**TeensyGPS Data Logger Software Description**

**Purpose**

The purpose of this document is to give a detailed description of the “TeensyGPS Data Logger”. It will explain the purpose, functionality and system constraints necessary for the software development team to develop the initial software libraries necessary for the TeensyGPS to be fully functional.

**Scope**

The “Teensy GPS Shield” is a module that interfaces with an Arduino Teensy 3.1 microcontroller and has 4 basic features:

1. MicroSD for Data Logging
2. GPS Receiver
3. 9DOF Sensor
4. CAN Bus Transceiver

These 4 features provide the essential data needed in many real world applications including RC aircraft, RC cars, UAVs, automotive racing, motor-cross and various other hobby and professional fields. In addition, having sufficient flexibility built into the data logging application would provide a system that could be useful for scientific studies such as measuring sea ice drift and at the same time being cost effective for the general hobbyist to use in RC vehicles.

The heart of the Teensy GPS Shield will be the Teensy 3.1 microcontroller. The Teensy microcontroller is Arduino compatible and all software will be written for the Arduino platform. Ultimately the software will be open source and available for the general public to download and modify. This software effort is being developed by multiple individuals and an existing set of libraries will serve as a starting point for any new developers brought onto the team. The existing set of libraries can be found here: <https://github.com/smokingresistor/TeensyGPS>

Of the 4 features available, first basic feature is the GPS receiver which receives the satellite signal from multiple satellites and calculates current latitude, longitude, altitude and provides other useful data. The GPS receiver is based around the SkyTraq Venus838FLPX receiver chip and the datasheet can be found here: <http://www.smokingresistor.com/wp-content/uploads/2014/06/Venus838FLPx_DS_v4.pdf> Configuration of the GPS receiver is achieved through serial commands that will be transmitted from the Teensy microcontroller. Configuration parameters will be transmitted to the Venus838FLPX receiver chip every time the device is powered on. The existing libraries for the Teensy GPS Shield on Github already have some of these parameters defined.

The second feature is the 9DOF sensor based around the ST Microelectronics LSM9DS0TR chip. This chip has sensor data available in all 3 axes for acceleration, angular rates and magnetic fields. Currently, the 9DOF sensor has not been integrated into the Teensy GPS Shield. However, an Arduino library for this particular LSM9DS0TR chip already exists and can be found here: <https://github.com/adafruit/Adafruit_LSM9DS0_Library>

The third feature is the CAN bus transceiver. The Teensy uses the FlexCAN library found at this link: <https://github.com/teachop/FlexCAN_Library> The Teensy communicates to a CAN bus transceiver chip part number SN65HVD232DR by Texas Instruments. The CAN bus interface is a DSUB 9 pin connector which is standard in most automotive CAN bus applications.

The final feature of the Teensy GPS shield is the data logging capability. The data from the GPS receiver and 9DOF sensor will be recorded to a MicroSD card. The GPS receiver has the capability of providing position data at a rate of 50Hz max, so data from both the GPS and 9DOF sensor will need to be recorded at the same 50Hz max rate. This data rate will be configurable via a config.json file that will be read from the MicroSD card at power on. In order to provide the data logging needs for a variety of industries, a lot of flexibility will be built into the data logging library. At this point, the Teensy GPS Shield library has basic logging capability for a few GPS parameters. Further development of the data logging application will be required.

In addition, to these 4 primary features available on the Teensy GPS Shield, there is an additional requirement to have filtered GPS data. Due to the inaccuracies of commercial GPS position data, the 50 Hz max data rate from the GPS receiver tends to be very noisy. To overcome this noisy GPS data, a Kalman filter has been added. The Kalman filter which has been implemented into the Teensy GPS Shield libraries is currently not operational. The software development team will be required to enable this filter and provide filtered GPS data as an output to the data logger.

**Data Logging Software Description**

The Teensy GPS data logger shall read the MicroSD card and use a JSON configuration file called “config.jsn” to configure the data logging parameters. The config.jsn file is a subset of the gpsd protocol standard found here: <http://www.catb.org/gpsd/gpsd_json.html> . The JSON file format is an open standard that uses human readable text and can be edited with any text editor. There are also many JSON viewers/editors available to make it easier to read and edit. The advantage of JSON is that it has convenient attribute-data pairs and is similar to XML but much simpler syntax for faster parsing.

