Geometris GSON Programming Quick Start

Version 1.0

October 10, 2023

Contents

Introduction

GSON (Geometris script object notation) is the scripting language used to customize the behavior of Geometris devices. Gson syntax is specified with a specific schema for the popular json (javascript object notation) data description language often used in web development, service interfaces, and structured data storage. This means that, using the supplied schema, you'll be able to use gson with the many tools and editors that have support for json, and anyone familiar with json and json schemas will easily understand its syntax.

Use Cases

Some example scenarios you can do with gson include:

- Creating a packet with a custom message type or custom payload in it, maybe computed or read from a peripheral.
- Summarize and compute key performance indicators for telematics data on the edge.
- Filter data for specific criteria from the speed, accelerometer, fuel usage, usage hours, and geofences to determine when to send a packet.
- Monitor data from a peripheral and include it in data packets.
- Control connected peripherals using GPIO or serial bus connections to indicate alerts, send messages, etc.

Features

Some additional features of gson we think you'll find useful:

- Support for jsonc – json with c style comments – so it's easy to understand the intentions of the script author.
- A plug-in for visual studio code.
  - Editing, code-completion, schema checking for gson scripts.
  - Assemble scripts into binary code for use on the device.
  - Upload scripts to a device via serial port.
  - Debug scripts, either via serial port, or live in the field.
    - Breakpoints, stepping through script code, and variable watches.
  - Debug using script log files. This allows debugging script behavior offline afterward.
- Script Capabilities:
  - Script variables set via configuration parameters. This allows you to control script behavior with configuration commands and CAST configuration files.
  - Define functions in separate library files. This makes code re-use simple.
  - Change monitor events: closely monitor device and engine variables such as speed, battery voltage, miles travelled, etc., and respond when they enter or exit given boundaries.
  - Timer events: to evaluate variables or do a task on a regular basis.
  - State machine logic: define state machine logic declaratively for easy state machine logic. This simplifies the logic for many scripts, so you have less code to maintain.

Starter Examples

Let’s cover the basics with a simple hello, world style script.

helloworld.jsonc

```
{
  // the json schema that specifies the gson syntax.
  "$schema": "..//gsonscriptschema.json",

  // Initialization section, used to declare timers, functions, configuration items, string variables, etc
  "init": {

    // timer section defines timers that signal on a regular basis
    "timers": [
      {
        "key": "@beat", // name of this timer
        "start_now": true, // start immediately with the script
        "duration": 1, // one second long
        "recurrent": true // repeat after signaling
      }
    ],

    // string section defines constant strings and variable strings
    "strings": [

      {
        "key": "@message", // name of the string
        "val": "hello, world, at beat {0}!" // constant string because we define a value
      }
    ]

  },


  // This is the main function of the script.
  // Any time an event occurs (device event, timer event, etc), this function will
  // respond to the event.
  "main": [

    { // a function is a list of commands. This command is an "if/then command,
      // that looks for a specific type of event to respond to.
      "if": [
        {
          "on": "event.timer", // if the event is a timer event
          "index": "@beat" // for this specific timer
        }
        // we could have additional conditions here

      ], // then do the following actions
      "then": [
        { "++": "#counter" }, // increment the variable "#counter"
        // simple variables like ints and floats, and booleans default to 0 (false)
        // and are initialized the first time they are used.
        // Variables starting with a # are durable, meaning they're saved between calls to main.
        // If you started with "$" instead, it would be a temporary variable, and would be
        // recycled at the end of main.

        { "let": "#mode","=": "#counter","%": 3 } // calculate the modulus of the counter with 3
      ]
    },
    { // Here's the next command to run. It's a simple if/then command, too.
      "if": [
        { "cmp": "#mode", "==": 1  } // every third beat
      ],
      "then": [
        {
          "log": "@message",
          "sub": [ "#counter" ]
          // log a message, which you can see in the log or in the debugger.
          // variables in the "sub" argument will be substituted in for
          // placeholders in the string @message. In this case, the placeholder
          // "{0}" will be replaced with the value of the variable "#counter".
        }
      ]
    }


  ]


}
```

Explanation:

- The schema. Using the gson schema is a key element for integration with your development tools, including the visual studio code plugin. It allows for validation, error checking, code-completion, display of script syntax documentation, etc.
- The init section. This section is used to declare things before the script is started. Some examples include:
    - Strings, either constant strings or string variables.
    - Timers, used to create regular events.
    - Functions, sub-routines callable from your main script function or each other.
    - Script Parameters, variables that can be written to only from configuration commands.
    - State machines, which define state machine based logic you can use to drive script behavior.
- The main section. The device will invoke this section every time an event occurs. Device events like periodics, fence crossings, idling, speeding, ignition on and off will invoke the function. Other user-defined events like timers can also occur.
    - Main commands. Inside the main function is a list of commands. The commands in this script are simple if/then statements, but other kinds exist, too. For example, simple lists of unconditional actions, while loops, switch statements.
    - Commands:
        - Conditions. Many types of commands support conditions, so that they will only invoke their actions if the conditions are true. Often the condition will be checking the type and identity of the incoming event. Gson can support complex conditions as well, as we'll see later.
        - Actions. Actions do useful things on the device, like compute a variable, log information, send a packet, or write to the serial port.
- On conditions. On conditions filter for events matching given ids and criteria. These include all of the whereqube events supported by the device:

```
{
    "$schema": "gsonscriptschema.json",

    "main":[

        { /* if we get a device driver id event. */
            "if":[
                {"on":"event.wq", "index":"event.driver.id"}
            ],
            "then":[
                { "buzzer.on": 30 }

            ]
        }
    ]


}
```

speedmode.jsonc

```
{
  "$schema": "./gsonscriptschema.json",

  "main": [
    { // Instead of an if/then command, we're going to use a "do" command.
      // This kind of command does actions unconditionally.
      "do": [
        {
          "let": "$state", "=": -1 // we assign this temporary variable to -1
        }
      ]
    },
    {
      "if": [
        {
          "change": "event.change.speed",
          // monitor speed changes. We'll can send an event when speed is
          // inside a given range, or outside a range, but we'll show something
          // more sophisticated with modal (inside/outside) changes.


          "modal": [
            // A modal monitor has an inside (mode 1) and outside (mode 0) range.
            // We're watching speed to see if it changes from one mode or another.

            //
            0.0,
            51.0, // inside range: 0 - 51 mph.
            // if we're in outside mode, we'll change to inside mode when
            // speed stays in this range for 4 seconds.

            //
            0.0,
            81.0 // outside range: 0 - 81
            // if we're in inside mode, we'll change to outside mode when
            // speed stays outside this range for 4 seconds.
          ],

          "default": 1, // start with inside mode
          "debounce": 4 // debounce time in seconds.

        }
      ],
      "then": [
        {
          "let": "$state", "=": "event.arg", "sel": 1

          // events can have arguments. The change event for modal monitors
          // has an argument at index 1, that describes the mode we left when it changed.

        }
      ]
    },
    {
      "if": [
        {
          "cmp": "$state", "==": 0 // are we in outside mode?
        }
      ],
      "then": [
        { "buzzer.on": 10 } // turn on the device buzzer for 10 seconds
      ],
      "else": [
        { "buzzer.on": 0 } // we're in inside mode, turn off the buzzer.
      ]
    }

  ]
}
```

