VIALATM SCRIPT V4

Table of Contents

Introduction

A script is a program that is executed on certain events (for example, when new messages are received from objects).

If a script is specified for an object, then it is executed, otherwise the script in the nearest parent group is executed.

The script consists of modules. The module block in the source code of the script begins with **.module** The presence of at least one of the modules determines the presence of a script for an object (or group). In case of errors during the execution of the script, they can be viewed in the script panel - the **Runtime error** button. In addition, a red gear icon is displayed for this object in the objects panel.

Restrictions:
The script does not allow the use of **for**, **forEach** and **while** statements.

.module Declare

This module defines the attributes available for display in tooltip and/or object variables that persist between invocations of the script program.

Integer EXTRABATPROC "Extra Bat. % charge"
Double EXTRABATVOLT

Basic definition format:

TYPE IDENT ["NAME"]

TYPE - defines the type of the attribute/variable. Valid types: **Integer**, **Double**, **String**.

IDENT - specifies an identifier. Only Latin letters, numbers and the _ symbol are allowed. The identifier must start with a Latin letter or the symbol _ . The maximum length of an identifier is 16 characters.
NAME – attribute name, if not set then this attribute is not available in tooltips. Characters **" ; =** are not allowed. The maximum length of the name is 48 characters. The names translate according to the language dictionary. If "V" is specified as the name, then the tooltip will display “Speed” and not “V”.

In addition to the main one, there is an additional format. If the attribute of incoming messages is not present in the list of the standard set of attributes (the **Attr** button in the script panel), but it needs to be displayed in tooltips, then the Attr keyword must be added to the beginning of the main format.

```
Attr Integer IO243 "Sensor 243"
```

.module onMessage

This module defines a Java program to be executed when new messages arrive.

Variables defined in the **Declare** block, standard attributes coming in messages, and object’s variables are available in this program. You can see the list of attributes by clicking on the **Attr** button (in the script panel), and the list of variables by clicking on the **Var** button.

Message attribute identifiers and object’s variables are case sensitive.

When using attributes, it is necessary to check their presence in the message or variables stored in the object. A parameter may not always be present in a message or in object variables. Validation examples:

```
double AV=0;
if (V!=null) {
  AV =  V*0.123
}
```

Script launch for saved events can be done from custom reports. You can use the **realtime** variable to determine the script's run mode. This variable is of type Boolean. If its value is **true**, it means that the script is launched when a message is received from the device, otherwise, the script is launched from the report.

```
if (realtime) {
  alt = V*100;
}
```

Comments

Scripts use comments like in Java. But they shouldn't placed before the first **onMessage** block, or **Declare** block. Examples:

```
double AV=0; // average script
/* this block uses when
V > 10
A < 100
*/
if (V!=null) {
//  AV =  V*0.123
}
```

Built-in functions

Message attributes

Group of functions for accessing the attributes of incoming messages:

getIntAttr(name)
getDoubleAttr(name)
getStringAttr(name)

setIntAttr(name,value)
setDoubleAttr(name,value)
setStringAttr(name,value)

Get functions return the value of the attribute given by the **name** string. If the attribute is not present in the message, **null** is returned. Returns types of Integer,Double, or String.

Set functions allow you to change the value of an attribute. The **value** types can be: Integer, Double, or String.

Examples:

```
Double MYBAT = getDoubleAttr("BAT2");
setStringAttr("MODE","SLEEP");
```

If the attribute is present in the list of standard attributes (the **Attrs** button in the script panel) or is defined in the **Declare** block, then the appropriate **identifier should be used** to access it, ie: not **setDoubleAttr("B",4.6);** but **B = 4.6;**

But in scripts for groups (where there are no standard attributes, since a group can have devices of different types), you should use the “Set” functions:

```
Double OB = getDoubleAttr("B");
if (OB!=null) {
 setDoubleAttr("B",OB*2);
}
```

Object variables

Getters / Setters

The function returns the current time in unix timestamp format (number of seconds since 1970).