The gpsd subset will be defined the config.jsn file as shown in tables 1 through 6. The user can choose which optional fields will be recorded.

CNF

The CNF object is a configuration object used to configure the Teensy GPS data logger. The information in this report will not be included in the data logs since it is input only.

**Table 6. CNF object**

| **Name** | **Always?** | **Type** | **Description** |
| --- | --- | --- | --- |
| class | No | string | Fixed: "CNF" for Venus838 configuration |
| can\_en | No | numeric | Enable Can Bus output 0: Disabled 1: Enabled |
| log\_en | No | numeric | Enable data logging to the MicroSD card 0: Disabled 1: Enabled |
| size | No | numeric | Maximum size for each data log file (in MB) with a max of 999MB. The minimum value is 1MB. |
| rate | No | numeric | GPS Position Update Rate with choices of 1, 2,4,5,8,10,20,25,40,50 Hz |
| log\_type | No | numeric | Three types of data logging are possible:  0=continuous 1=event (continuous data logging will begin once event trigger is reached) 2=interval (logging only occurs within the interval that has been defined) |
| trig | No | numeric | Event Triggers are used to enable data logging when an event occurs. The possible events are: 0: longitude 1: latitude 2: altitude (meters) 3: speed over ground (meters per second) |
| trigv | No | numeric | Event Trigger Value is used for single event triggering only. Once the value in this field is obtained, logging will be enabled |
| intv | No | numeric | Interval Triggers are used to start and stop data logging. The possible intervals are: 0: Time elapsed after valid position fix (seconds) 1: Distance traveled after valid position fix (meters) 2: Speed over ground after valid position fix (meters/second) |
| min | No | numeric | Minimum value to enable logging (ex: 60 seconds after valid position fix) |
| max | No | numeric | Maximum value to disable logging (ex: 3600 seconds after valid position fix) |
| newlog | No | numeric | Generate new log file when passing through a beacon point 0: Disabled 1: Enabled |
| blat | No | numeric | Beacon latitude |
| blong |  | numeric | Beacon longitude |
| btime | No | numeric | Beacon timeout (seconds). If you don’t reach the beacon location within the timeout limit, a new log file will be generated. |
| btol | No | numeric | Beacon tolerance (meters). This should be no less than 15 meters. |

TPV

A TPV object is a time-position-velocity report. The "class" and "mode" fields will reliably be present. The "mode" field will be emitted before optional fields that may be absent when there is no fix. Error estimates will be emitted after the fix components they're associated with. Others may be reported or not depending on the fix quality.

**Table 1. TPV object**

| **Name** | **Always?** | **Type** | **Description** |
| --- | --- | --- | --- |
| class | Yes | string | Fixed: "TPV" |
| device | No | string | Fixed: "Venus838" |
| mode | Yes | numeric | Fix mode: %d, 0=no fix, 1=2D, 2=3D, 3=3D+DGNSS |
| time | No | string | Time/date stamp in ISO8601 format, UTC. May have a fractional part of up to .001sec precision. May be absent if mode is not 1,2 or 3. |
| lat | No | numeric | Latitude in degrees: +/- signifies North/South. Present when mode is 1, 2 or 3. |
| lon | No | numeric | Longitude in degrees: +/- signifies East/West. Present when mode is 1, 2 or 3. |
| alt | No | numeric | Altitude in meters. Present if mode is 2 or 3. |
| latf | No | numeric | Kalman filtered latitude in degrees: +/- signifies North/South. Present when mode is 1, 2 or 3. |
| lonf | No | numeric | Kalman filtered longitude in degrees: +/- signifies East/West. Present when mode is 1, 2 or 3. |
| altf | No | numeric | Kalman filtered altitude in meters. Present if mode is 2 or 3. |
| track | No | numeric | Course over ground, degrees from true north. |
| speed | No | numeric | Speed over ground, meters per second. |
| speedf | No | numeric | Kalman filtered speed over ground, meters per second. |
| GDOP | No | numeric | Geometric dilution of precision. Scaling 0.01 |
| PDOP | No | numeric | Position dilution of precision. Scaling 0.01 |
| HDOP | No | numeric | Horizontal dilution of precision. Scaling 0.01 |
| VDOP | No | numeric | Vertical dilution of precision. Scaling 0.01 |
| TDOP | No | numeric | Time dilution of precision. Scaling 0.01 |

ATT

An ATT object is a vehicle-attitude report. It is returned by digital-compass and gyroscope sensors; depending on device, it may include: heading, pitch, roll, yaw, gyroscope, and magnetic-field readings.