Explanation:

- No init section. If you don't have anything to declare, as in this example, the init section isn't required.
- Initializing a temporary variable. If we get a speed change event, we'll set this variable $state to get mode information about the change. However, if the event is something else (for example, ignition off event), then we assign $state to -1 to indicate it's irrelevant.
- Monitor speed for changes. We set up a monitor on speed to see if changes to speed meet certain conditions. For example, we could see if the speed slows way down, or if the vehicle is speeding outside a given range. In this script, we'll use both:
    - Modal change monitoring.
        - In this case, we want to warn the driver with the buzzer if they're above 81 mph by turning on the device buzzer. We'll consider this "outside" mode since the speed is outside a boundary of 0-81 mph.
        - We want to continue to consider the driver speeding until they slow down to 51 mph or less. This will put them into inside mode.
        - Modal change monitors are designed for this use case. A given debounce property tells gson how long the speed variable will need to match the boundary conditions to change modes.
        - The monitored variable (speed in this case) will need to stay inside the inside mode range to switch to inside mode.
        - Speed will need to stay outside the outside mode range for the specified seconds to switch to outside mode.
        - Changing modes will cause a change event.
- Getting event details.
    - Events can have arguments; how many and what they mean vary per event. For change monitor events on a modal monitor, argument 1 is the mode we changed from.
- Turn the buzzer on or off. We can specify how many seconds we want the buzzer on, and zero means to turn it off.
- Simpler monitoring. "inside" and "outside" monitoring is supported for a wide variety of properties. Here's an example:

```
{
    "$schema": "./gsonscriptschema.json",
    "main":[
        {
            "if": [
                {
                    "change": "event.change.speed", //speed range
                    "outside": [0, 80],
                    "debounce": 1
                }
            ],
            "then": [ { "buzzer.on": 3 } ] //result
        }
    ]
}
```

subroutines.jsonc

```
{
  "$schema": "./gsonscriptschema.json",
  "init": {
    // let's set up a 1 second recurrent timer
    "timers": [
      {
        "key": "@driver",
        "duration": 1,
        "recurrent": true,
        "start_now": true
      }
    ],

    // we declare some constant strings to help with logging
    "strings": [
      {
        "key": "@logger",
        "val": "called from {0}."
      },
      {
        "key": "@fnA",
        "val": "main function"
      },
      {
        "key": "@fnB",
        "val": "function B"
      },
      {
        "key": "@fnC",
        "val": "function C"
      }
    ]


  },

  "main": [
    {
      "do": [
        { "log":"@logger", "sub":[ "@fnA" ]},
        { "call": "fnB" }, // call our subroutine
        { "call": "fnC" } // call our subroutine
      ]
    }
  ],


  "subs": [
    {
      "name": "fnB",
      "fn": [
        {

          "do": [
            { "log":"@logger", "sub":[ "@fnB" ]},
            { "return": 0 }
          ]
        }
      ]
    },
    {
      "name": "fnC",
      "fn": [
        {

          "do": [
            { "log":"@logger", "sub":[ "@fnC" ]},
            { "return": 0 }
          ]
        }
      ]
    }
  ]
}
```

Explanation:

- This script illustrates how to declare sub-routines in the same file, then call them.
- “subs”, this optional section allows declaration of callable sub-routines. There is also a way to declare them in a separate file and then import them in via the init section, but for declarations in the same file, the subs section describes each subroutine.
- Declaring sub-routines. Each sub-routine requires a unique name and then specifies a list of commands (in “fn”) to perform. The commands are just like the commands described in the main section.
- Calling sub-routines. In the main section actions, the “call” action is used to invoke a sub-routine.

imports.jsonc

```
{
  "$schema": "./gsonscriptschema.json",
  "init": {
    // let's set up a 1 second recurrent timer
    "timers": [
      {
        "key": "@driver",
        "duration": 1,
        "recurrent": true,
        "start_now": true
      }
    ],

    // we declare some constant strings to help with logging
    "strings": [
      {
        "key": "@logger",
        "val": "called from {0}."
      },
      {
        "key": "@fnA",
        "val": "main function"
      },
      {
        "key": "@fnB",
        "val": "function B"
      },
      {
        "key": "@fnC",
        "val": "function C"
      }
    ]
  },

  // import sub-routines from another file
  "imports": ["demosubs.jsonc"],

  "main": [
    {
      "do": [
        { "log":"@logger", "sub":[ "@fnA" ]},
        { "call": "fnB" }, // call our subroutine
        { "call": "fnC" } // call our subroutine
      ]
    }
  ]

}
```

demosubs.jsonc

```
{
  "$schema": "./gsonscriptschema.json",


  "subs": [
    {
      "name": "fnB",
      "fn": [
        {

          "do": [
            { "log":"@logger", "sub":[ "@fnB" ]},
            { "return": 0 }
          ]
        }
      ]
    },
    {
      "name": "fnC",
      "fn": [
        {

          "do": [
            { "log":"@logger", "sub":[ "@fnC" ]},
            { "return": 0 }
          ]
        }
      ]
    }
  ]
}
```

Explanation:

- Here we see the same logic using sub-routine imports from a separate file. You can create sub-routine files to provide various functions for your scripts, and then import and call them as needed.

controlflow.jsonc

```
{
    "$schema": "gsonscriptschema.json",
    "init":{
      "timers": [
        { "key": "@checktimer", "duration": 10, "start_now": true, "recurrent": true }
      ],
      "strings": [
        { "key": "@formatstr", "val": "65.28.9.36.3.4.7.8.192" },
        { "key": "@bingo","val": "bingo!" },
        { "key":"@debugmark","val":"debug: {0}" },

        { "key" :"@varstr0" } // string variable, not constant. Up to 4 string variables may be declared.
      ]
    },

    "main":[

        {
            "if":[{ "on":"event.timer", "index":"@checktimer" }],
            "then":[{ "packet.send": 15, "format": "@formatstr"}]
        },
        {
            "do":[
              {
                "=>":[ // lambda / implicit function / nested commands
                  {

                    "do":[
                      { "let":"#test", "=":3 },
                      {
                        "=>": [
                          {
                            "if": [ {"cmp": "#test", "==": 3 } ],
                            "then": [
                              { "log": "@debugmark", "sub": [ 1 ] },
                              { "let": "#x", "=":5}

                            ]
                          }
                        ]
                      }
                    ]

                  },
                  { // switch statement
                    "switch":"#test",
                    "cases":[
                      {
                        "is": 1, // literal ints or strings can be cases
                        "then": [ { "log": "@debugmark", "sub":[ 2 ] } ]
                      },
                      {
                        "is":3,
                        "then": [
                          { "log": "@debugmark", "sub":[ 2 ] },
                          { "let":"@varstr0", "=":"@bingo" }
                        ]
                      },
                      {
                        "default": [
                          { "log": "@debugmark", "sub":[ 3 ] }
                        ]
                      }

                    ]
                  } // switch
                ] // => actions
              } // then action
            ] // then
        }, // test lambda

        { "do":[ { "let":"#i", "=": 0 } ] }, // init index for loop

        { // while loop
          "while": [
            { "cmp":"#i", "<": 10 }
          ],
          "limit": 10,
          "then": [
            { "log": "@debugmark", "sub":[ "#i" ]},
            { "++":"#i" }
          ]

        }
    ]
}
```