Group of functions for accessing object variables:

getIntVar(name)
getDoubleVar(name)
getStringVar(name)

setIntVar(name,value)
setDoubleVar(name,value)
setStringVar(name,value)

The function group is similar to the message attributes group. Predefined system variables cannot be used in Set...Var functions. When you click on the **Var** button in the script panel, a list of current object variables is displayed. System variables appear at the top of the list and are marked with the **System** attribute. Examples:

```
Integer AZIM = getIntVar("azimut");
setDoubleVar("DENS",1.2)
```

delVariable(name)

Removing a variable with the identifier name from the object's variable set.

Global object variables

Functions for accessing global object variables defined outside of the script (defined in the object's main information panel).

getGlobalInt(name) Returns Integer for the global variable with ident of "name"
getGlobalDouble(name) Returns Integer for the global variable with ident of "name"
getGlobalStr(name) Returns String for the global variable with ident of "name”

Title, IMEI, Handler and BBT

getTitle()
getIMEI()
getHandler()

The functions return the name of the object, IMEI and the name of the protocol by which the object transmits data. All return values are of type String.

getInteger("BBT")

For objects of all device types there is a predefined variable **BBT** - Server timestamp (black box). It is added to the message if the time of the message and the time of its registration on the server differ by more than 5 minutes, or if the time of the message is older than the last message registered on the server. The variable stores the time the message arrived at the server in Unix timestamp format (number of seconds since 1970). Type - **Integer**.

Time functions

now()

The function returns the current time in unix timestamp format (number of seconds since 1970).

getMessageTime()

The function returns the time (unix timestamp) in the incoming message.

getElapsedTime()

The function returns the number of seconds elapsed since the object's last message.

isInTimeOfDay(from,to,offset)

Returns **true** if the current time is within the specified interval, taking into account the **offset** parameter (GMT offset in seconds). Otherwise, **false** is returned. Example:

```
String TOD = (isInTimeOfDay("01:30","18:30",7200))?"Work":"Rest";
```

getTime(timestamp,format,gmtoffset)

Returns String for given **timestamp** (unix timestamp) in the format specified in the **format** ("MMM.dd HH:mm", "yyyy.MM.dd HH:mm", …). **gmtoffest** set GMT offset in seconds. Example:

```
String INCOME_TIME = getTime(BBT,"yyyy.MM.dd  HH:mm:ss",7200);
```

Mileage and acceleration

getDistanceTo(latlon)

The function returns the distance (in meters) from the location in the current message to the point specified in the **latlon** parameter. The return type is Integer. Example:

```
Integer DIST2 = getDistanceTo("51.820191,-1.876096");
```

getDistance()

The function returns the distance (in meters) from the location in the current message to the previous location of the object. Example:

```
Integer DIST = getDistance();
```

getAcceleration()

The function returns the acceleration calculated from the speed in the current event and the last speed of the object. The return type is Double. Example:

```
Double ACCEL = getAcceleration();
```

getUMS(intvalue)

Returns an integer value, taking into account the measurement system set for this object (metric or imperial).
When the measurement system is set to imperial and the function is applied to a speed value, returns the speed value in miles/hour,
When applying the function to a distance value (and set to imperial), an amount of thousandth of a mile is returned.
The function is recommended for displaying data through a script when using the imperial measurement system.

Events

eventAdd(name)

The event specified by the **name** parameter is added to the current message. Example:

```
eventAdd("Normal");
```

eventAddByBit(sensor,bitpos,name)

If the **bitpos** bit in the variable specified by the **sensor** parameter is set to 1, then the event specified by the **name** parameter is added to the current message. Bits are numbered from right to left, starting from 1. Example:

```
eventAddByBit(SENSOR1,2,"COOLER ON");
```

If SENSOR1=5 (00000101), then the event will not be added, if SENSOR1=6 (00000110), then the event will be added.

eventContains(name)

Returns **true** if the message contains a **name** event, **false** otherwise. Example:

```
if (eventContains("Alert")) {
```

```
  MODE = 20;
}
```

eventClearAll()