The "class” field will reliably be present unless all other flags are set to false. Others may be reported or not depending on the specific device type.

**Table 5. ATT object**

| **Name** | **Always?** | **Type** | **Description** |
| --- | --- | --- | --- |
| class | Yes | string | Fixed: "ATT" |
| device | No | string | Fixed: "LSM9DS0TR" |
| time | No | numeric | Time/date stamp in ISO8601 format, UTC. May have a fractional part of up to .001sec precision. May be absent if mode is not 1, 2 or 3. |
| heading | No | numeric | Heading, degrees from true north. |
| pitch | No | numeric | Pitch in degrees. |
| yaw | No | numeric | Yaw in degrees |
| roll | No | numeric | Roll in degrees. |
| dip | No | numeric | Local magnetic inclination, degrees, positive when the magnetic field points downward (into the Earth). |
| mag\_len | No | numeric | Scalar magnetic field strength. |
| mag\_x | No | numeric | X component of magnetic field strength (gauss) |
| mag\_y | No | numeric | Y component of magnetic field strength (gauss) |
| mag\_z | No | numeric | Z component of magnetic field strength (gauss) |
| acc\_len | No | numeric | Scalar acceleration. |
| acc\_x | No | numeric | X component of acceleration (g) |
| acc\_y | No | numeric | Y component of acceleration (g) |
| acc\_z | No | numeric | Z component of acceleration (g) |
| gyro\_x | No | numeric | Angular rate (degrees/sec) about the X axis |
| gyro\_y | No | numeric | Angular rate (degrees/sec) about the Y axis |
| gyro\_z | No | numeric | Angular rate (degrees/sec) about the Z axis |
| quat1 | No | numeric | Quaternion 1 |
| quat2 | No | numeric | Quaternion 1 |
| quat3 | No | numeric | Quaternion 1 |
| quat4 | No | numeric | Quaternion 1 |
| temp | No | Numeric | Temperature (°C) |

The following is an example of a config.jsn file that will be read by the TeensyGPS data logger. By simply adding a true or false next to the key attribute determines whether this field gets logged to the MicroSD card. Any key attributes that are omitted will be considered false and not get logged.

{"class" : "CNF","log\_en" : "1","can\_en" : "0","newlog" : "0","rate" : "50","size" : "10","blat" : "","blong" : "","btime" : "600","btol" : "15","log\_type" : "0","trig" : "","trigv" : "","intv" : "","min" : "","max" : ""}

{"class" : "TPV","device" : "true","mode" : "true","time" : "true","lat" : "true","lon" : "true","alt" : "true","latf" : "true","lonf" : "true","altf" : "true","track" : "true","speed" : "true","speedf" : "true","gdop" : "true","pdop" : "true","hdop" : "true","vdop" : "true","tdop" : "true"}

{"class" : "ATT","device" : "true","time" : "true","heading" : "false","pitch" : "false","yaw" : "false","roll" : "false","dip" : "false","mag\_len" : "false","mag\_x" : "false","mag\_y" : "false","mag\_z" : "false","acc\_len" : "false","acc\_x" : "false","acc\_y" : "false","acc\_z" : "false","gyro\_x" : "false","gyro\_y" : "false","gyro\_z" : "false","temp" : "false"}

Once the config.jsn file has been read by the Teensy GPS, the data logging files will be written to the MicroSD card in comma separated variable (CSV) format and each log file will include a single header row. The single header row starts with the TPV message class, TPV parameters being logged, ATT class and the ATT parameters being logged. If there are parameters flagged “false” in the config.jsn file, these parameters will not be in the header row and their data will not be logged. Since the user has the option of choosing which parameters get logged, the header file will be the key to decoding the data contained within.