Explanation:

- Send a packet with a custom reason. The packet.send action can send device packets to the server with a custom reason, and as we'll see later, a custom payload as well.
- Implicit functions. In a gson command, there will typically be a list of conditions, and then a list of actions to perform. Sometimes, you'll need more sophisticated logic such that your actions should be additional commands, along with conditions and their own actions. The "=>" action is used to achieve this. It allows a list of additional commands to perform as an action. Each use of "=>" will use up one of the available function declarations for the script. Gson allows up to 64 functions to be declared for a script.
- Switch statement. A switch statement can examine the contents of an int or string variable, and match values to actions to perform. This is just like the classic C language family switch statement, making multiple comparisons simpler to script.
- While statement. Gson allows limited loop logic to perform a set of actions iteratively. The while loop will continue until the given condition is false, or the limit is reached. Gson will also exit any loop taking more than 5 seconds of runtime automatically so as to preserve device function.

math.jsonc

```
{
  "$schema": "..//gsonscriptschema.json",
  "init": {
    "timers": [
      {
        "key": "@checktimer", // we'll use a timer to trigger every 10 seconds
        "duration": 10, "start_now": true, "recurrent": true
      }
    ],
    "strings": [
      { // this is the string of the log format to output a few of our variables
        "key": "@clogfmt", "val": "counter: {0}, flt:{1}, bool:{2}"
      },
      { // this string will be used to demonstrate string concatenation
        "key": "@counterchar", "val": "!"
      },
      { // this is a variable string because it doesn't have a provided constant value
        "key": "@varstr"
      }
    ],
    "startup": [ // the start up function runs once when the script loads
      {
        "do": [
          // here are some simple assignments
          { "let": "#testflt","=": 1.618 },
          { "let": "#testint","=": 1 },
          { "let": "#testbool","=": true }
        ]
      }
    ]
  },
  "main": [
    {
      "if": [
        { // if the event is our timer
          "on": "event.timer", "index": "@checktimer"
        },
        { // AND counter is less than 60
          "cmp": "#counter","<": 60
        }
      ],
      "then": [
        {
          "++": "#counter","limit.max": 5 // increment counter by 1, but limit it to 5
                        // decrement is supported too with --
        },
        {
          "let": "#testflt","=": "#testflt", "*": 1.025 // multiplication.
        },
        {
          "let": "@varstr","=": "@varstr", "+": "@counterchar" // addition of strings performs concatenation
        },

        {
          "let": "#testbool","=!": "#testbool" // equals not -- toggles a boolean
        },
        {
          "let": "$counterpart", "=": "#counter", "%": 5 // modulus operator
        },
        {
          "let": "$tempthing","=": 5.0,"/": "$counterpart" // division
        },
        {
          "map": "$tempthing", // map $tempthing from one range to another
          "from": [ 0.0, 1.0 ],"to": [ 1.0, 10.0 ],
          "into": "$remapped"
        },
        {
          "let": "$minimum", "min": [30,"#counter"] // min and max operators
        },
        {
          "log": "@clogfmt", "sub": ["#counter","#testflt","#testbool"]
        }
      ]
    }
  ]
}
```

Explanation:

- This script demonstrates math operations.
- The interpreter can automatically coerce values of type int into floats when needed.
- Strings cannot be interpreted as numeric values, but the + operator can concatenate strings.
- Initial values for durable variables (beginning with “#”) may be done in the startup function.
- Temporary variables are reset to default at the beginning of the main function:
    - 0 for ints
    - 0.0 for floats
    - False for booleans
    - “” for strings

booleans.jsonc

```
{
  "$schema": "gsonscriptschema.json",
  "init": {
    "params": [ // script parameters are script variables set by configuration commands. Config params 750 - 781
      {
        // notice this is used to control the duration of our timer.
        // these are treated like read only variables inside the script.
        "key": "*timerduration", "dataType": "integer","default": 10,
        "config": 0 // set with parameter 750
      },
      {
        "key": "*initialstate", "dataType": "boolean", "default": false,
        "config": 1 // set with parameter 751
      }
    ],
    "timers": [
      { "key": "@beat","start_now": true, "recurrent": true, "duration": "*timerduration" }
    ],
    "strings": [
      { "key": "@packetfmt", "val": "65.28.3.4.128" }, // custom packet format
      { "key": "@payload", "val":"{0}:{1}" } // custom payload format
    ],
    "startup": [
      {
        "do": [
          { "let": "#state", "=": "*initialstate" }, // notice we use a parameter here
          { "let": "#counter","=": 1 }
        ]
      }
    ]
  },
  "main": [
    {
      "if": [ // conditions are "anded" together
        { "on": "event.timer","index": "@beat" },
        // except when you use "into" to send the result into another variable
        { "cmp": "#counter","<": 4,"into": "$lowerrange" },
        { "cmp": "#counter",">=": 6,"into": "$upperrange" },
        // now we can or our temporary conditions together
        { "cmp": "$lowerrange", "or": "$upperrange" } // on timer and (lowerrange or upperrange)
      ],
      "then": [
        { "let": "#state", "=!": "#state"  },// we can do boolean operations in the actions, too
        { "let": "$tempNE", "=": "#state", "!=": true },
        { "let": "$tempGT", "=": "#counter", ">":  10 },
        { "let": "$doit", "=": "$tempNE", "and": "$tempGT" }
      ]

    },
    {
      "if": [ { "cmp": "$doit","==": true } ],
      "then": [
        // custom packet send. We send a custom packet id > 200, with custom packet format and payload
        { "packet.send": 222, "format": "@packetfmt", "payload": "@payload", "sub": [ "#counter", "#state"] }
      ]
    },
    {
      "do": [ { "++": "#counter" } ]
    }
  ]

}
```

Explanation:

- Script parameters.
    - These variables are set using device configuration parameters 750-781, with the SETPARAMS command. SETPARAMS immediately changes the value of the parameter.
    - The parameters may be read, but not written to, within the script.
    - You declare a data type and default value, for if the parameter is never set from the outside.
    - Notice we use *timerduration to specify our timer later.
    - The “config” property of a parameter is added to 750 to index the corresponding configuration.
- Complex Boolean criteria.
    - Normally, all Boolean expressions for an “if” are joined together with “and” for a complete expression.
    - Any boolean expression may instead put its result into a variable using the “into” property.
    - The variable you put results into may become the inputs to later Boolean evaluations.
    - Actions can also compute Boolean expressions and comparisons. Later commands may leverage these.
- Custom packets include:
    - A custom reason. Custom reasons start at 200.
    - Custom packet format. Variable 128 may be used as a custom payload.
    - Custom payload. This string may use string substitution to insert script variables into the payload.

machine.jsonc

```
{
    "$schema": "..//gsonscriptschema.json",
    "init":{
        "timers":[ {  "key": "@beat", "start_now": true, "duration": 1, "recurrent": true } ],
        "strings": [ { "key": "@log", "val": "state: {0}. time: {1}" } ],
        "machines": [
          {
            "key": "::testmachine", // unique name for the machine
            "states": [ // declare states, up to 20 per machine. First state declared is the initial state.
              {
                "key": "quiet", // name of the state
                "transitions": [ // declare transitions to other states, up to 12
                  {
                    "action": "greetformal", // action that will cause the transition
                    "goto": "hello" // new state to go to
                  }
                ]
              },
              {
                "key": "hello", // another sstate
                "transitions": [
                  {
                    "action": "done", // action to cause transition
                    "goto": "quiet" // state to go to
                  }
                ],
                // after entering this state, if no other transitions occur
                // within 3 seconds, the interpreter will automatically
                // invoke the "done" action
                "timeout": 3,
                "timeout.action": "done"
              }
            ]
          }
        ]
    },
    "main": [
      {
        "if": [
          // event transition catches state transitions. We can filter for the machine and states.
          { "on": "event.transition", "machine": "::testmachine", "state.match": [ "::testmachine.quiet", "::testmachine.hello" ] }
        ],
        "then": [
          // we can get the state out into a variable
          { "machine.get": "::testmachine", "into": "#state" },
          { "log": "@log", "sub": [ "#state", "time.utc" ] }
        ]
      },
      {
        "if": [ { "on": "event.timer", "index": "@beat" } ],
        "then": [
          { "++": "#counter" },
          { "machine.get": "::testmachine", "into": "#state" },
          {
            "=>": [
              {
                "if": [
                  { "cmp": "#counter","==": 1 },
                  // we can compare a variable to a state by state name
                  { "cmp": "#state","==": "::testmachine.quiet" }
                ],
                "then": [
                  { // we can invoke an action on a state machine to cause a transition
                    "machine.invoke": "::testmachine", "action": "::greetformal"
                  }
                ]
              }
            ]
          }
        ]
      }
    ]
}
```

Explanation:

- An example of a state machine is a traffic light. Depending on the state {red,yellow,green,blinking yellow, blinking red}, the logic of how the vehicles at the intersection behavior will change. The transitions between the states may be caused by a time out, or by sensors that detect the position of vehicles at the intersection.
- Scripts may also have different states, and different behavior for those states. For example, turning on or off an alarm peripheral based on ignition state, previous alarm duration {short, medium, long}, a script parameter, and the id of the last geofence entered.
- Gson supports state machines. These can be used to greatly simplify script logic when behavior is stateful.
- In the example script, we declare a state machine. You may declare up to four state machines in a script.
- A state machine has a list of states.
- Each state has a list of actions that transition the machine to a different state.
- Gson supports automatic timeouts on states to invoke a given action.
- You can intercept a state transition with the “on” condition, and filter for just the states you want to address.
- You can read a state machine’s state into a variable.
- You can invoke a machine action in the script logic.

gpio.jsonc

```
{
    "$schema": "gsonscriptschema.json",
    "init":{
        "timers": [
            {
                "key": "@checktimer", "duration": 30,
                "start_now": true, "recurrent": true
            }
        ],
        "strings": [ { "key": "@formatstr", "val": "65.28.9.36.3.4.7.8.192" } ],
        "startup": [
            {
                "do":[
                    { "let":"#sendpacket", "=": false },
                    { "let":"#packetsent", "=": false }
                ]
            }
        ]
    },
    "main":[
        { // if ignition is on, set line 1 high to turn on driver id reader
            "if":[
                { "cmp":"calc.ignition.session.seconds", ">": 0 }  // ignition is on
            ],
            "once":[ // once will do these actions once until the condition is no longer true
                { "gpio.set": 1 } // set bit 1
            ]
        },
        { // if ignition is off, set line 1 low to turn off driver id reader
            "if":[
                { "cmp":"calc.ignition.session.seconds", "<=": 0 }  // ignition is off (replace prop)
            ],
            "once":[
                { "gpio.clear": 1 }, // clear bit 1
                { "let":"#packetsent", "=":false }
            ]
        },
        {
            "if":[
                { "cmp":"calc.ignition.session.seconds", ">":30 }, // ignition on > 30 seconds
                { "cmp":"onewire.driverid.count", "<=": 0 } // we didn't find a driver id yet
            ],
            "then":[ // we didn't get an id
                { "buzzer.on": 1 }, // buzzer 1 second
                { "gpio.blink":3, "rate":4, "duration":50 },
                { "let":"#sendpacket", "=":true }
            ]
        },
        { // if we get a device driver id event, send this packet.
            "if":[
                {"on":"event.wq", "index":"event.driver.id"}
            ],
            "then":[
                { "gpio.blink":3, "rate":0, "duration":0 },
                { "let":"#sendpacket", "=": true },
                { "let":"#packetsent", "=": false }
            ]
        },
        { // maybe send a packet
            "if":[
                { "cmp":"#sendpacket", "==": true },
                { "cmp":"#packetsent", "!=": true }
             ],
             "then":[
                { "packet.send":210, "format":"@formatstr" },
                { "let":"#packetsent", "=": true },
                { "let":"#sendpacket", "=": false }
             ]
        }
    ]
}
```

Explanation

- This script demonstrates the use of general purpose io with the device, which may be used to control external peripherals:
  - "gpio.set" – turn on the given bit
  - "gpio.clear" – turn off the given bit
  - "gpio.blink" – turn the bit on and off with a given rate
  - In this script, we imagine a peripheral used to alert the driver that they must conform to driver id usage policy, and scan their id before driving.
- We introduce another control flow statement the "if/once". This is used to perform the actions one time. They will not be done again until the conditions in the "if" are first false, then true again.
- Driver id event. Whereqube driver id functionality can be used in scripting by capturing the given event. The given packet format used with "packet.send" includes the driver id.