All events are removed from the message. Example:

```
eventClearAll();
```

eventClear(name)

If there is an event named **name** in the message, then it is removed from the message. Example:

```
eventClear(("Normal");
```

eventReplace(name,toname)

If the message contains an event named **name**, then it is renamed to **toname**. Example:

```
eventReplace(("Normal","Normale");
```

Util functions

round(val,nsize)

Returns the rounding of the **val** (Double) parameter to **nsize** decimal places. Example:

```
Double nval = round(PWAL,2);
```

round(val)

Returns the rounding of the **val** (Double) parameter to the nearest integer. Example:

```
Integer nval = round(PWAL);
```

isBitSet(val,bitpos)

Returns **true** if the **bitpos** bit is set to 1 in the **val** (Integer) parameter. Bits are numbered from right to left, starting from 1. Example:

```
if (isBitSet(IVAL,3)) {
  MODE = 20;
}  else {
  MODE = 21;
}
```

If IVAL=5 (00000101), then the MODE=20 statement will be executed, if IVAL=9 (00001001), then the MODE=21 statement will be executed.

getSigned(val,numberofbytes)

Used to convert positive integers to signed numbers. The **val** parameter contains a positive number, the **numberofbytes** parameter specifies the dimension of the number into which the first parameter is converted. Example:

```
Integer nval = getSigned(val,1);
```

If the val parameter was equal to 136, then after the statement is executed, the navl variable will be assigned -120. If val was 125, the nval would be set to 125.

hasLatLon()

Returns **true** if the current message contains coordinates, **false** otherwise.

isCommand()

The function returns **true** if the current message is a command or an object's response to a command, and **false** otherwise.

getStringByVal(param, cond1, cond2, ...)

Based on the value of the **param** parameter, a string is returned, selected from the conditions strings **cond1**, **cond2**, … Condition strings have the format "*condition value string*". Spaces are required between *condition*, *value* and *string*. *condition* can have one of the following values: < <=> >= = . *string*(should not contain spaces) specifies the parameter that is returned when the condition is met. If none of the conditions are met, then **null** is returned. Example:

```
String sector = getStringByVal(F,"< 90 SECTOR1","< 180 SECTOR2","< 270 SECTOR3",">= 270 SECTOR4");
```

getCalibrationValue(xparam, double[][] arr)

The function returns a numeric value using the linear functions specified by the **arr** parameter. Two points [n][0] and [n+1][1] define a linear function ax + b. If the value of the **xparam** parameter is outside the range specified by the arr parameter, then **null** is returned. Function is useful to calculate the fuel in the tank by the value of the sensor (linear functions in this case define the calibration table). Example:

```
Double Fuel = getCalibrationValue(X1,  new double[][] { { 20, 0 }, { 1000, 40 } });
```

Object Icon

setIcon(icon)

Function is used to change the object icon. The icon is set by the **icon** parameter. Example:

```
setIcon("ncar03");
```

setIconColor(color)

Function is used to change the color of the object's icon. The color is specified in the format “RRGGBB”. Only the dots of the icon (png format) colored in black are changed. Example:

```
setIconColor("FF0000");
```

iconByVal(xparam, cond1, cond2, ...)

Setting the icon image based on the value of the **param** parameter. See the getStringByVal(param, cond1, cond2, ...) function for the format of condition strings cond1, cond2, ... The name of the icon is given without extension. Icons are selected from the current group of object's icons. Example:

```
iconByVal(V, "< 10 car2", "< 50 car3", ">= 50 car4");
```

iconColorByVal(xparam, cond1, cond2, ...)