After the single header row is written, the data that follows will also be in CSV format and only that parameters that have been flagged as true will be written to the data log. An example of a data log will be as follows:

class,mode,time,lat,lon,alt,track,speed,climb,class,heading,pitch,roll,mag\_xTPV,2,2005-06-08T10:34:48.283Z,46.49829337,7.567411672,1343.127,10.3788,0.091,-0.085,ATT,14223.00,169.00,-43.00,2454.000TPV,2,2005-06-08T10:34:48.283Z,46.49829337,7.567411672,1343.127,10.3788,0.091,-0.085,ATT,14223.00,169.00,-43.00,2454.000TPV,2,2005-06-08T10:34:48.283Z,46.49829337,7.567411672,1343.127,10.3788,0.091,-0.085,ATT,14223.00,169.00,-43.00,2454.000TPV,2,2005-06-08T10:34:48.283Z,46.49829337,7.567411672,1343.127,10.3788,0.091,-0.085,ATT,14223.00,169.00,-43.00,2454.000

**Power-On Events**

When the Teensy GPS power on, here are the 3 possible scenarios that would occur on boot.

1. SD Card not detected: No data will be logged
2. No config.jsn file found: If the data logger cannot find the config.jsn file then it will create a config.jsn file based on the existing configuration settings found in non-volatile memory. This means that if you have been using the data logger and you insert a newly formatted MicroSD card, your system will maintain its current settings that had previously been stored in non-volatile memory.
3. Config file found: During power up, the data logger will look for a config.jsn file. If the file is found, the data logger will use those settings and overwrite any previously stored system settings in non-volatile EEPROM.

The data logger will create a new file each time it runs; the highest-numbered file is the most recent log. The file naming convention will be LOG00001.csv. Log files will increment with most recent number stored in non-volatile EEPROM. The max number of log files will be 65533 and then will roll over and start again with LOG00001.csv overwriting any files that currently exist with the same name. If the MicroSD card is full, then data will no longer be written to the MicroSD card.



**Figure 1**

**MicroSD Software Description**

The MicroSD card is connected to the Teensy microcontroller via an SPI bus. The SD card detect signal is connected to the Teensy on pin 15. The SPI bus is connected to the Teensy on pins 7, 8, 9, 14 as shown in Figure1. An SPI library already exists for the Teensy 3.x microcontroller and can be found here: <https://github.com/xxxajk/spi4teensy3>

**GPS Receiver Software Description**

The SkyTraq Venus838FLPX GPS receiver is connected to the Teensy microcontroller via pins 9 &10 as shown in Figure 1. The GPS receiver communicates with the Teensy via serial channel at 115,200 baud with 8 data bits, no parity and 1 stop bit. There will only be 1 command that needs to be sent from the Teensy to the Venus838 chip which will be the position update rate which varies from 1 Hz to 50 Hz and is defined in Table 5. After reading the desired position update rate from the config.jsn file, the update rate value will be written to the Venus838 flash configuration. Once the position update rate is sent to the GPS receiver, the Teensy will need to start processing the data from the GPS receiver. If there is no MicroSD card or there is an error reading the config.jsn file, then the command to change the position update rate will not be sent.

The data received from the GPS will be in a binary format and the binary message protocol is defined in this document: <http://www.smokingresistor.com/wp-content/uploads/2014/06/AN0028_1.4.31.pdf>

**9DOF Sensor Software Description**

The LSM9DS0 9DOF sensor connects to the Teensy via i2c bus on Teensy pins 16 and 17 as shown on Figure 1. An i2c library already exists for the Teensy 3.x microcontroller and can be found here: <https://github.com/nox771/i2c_t3> The 9DOF sensor will need to be initialized on boot up and configured to output the appropriate data as listed in Table 3. Once the 9DOF sensor is configured properly, the Teensy microcontroller will read the 9DOF sensor output at the same rate which it’s reading the GPS data. The 9DOF sensor data will then be logged to the MicroSD card based on the config.jsn file.

**CAN Bus Software Description**

The CAN bus transceiver chip SN65HVD232DR is connected to the Teensy via serial channel on Teensy pins 3 & 4 as shown in Figure 1. The data transmitted to the CAN transceiver shall be fixed at 500kbps. The data that will need to be transmitted from the Teensy to the CAN bus transceiver will be defined in the config.jsn file found on the MicroSD card. Hence the data logger and the CAN bus transceiver will share the same configuration file and all data that gets sent to the MicroSD card for logging will also get sent to the CAN bus transceiver.

**Kalman Filter Software Description**

The Kalman filter will be implemented in the Teensy microcontroller and will be used to filter the GPS data. Specifically, the Kalman filter will be applied to Latitude, Longitude, Altitude and Speed. These four filtered parameters will only get logged to the MicroSD card if it is defined in the config.jsn file.