uart.jsonc

```
{
  "$schema": "../gsonscriptschema.json",
    "init": {
    "timers":[
      {
          "key": "@beat", "start_now": true,"duration": 4, "recurrent": true
      }
    ],
    "strings": [
      {
        "key": "@debugstr",
        "val": "received: {0}"
      },
      {
        "key": "@formatstr",
        "val": "command: {0}"
      },
      {
        "key":"@hi",
        "val":"GSON: HELLO WORLD"
      },
      {
        "key": "@infostr" // variable string (no constant val specified)
      }
    ],
    "startup": [
      {
        "do": [ // initial startup, open our channel
          {
            "channel.open.text.uart": 1,
                "baud": 9600, // 8 N 1 are defaults
                "channel": "#com1"

          }
        ] // end actions
      }
    ] // end startup
  },
  "main": [
    {
      "do": [ // redo in case it was closed. If #com1 is still open, will re-use existing.
        {
          "channel.open.text.uart": 1, // channel in text mode will have different events from stream mode
              "baud": 9600, // 8 N 1 are defaults
              "channel": "#com1"
        }
      ]
    },
    {
      "if":[ { "on": "event.timer", "index":"@beat" } ], // timer every second
      "then":[
        {
          "channel.xmit.text": "#com1",
          "message": "@hi"
        }
      ]
    },
    {
      "if": [
        { // watch for receiving text data
          "on": "event.channel.receive",
          "channel": "#com1",
          "channel.terminators.hex": [ "0D0A" ] // a message ends with CR LF
        }

      ],
      "then": [
        {
          "let": "@infostr", "=": "event.arg", "sel": 3 // event.channel.receive puts the received message in event.arg[3]
        },
        {
          "channel.xmit.text": "#com1",
          "message": "@formatstr",
          "sub": [ "@infostr" ] // transmit text message through channel
                                  // xmit through stream will require an action with different arguments.
        }
      ]

    }
  ] // end main
}
```

Explanation

- This script demonstrates the use of serial ports using Gson.
- Opening a serial port is done with channel.open.text.uart. If the com port is already open, then this command is ignored.
- channel.xmit.text is used to send string text data through the given com port. Note that the use of substitution parameters on the text string is available, so you can insert variables into the outgoing text.
- “on” event handler can see incoming received data with “channel” and “channel.terminators.hex”
    - channel.terminators.hex watches for lines terminating with the given characters. The ascii characters to terminate with are specified as hexadecimal numbers.
    - The given example watches for carriage return / line-feed

strings.jsonc

```
{
  "$schema": "..//gsonscriptschema.json",
  "init": {
    "timers": [  { "key": "@checktimer","duration": 4, "start_now": true, "recurrent": true }  ],
    "strings": [
      { "key": "@formatstr", "val": "65.28.9.36.3.4.7.8.192" },
      { "key": "@search","val": ".192" },
      { "key": "@mark", "val": "count: {0}, result: {1}, token:{2}" },
      { "key": "@clogfmt", "val": "mystr: {0}" },
      { "key": "@varstr" },
      { "key": "@mystr" }
    ]
  },
  "main": [
    {
      "if": [ { "on": "event.timer","index": "@checktimer" },
        { "cmp": "#counter","<": 20 }
      ],
      "then": [
        { "let": "#find", "=string.find": { "in": "@formatstr", "find": "@search", "from": 4 } },
        { "let": "@mystr","=string.extract": { "in": "@formatstr","start": 3, "end": 4 } },
        { "let": "#count","=string.token.count": "@formatstr","separators": [ "." ] },
        { "let": "#test", "=string.cast": "@mystr", "dataType": "integer" },
        { "let": "#len", "=string.length": "@formatstr" },
        {
          "let": "#token", "=string.token": "@formatstr", "sel": "#counter",
          "dataType": "integer", "separators": [ "." ], "result": "#res"
        },
        {
          "let": "@mystr", "=string.format": "@mark",
          "sub": [ "#count", "#result", "#token" ]
        },
        { "log": "@clogfmt", "sub": [ "@mystr" ] },
        { "++": "#counter", "limit.max": 20 }
      ]
    }
  ]
}
```

- This script illustrates using gson to process text data in strings.
- The “string.find” action searches a given string for a specified substring.
    - The variable “#find” will get the position of where the substring starts in the given search string, or -1 if not found.
    - “in” specifies the string to search
    - “find” is the string to search for
    - “from”, which is optional, allows searching from the given position in the search string (from position 0 at the start).
- “string.extract” gets a substring out of the given string.
    - The variable “@mystr” will receive the result
    - “in” is the string to extract from
    - “start” is the starting position to begin extraction
    - “end” is the position to finish extraction
- “string.token.extract” counts how many tokens there are inside the given string, separated by the given separators.
- “string.cast” can convert text into numeric, integer, or Boolean variables
- “string.len” gets the length of the given string.
- “string.token” separates the given string into tokens, extracts one by index, and converts it to the specified data type.
    - The variable “#token” will receive the result
    - “@formatstr” is the text containing the tokens, in this case separated by “.”
    - “sel” indicates which token to extract
    - “dataType” casts the token to Boolean, numeric, integer, or string.
    - “result” provides status information:
        - 0: token extracted
        - 1: no token at the given index
        - 2: empty token
- “string.format” creates a string using a format string and substitution parameters.

Events Reference

The following events may be used with the “on” condition to capture events.

- An ordinary “on” condition may have these properties:
- “on”: value is the event id
- “index”: if applicable, the event index to filter for.

Change events monitor a property and capture changes.

- “change”: identifies the property to monitor.
- “inside”/”outside”/”modal”: defines the boundary conditions to watch for
- “debounce”: prevents duplicate events.

Type	Event Name	Description	Parameters
Event Id	event.wq		
Event Index	event.power.up	Use with event.wq as Event Index. (Parameter 0). A WhereQube POWER_UP event occurred.	
Event Index	event.ignition.off	Use with event.wq as Event Index. (Parameter 0). A WhereQube IGN_OFF event occurred.	
Event Index	event.ignition.on	Use with event.wq as Event Index. (Parameter 0). A WhereQube IGN_ON event occurred.	
Event Index	event.alarm	Use with event.wq as Event Index. (Parameter 0). A WhereQube ALARM event occurred.	
Event Index	event.power.off	Use with event.wq as Event Index. (Parameter 0). A WhereQube POWER_OFF event occurred.	
Event Index	event.on.periodic	Use with event.wq as Event Index. (Parameter 0). A WhereQube ON_PERIODIC event occurred.	
Event Index	event.off.perodic	Use with event.wq as Event Index. (Parameter 0). A WhereQube OFF_PERIODIC event occurred.	
Event Index	event.poll	Use with event.wq as Event Index. (Parameter 0). A WhereQube POLL event occurred.	
Event Index	event.emergency	Use with event.wq as Event Index. (Parameter 0). A WhereQube EMERGENCY event occurred.	
Event Index	event.battery.warning	Use with event.wq as Event Index. (Parameter 0). A WhereQube BATT_WARN event occurred.	
Event Index	event.idling	Use with event.wq as Event Index. (Parameter 0). A WhereQube IDLING event occurred.	
Event Index	event.power.cut	Use with event.wq as Event Index. (Parameter 0). A WhereQube POWER_CUT event occurred.	
Event Index	event.begin.stop	Use with event.wq as Event Index. (Parameter 0). A WhereQube BEGIN_STOP event occurred.	
Event Index	event.end.stop	Use with event.wq as Event Index. (Parameter 0). A WhereQube END_STOP event occurred.	
Event Index	event.motion.alarm	Use with event.wq as Event Index. (Parameter 0). A WhereQube MOTALARM event occurred.	
Event Index	event.speeding	Use with event.wq as Event Index. (Parameter 0). A WhereQube SPEEDING event occurred.	
Event Index	event.fence.exit	Use with event.wq as Event Index. (Parameter 0). A WhereQube FENCEEXIT event occurred.	1:Fence id

