh

```
#! /bin/sh
#
# Convert LUA scripts to frp archiv files
#
# @file make_script.sh 2017-05-12  @author fbeqiri
file=$1
file_dir="$(dirname "$file")"
echo $file_dir
script_dir="./"
echo $script_dir
cp "$file" "$script_dir"
#cd $file_dir
filename=${file%.*}
file=$(basename "$file")
echo $file
if [ ! -f $file ]; then
  echo "use ./make_script.sh <your script>.lua"
  exit 0
fi

echo "Converting [$file] file into [frp] and [zip] files..."

echo '<?xml version="1.0" encoding="UTF-8"?>
<falcom-resource-package xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
<version-info number="2"/>
     <resources>
     <agps> <file format="flat" size="0" crc="@md5sum">@script</file>
</agps>
</resources>
<devices>
     <device class="all" type="all">
           <module type="gps" option="ublox">
                 <resource type="agps">
                 <version>@version</version>
                 <file format="flat" crc="@md5sum">@script</file>
                 <descriptor firmwaresize="0" crc="null">null</descriptor>
                 </resource>
           </module>
     </device>
</devices>
</falcom-resource-package>' > content.xml
sed -i -e s/@script/$file/g -e s/@md5sum/`md5sum $file | cut -d ' ' -f 1`/g
content.xml
sed -i -e s/@script/$file/g -e s/@version/$file/g content.xml

final=$filename.frp
```

```
echo $final
if [ -f $final ]; then rm $final; fi
echo "1. Creating frp file [$final] ..."
echo "1. Creating frp file [$file] ..."
zip -9 $final $file content.xml
rm content.xml
mv $final $file_dir

# create webupdate file
final=$filename.zip
echo "2. Creating Lua webupdate file [$final] ..."
gzip -9 -kf $file
mv $file.gz $final
mv $final $file_dir

# create webupdate .gz file
final=$filename.gz
current_year=$(date +'%Y')
tfile="${file%.*}"
echo "3. Creating Lua webupdate .gz file [avl_$tfile-L$current_year.gz] ..."
gzip -9 -kf $file
mv $file.gz avl_$tfile-L$current_year.gz
mv avl_$tfile-L$current_year.gz $file_dir
rm $file
```