Setting the color of the icon based on the value of the **param** parameter. See the getStringByVal(param, cond1, cond2, ...) function for the format of condition strings cond1, cond2, ... The color is specified in RRGGBB format. Example:

```
iconColorByVal(V, "< 1 cccccc", "< 40 00ff00", "< 60 0000ff", ">= 60 ff0000");
```

Geofence

isInZone(zoneid)

Returns **true** if the object is in the zone with the **zoneid**, **false** otherwise. The zone ID is displayed in the "Basic info" geofence panel. Example:

```
if (isInZone(4443)) { …..  }
```

timeStayInZone(zoneid)

Returns the number of seconds the object has been in the given zone. If the object is outside the zone, or has just entered the zone, 0 is returned. Example:

```
Integer INZONE1 = timeStayInZone(4443);
```

getCurrentZoneId()

The function returns the identifier of the geofence (type of Integer) in which the object is currently located. The value 0 means that the object is outside the geofences of the current group. The value -1 means that the current message has no GPS coordinates.

```
Integer CURRENT_ZONE = getCurrentZoneId();
```

getCurrentZoneId(tag)

The function is similar to the **getCurrentZoneId** function, except that only zones with the tag specified by the parameter are taken into account.

```
Integer CURRENT_ZONE = getCurrentZoneId(“parking”);
```

getZoneAttr(zoneid,attrname)

The function returns the value of the attribute specified by the **attrname** (String) parameter for the geofence with the **zoneid** (Integer) identifier.
List of attribute names:
"name" - geofence name

```
String CURRENT_ZONE_NAME = getZoneAttr(curzone,”name”);
```

Command

sendCommand(command,humanreadabletext);

Sending a command to the device. The **command** parameter specifies the body of the command that is sent to the device (command formats depend on the object device type). The **humanreadabletext** parameter specifies the text that will be displayed in event logs and reports. Example:

```
sendCommand("**,imei:%IMEI,C,60s","Report interval 1min");
```

Mobile info (MCC,MNC)

getMobileInfo()

Returns information about the current mobile operator. A String is returned containing the country and current mobile operator through which the data is being transferred. Example: “China - China Mobile“.

isMobileChanged()

Returns **true** if there was a change of the mobile operator in the current message. Otherwise, **false** is returned.

Task

getTaskState()

The function returns the state (status) of the task execution for the current object. If the object has not performed (and is not currently performing) tasks, then **null** is returned. The return type is a **String**. Constants available for statuses:

- **IDLE** - the object is free, ready to perform tasks
- **EXEC** - the object is performing a task
- **DONE** - the object has completed the task (perhaps it is returning to the base)
- **CANC**- task for the object was canceled in the process of performing the task (perhaps it is returning to the base)

```
String state = getTaskState();
```

setTaskState(state)

The function sets a new task execution state for the current object.

```
setTaskState(DONE);
```

Examples

The color of the icon is set depending on the speed. Mileage of the object in km mapped to ODOM variable.

```
.module Declare
Double ODOM "Odometer"
.module onMessage
iconColorByVal(V, "= 0 CCCCCC", "< 40 FFFF00", "<= 60 00CCCC", "> 60 996633");
ODOM = round (((double) getOdometer(2000, 800000, "Passed 800km ") )/1000,1);
```

Executed for all objects of the group and its child subgroups (for those objects that do not have their own scripts defined). The same actions are performed as in the previous script, but access to the parameters is made not by identifiers, but through the get...Attr functions In addition, for objects that use Teltonika protocol, the FLAG1 variable is set based on the value of bit 18 from the IO517 attribute, and for objects that use Coban protocol, the "ACCOFF" events are renamed to "IGNITION OFF".

```
.module Declare
String FLAG1 "Flag1"
```

```
Double ODOM "Odometer"
.module onMessage
iconColorByVal(getIntAttr("V"), "= 0 CCCCCC", "< 40 FFFF00", "<= 60 00CCCC", "> 60 996633");
ODOM = round (((double) getOdometer(2000, 800000, "Passed 800km ") )/1000,1);
switch (getHandler()) {
case "Teltonika":
    String str517 = getStringAttr("IO517");
    if ( str517!=null)  {
         long state = Long.parseLong(str517,16);
         FLAG1 = isBitSet(state,18)?"Y":"N";
   }
   break;
case "Coban":
   eventReplace("ACCOFF","IGNITION OFF");
   break;
}
```