Event Index	event.fence.entry	Use with event.wq as Event Index. (Parameter 0). A WhereQube FENCEENTRY event occurred.	1:Fence id
Event Index	event.heartbeat	Use with event.wq as Event Index. (Parameter 0). A WhereQube HEARTBEAT event occurred.	
Event Index	event.live	Use with event.wq as Event Index. (Parameter 0). A WhereQube LIVE event occurred.	
Event Index	event.heading	Use with event.wq as Event Index. (Parameter 0). A WhereQube HEADING event occurred.	
Event Index	event.distance	Use with event.wq as Event Index. (Parameter 0). A WhereQube DISTANCE event occurred.	
Event Index	event.cluster	Use with event.wq as Event Index. (Parameter 0). A WhereQube CLUSTER event occurred.	
Event Index	event.emergency.end	Use with event.wq as Event Index. (Parameter 0). A WhereQube EMERGENCY_END event occurred.	
Event Index	event.alarm.end	Use with event.wq as Event Index. (Parameter 0). A WhereQube ALARM_END event occurred.	
Event Index	event.io.change	Use with event.wq as Event Index. (Parameter 0). A WhereQube IO_CHANGE event occurred.	
Event Index	event.idling.end	Use with event.wq as Event Index. (Parameter 0). A WhereQube IDLING_END event occurred.	
Event Index	event.hard.stop	Use with event.wq as Event Index. (Parameter 0). A WhereQube HARDSTOP event occurred.	
Event Index	event.speeding.end	Use with event.wq as Event Index. (Parameter 0). A WhereQube SPEEDING_END event occurred.	
Event Index	event.hard.brake	Use with event.wq as Event Index. (Parameter 0). A WhereQube HARDBRAKE event occurred.	
Event Index	event.hard.turn	Use with event.wq as Event Index. (Parameter 0). A WhereQube HARDTURN event occurred.	
Event Index	event.hard.acceleration	Use with event.wq as Event Index. (Parameter 0). A WhereQube HARDACCEL event occurred.	
Event Index	event.mil.on	Use with event.wq as Event Index. (Parameter 0). A WhereQube MILON event occurred.	
Event Index	event.mil.off	Use with event.wq as Event Index. (Parameter 0). A WhereQube MILOFF event occurred.	
Event Index	event.vin	Use with event.wq as Event Index. (Parameter 0). A WhereQube VIN event occurred.	
Event Index	event.fuel.loss	Use with event.wq as Event Index. (Parameter 0). A WhereQube FUELLOSS event occurred.	
Event Index	event.refuel	Use with event.wq as Event Index. (Parameter 0). A WhereQube REFUEL event occurred.	
Event Index	event.moving	Use with event.wq as Event Index. (Parameter 0). A WhereQube MOVING event occurred.	
Event Index	event.dmchange	Use with event.wq as Event Index. (Parameter 0). A WhereQube DMCHANGE event occurred.	
Event Index	event.conn	Use with event.wq as Event Index. (Parameter 0). A WhereQube CONN event occurred.	
Event Index	event.disconn	Use with event.wq as Event Index. (Parameter 0). A WhereQube DISCONN event occurred.	
Event Index	event.bus.malf	Use with event.wq as Event Index. (Parameter 0). A WhereQube BUSMALF event occurred.	
Event Index	event.bus.malf.end	Use with event.wq as Event Index. (Parameter 0). A WhereQube BUSMALF_END event occurred.	
Event Index	event.idle.shutdown	Use with event.wq as Event Index. (Parameter 0). A WhereQube IDLESHUTDOWN event occurred.	
Event Index	event.cell	Use with event.wq as Event Index. (Parameter 0). A WhereQube CELL event occurred.	

Event Index	event.no.cell	Use with event.wq as Event Index. (Parameter 0). A WhereQube NOCELL event occurred.	
Event Index	event.ble.interrupt	Use with event.wq as Event Index. (Parameter 0). A WhereQube BLEINTR event occurred.	
Event Index	event.ble.init	Use with event.wq as Event Index. (Parameter 0). A WhereQube BLEINT event occurred.	
Event Index	event.sjdr	Use with event.wq as Event Index. (Parameter 0). A WhereQube SJDR event occurred.	
Event Index	event.sjdr.end	Use with event.wq as Event Index. (Parameter 0). A WhereQube SJDR_END event occurred.	
Event Index	event.on.periodic.2	Use with event.wq as Event Index. (Parameter 0). A WhereQube ON_PERIODIC2 event occurred.	
Event Index	event.tire	Use with event.wq as Event Index. (Parameter 0). A WhereQube TIRE event occurred.	
Event Index	event.pto.on	Use with event.wq as Event Index. (Parameter 0). A WhereQube PTOON event occurred.	
Event Index	event.pto.off	Use with event.wq as Event Index. (Parameter 0). A WhereQube PTOOFF event occurred.	
Event Index	event.door	Use with event.wq as Event Index. (Parameter 0). A WhereQube DOOR event occurred.	
Event Index	event.setpoint	Use with event.wq as Event Index. (Parameter 0). A WhereQube SETPOINT event occurred.	
Event Index	event.r.mil.on	Use with event.wq as Event Index. (Parameter 0). A WhereQube RMILON event occurred.	
Event Index	event.r.mil.off	Use with event.wq as Event Index. (Parameter 0). A WhereQube RMILOFF event occurred.	
Event Index	event.r.power	Use with event.wq as Event Index. (Parameter 0). A WhereQube RPOWER event occurred.	
Event Index	event.r.periodic	Use with event.wq as Event Index. (Parameter 0). A WhereQube RPERIODIC event occurred.	
Event Index	event.shock	Use with event.wq as Event Index. (Parameter 0). A WhereQube SHOCK event occurred.	
Event Index	event.postshock	Use with event.wq as Event Index. (Parameter 0). A WhereQube POSTSHOCK event occurred.	
Event Index	event.cast	Use with event.wq as Event Index. (Parameter 0). A WhereQube CAST event occurred.	
Event Index	event.driver.id	Use with event.wq as Event Index. (Parameter 0). A WhereQube DRIVERID event occurred.	
Event Index	event.prndl	Use with event.wq as Event Index. (Parameter 0). A WhereQube PRNDL event occurred.	
Event Index	event.charge.start	Use with event.wq as Event Index. (Parameter 0). A WhereQube CHGSTART event occurred.	
Event Index	event.charge.stop	Use with event.wq as Event Index. (Parameter 0). A WhereQube CHGSTOP event occurred.	
Event Id	event.timer	A timer timed out.	0:Timer id.
Event Id	event.transition	An action was performed on a state machine that caused a state transition.	0:State machine id. 1:State transitioning from. 2: action id. 3:state transitioning to.
Event Id	event.channel.closed	A data channel was closed.	0:Channel id. 1: Stream id. 2: Channel type.
Event Id	event.channel.reset	The data channel was reset.	0:Channel id. 1: Stream id. 2: Channel type.
Event Id	event.channel.receive	Data was received on the data channel.	0:Channel id. 1: Stream id. 2: Channel type. 3:On text channels, the received string.

Event Id	event.stream.written	A data channel stream was written to.	0:Channel id. 1: Stream id. 2: Channel type.
Change Monitor Event Id	event.change.accel	Monitors changes to the accelerometer with the given boundaries.	
Change Monitor Event Id	event.change.battery.device.engine.voltage	Monitors changes to the device voltage with the given boundaries.	
Change Monitor Event Id	event.change.battery.engine.voltage	Monitors changes to the engine voltage with the given boundaries.	
Change Monitor Event Id	event.change.comm.rssi	Monitors changes to the modem signal strength with the given boundaries.	
Change Monitor Event Id	event.change.device.voltage	Monitors changes to the device voltage with the given boundaries.	
Change Monitor Event Id	event.change.gpsfix	Monitors changes to the satellite fix count with the given boundaries.	
Change Monitor Event Id	event.change.heading	Monitors changes to the heading with the given boundaries.	
Change Monitor Event Id	event.change.input	Monitors changes to the gpio inputs with the given boundaries.	
Change Monitor Event Id	event.change.input.high	Monitors changes to the gpio inputs with the given boundaries.	
Change Monitor Event Id	event.change.input.low	Monitors changes to the gpio inputs with the given boundaries.	
Change Monitor Event Id	event.change.queue	Monitors changes to the store and forward queue with the given boundaries.	
Change Monitor Event Id	event.change.speed	Monitors changes to the ecu speed with the given boundaries.	
Change Monitor Event Id	event.change.speed.km	Monitors changes to the ecu speed in kilometers with the given boundaries.	
Change Monitor Event Id	event.change.speed.gps	Monitors changes to the gps speed with the given boundaries.	
Change Monitor Event Id	event.change.time.day	Monitors changes to the day with the given boundaries.	
Change Monitor Event Id	event.change.time.hour	Monitors changes to the hour with the given boundaries.	
Change Monitor Event Id	event.change.vin	Monitors changes to the vehicle vin.	

Read-Only Device Properties Reference

Property Name	Data Type	Explanation
event.id	Integer	Id of the currently processed event in the script event queue.
event.index	Integer	Specific identification of an event, for example, the exact WhereQube event or timer id.
event.arg	Per Event	Events optionally have arguments, indexed 0-3, to provide extra data, of varying types. Use the "sel" property to select an argument index.Consult the event list for specifics for each event.
timer.active	Boolean	True if the timer is active, otherwise false. Use the "sel" property to select a timer to check.
timer.value	Integer	Number of seconds left for the indicated timer. Use the "sel" property to select a timer to check.
time.utc	Integer	UTC Time in unix time format.
time.hour	Integer	Current hour, UTC.
time.minute	Integer	Current minute, UTC.
time.seconds	Integer	Current seconds, UTC.
time.month	Integer	Current month, UTC, 1 - 12
time.month.day	Integer	Current day of the month, UTC, 1 - 31
time.week.day	Integer	Current day of the week. UTC, 0 - 6
time.year	Integer	Current year, UTC.
time.year.day	Integer	Current day of the year, UTC.
accel.x	Float	Accelerometer x magnitude, in g's. This axis is considered the forward/back axis of the vehicle.
accel.y	Float	Accelerometer y magnitude, in g's. This axis is considered the side to side axis of the vehicle.
accel.z	Float	Accelerometer z magnitude, in g's. This axis is considered the up/down axis of the vehicle.
accel.mag	Float	Magnitude of accelerometer vector, in g's.
id.serialnumber	String	Serial number of the device.
id.imei	String	IMEI of the device.
id.iccid	String	ICCID of the device.
id.phonenumber	String	Phone number of the modem, if available.
dev.queue.count	Integer	Count of packets in the store and forward queue.
dev.dev.input	Integer	Bitmap describing GPIO states (high/low for each bit)
dev.awake	Boolean	True if the device is awake.
dev.snoozing	Boolean	True if the device is asleep.

dev.battery.voltage	Integer	Battery voltage measured in mv of the internal battery in Device
dev.ext.voltage	Integer	Battery voltage measured in mv measured by device
dev.temperature	Float	Current device temperature in Celsius.
dev.shaking	Boolean	True if the device detects shaking, false otherwise.
ble.client.mac	String	Bluetooth MAC address.
ble.client.connected	Integer	1 if connected, 0 if not connected.
ble.client.connected.seconds	Integer	Count of seconds bluetooth has been connected.
ble.mac	String	Bluetooth MAC address.
ble.state	Integer	1 if connected, 0 if not connected.
ble.version	Integer	Bluetooth version.
comm.cellid	String	Modem cell id.
comm.lac	String	Location area code.
comm.mcc	String	Mobile country code.
comm.mnc	String	Mobile network code.
comm.operator	String	Cellular carrier.
comm.band	String	Wireless band.
comm.ipaddr	String	Modem ip address.
comm.rssi	Integer	Signal strength.
comm.connected	Boolean	True if the modem is connected, false otherwise.
comm.disconnected	Boolean	True in not connected, false otherwise.
comm.roaming	Boolean	True if in roaming mode, false otherwise.
version.app	String	App firmware version.
version.ble	Integer	Bluetooth firmware version.
version.modem	Integer	Modem firmware version.
version.sup	Integer	Supervisor firmware version.
calc.odometer	Float	Odometer in miles from GPS.
calc.odometer.km	Float	Odometer in kilometers from GPS.
calc.roll.total.minutes	Integer	Roll time since moving in minutes
calc.roll.total.seconds	Integer	Roll time since moving in seconds
calc.ignition.total.minutes	Integer	Total minutes with ignition on.
calc.ignition.total.seconds	Integer	Total seconds with ignition on.
calc.idle.total.minutes	Integer	Total minutes idling.
calc.idle.totalseconds	Integer	Total seconds idling.
calc.idle.session.minutes	Integer	Idling session minutes.
calc.idle.session.seconds	Integer	Idling session seconds.
calc.ignition	Boolean	True if the vehicle ignition is on.
calc.moving	Boolean	True if the vehicle is moving.
calc.idling	Boolean	True if the vehicle is idling.
calc.moving.session.minutes	Integer	Moving session minutes.
calc.moving.session.seconds	Integer	Moving session seconds.
gps.raw.lon	Float	Raw longitude value (irrespective of satellite fix)
gps.raw.lat	Float	Raw latitude value (irrespective of satellite fix)

gps.raw.alt	Float	Raw latitude value (irrespective of satellite fix)
gps.raw.speed.kmh	Float	Raw altitude value (irrespective of satellite fix)
gps.raw.speed.mph	Float	Raw gps speed in mph.
gps.raw.speed.mps	Float	Raw gps speed in meters per second.
gps.raw.hdop	Float	GPS HDOP reading
gps.raw.heading	Float	Raw gps heading
gps.raw.fix	Integer	GPS satellite fix count.
gps.raw.valid	Boolean	Raw GPS, True if valid reading, false otherwise.
gps.raw.on	Integer	Non-zero if GPS on, 0 otherwise.
gps.raw.sats.inuse	Integer	Raw GPS satellites in use.
gps.raw.sats.tracking	Integer	Raw GPS satellites tracking.
gps.latest.lon	Float	Most recent validated gps longitude.
gps.latest.lat	Float	Most recent validated gps latitude.
gps.latest.alt	Float	Most recent validated gps altitude.
gps.latest.speed.kmh	Float	Latest valid GPS speed kmh.
gps.latest.speed.mph	Float	Latest valid GPS speed mph.
gps.latest.speed.mps	Float	Latest valid GPS speed mps
gps.latest.hdop	Float	Latest valid GPS HDOP
gps.latest.heading	Float	Latest valid GPS heading
gps.latest.utc	Integer	Latest valid GPS time.
gps.latest.age	Integer	Latest location age.
gps.latest.fix	Integer	Latest valid GPS satellite fix count.
gps.latest.sats.used	Integer	Latest valid GPS satellite use count.
onewire.all.count	Integer	Count of readings from onewire.
onewire.driverid	String	One wire driver id.
onewire.driverid.count	Integer	Count of driver ids.
onewire.temperature	String	Temperature reading from onewire.
onewire.temperature.count	Integer	Count of temperature readings.
onewire.temperature.C	Float	Onewire reading in celsius.
onewire.temperature.F	Float	Onewire reading in farenheit.
onewire.all	String	All onewire readings together.

Read-Only Engine Properties Reference

Property Name	Data Type	Explanation
eng.fuel.econ.cumulative.mpg	Integer	Cumulative fuel economy in hundredths of miles per gallon.
eng.fuel.econ.trip.mpg	Integer	Trip fuel economy in hundredths of miles per gallon.
eng.fuel.econ.instant.mpg	Integer	Instant fuel economy in hundredths of miles per gallon.
eng.fuel.level	Integer	Fuel level percentage.
eng.fuel.used.total	Integer	Total fuel used.
eng.fuel.used.idle	Integer	Total fuel used while idling.
eng.fuel.used.trip	Integer	Fuel used for current trip.
eng.gas.used.total	Integer	Total gas used for gaseous engines.
eng.gas.used.trip	Integer	Trip gas used for gaseous engines.
eng.rpm	Integer	Engine RPMs.
eng.throttle	Integer	Engine throttle level.
eng.voltage	Integer	Engine battery voltage.
eng.ambient.temperature	Integer	Engine ambient temperature.
eng.speed.miles	Integer	Engine speed in miles per hour.
eng.speed.km	Integer	Engine speed in kilometers per hour.
eng.speed.limit	Integer	Max speed limit set in ecu in kilometers per hour.
eng.odometer.miles	Float	Odometer in miles.
eng.odometer.km	Float	Odometer in kilometers.
eng.mil.status	Integer	Malfunction indicator light status.
eng.dtc.count	Integer	Diagnostic trouble code count
eng.dtc.distance.miles	Integer	Miles traveled
eng.dtc.distance.km	Integer	Kilometers travelled
eng.dtc.active	String	Active diagnostic trouble codes
eng.dtc.all	String	All diagnostic trouble codes
eng.dtc.dm01	String	Diagnostic Trouble Codes reported by ECU for J1939 Protocol
eng.dtc.dm02	String	Diagnostic Trouble Codes reported by ECU for J1939 Protocol
eng.vin	String	Vehicle Identification Number
eng.charge.level	Integer	Electric vehicle charge level
eng.phev.state	Integer	PHEV state data (packet variable 127)
eng.phev.voltage	Integer	PHEV voltage
eng.phev.current	Integer	PHEV current
eng.enginehours.hours	Integer	Engine hours in hours.
eng.enginehours.seconds	Integer	Engine hours in seconds
eng.idlehours.hours	Integer	Idling hours.

eng.idlehours.seconds	Integer	Idling time in seconds.
eng.ptohours.hours	Integer	PTO On hours.
eng.ptohours.seconds	Integer	PTO On seconds
eng.pto.state	Integer	PTO state
eng.oil.level	Integer	Engine oil level in tenths of a percentage.
eng.oil.temperature	Integer	Engine oil temperature in hundredths of degrees celsius.
eng.oil.life	Integer	Oil life.
eng.coolant.temperature	Integer	Coolant temperature.
eng.coolant.level	Integer	Coolant level in tenths of a percentage.
eng.def.level	Integer	Diesel exhaust fluid livel in tenths of a percentage.
eng.def.temperature	Integer	Diesel exhaust fluid temperature
eng.dpf.regen.inhibit	Integer	
eng.dpf.soot.load	Integer	Soot dpf build up 0 - 250%
eng.dpf.lastregen.seconds	Integer	Time since last regeneration in seconds
eng.dpf.soot.regen.threshold	Integer	1/10000 percentage
eng.dpf.status	Integer	Regeneration need level 0 - 11
eng.dpf.active.regen.status	Integer	Regeneration activity, 0 - 11
trans.oil.level	Integer	Oil level
trans.oil.temperature	Integer	Oil temperature in hundredths of degrees celsius
trans.gear	Integer	Gears -125 - +125
cruisecontrol.state	Integer	1=Hold,2=Accelerate,3=Decelerate,4=Resume,5=Set,6=Override
prndl.position	Integer	0 = Park (P), 1 = Reverse (R), 2 = Neutral (N), 3 = Drive (D), 4 = Low/Sport/Manual (L)
seatbelt.state	Integer	State of switch used to determine if Seat Belt is buckled. (0) NOT Buckled. (1) OK - Seat Belt is buckled. (2) Error - Switch state cannot be determined. (3) & (255) Not Available
tpms.all	String	Tire pressure management data (see variable 121)
tpms.faulty	String	Tire pressure management data for faulty tires (see variable 122)