Pawel Gburzynski

The DOGS praxis

(preliminary document)

Version 0.3 (CC1350)

September 15, 2021

Preamble

The role of this application is to demonstrate the operation of the set of sensors onboard the CC1350 SensorTag (aka CC1350STK) and provide a generic tool for collecting sensor data from the device without having to connect it over a wire to a PC. It is intended for experiments where possibly large (relatively speaking) sets of data must be collected and stored for off-line analysis and interpretation.

This document is intentionally sketchy. Detailed information regarding the PCBs, the sensors, the PicOS platform, the RF interface, is contained in other documents by Texas Instruments and/or Olsonet Communications. Those documents are publicly available and can be easily obtained, e.g., from me.

The devices

The setup involves two devices: a CC1350STK (see Figure 1) called the Tag in the sequel, and a LAUNCHXL-CC1350 (Figure 2) called the Peg and acting as the wireless interface between the Tag and the PC. The Tag is powered from a battery and can be freely moved around within the Tag-Peg transmission range, which is of order 100 m in open space. The wireless communication channel is neither BT nor WiFi. It is built around the so-called proprietary mode of CC1350 and operates within the 916 MHz ISM band.

Figure 1: CC1350STK

The Tag is typically stored in a dormant state in which it uses practically zero battery power. There is no OFF switch, as such, on the device; however, the deepest sleep mode of the microcontroller effectively amounts to an OFF state because the battery drain is then negligible. To wake up the device from the dormant state, one should press Switch 1 (see Figure 3). The device will then reset to an active state, and the LED will blink twice.

Note that powering the device up (say by inserting the battery or connecting it via the USB debug interface) will cause it to go immediately dormant. Thus, pushing Switch 1 is necessary to bring the Tag to life after that. This way Switch 1 does its best to emulate a power switch.

Pressing Switch 1 while the device is active will have no effect unless the switch is held pressed for about 5 seconds. Then the LED will blink very quickly several times and the device will enter the dormant state. The switch should be released when the LED starts blinking. Otherwise, if the switch is still pressed when the LED is done blinking, the device will reset but, finding the switch pressed upon restart, it will become active.

Switch 2 is not used at present. Switch 1 is the only means to control the device manually

Figure 2: LAUNCHXL-CC1350

(without connecting to it from the Peg over the RF link). Note that the Tag can also be put to sleep remotely by a command from the Peg.

The Tag uses most power when sending data over the RF channel. With the transmitter continuously turned on, the device drains about 10 mA of current (at 3 V) which means that a 1000 mAh battery will last for about 4 days. In my test, the Tag streamed continuously (at 128 samples per second) transmitting the data to the Peg for 226 hours on two (no name) AA-type batteries. CRC 2032 batteries, like the one shown in Figure 1) come in different qualities, with the nominal capacity typically advertised as ~200 mAh. That would translate into over 15 h of continuous streaming, although (at this current drain) figures in the ballpark of 2-3 hours are more likely.[1]

If the radio is not used for 30 seconds (meaning there is no communication in either direction), the Tag enters the so-called WOR mode where it turns the radio off for some time, then listens for a short while, and so on, in a reasonably frugal duty cycle. In this mode, the battery will last for several months.[2] A special request from the Peg is needed to activate the Tag in the WOR mode. The action takes about three seconds.

When the Tag is activated (after the initial push of Switch 1), it enters the fully attentive mode, but if nothing happens for 30 seconds, it will transit to WOR. This will also happen when the Peg stops conversing with the Tag. For long-time storage, to make sure that the battery drain is minimized, the Tag should be put to sleep (by pressing Switch 1 for 5 seconds). Of course, in that mode the Tag will not respond to remote commands from the Peg.

The sensors

The Tag is equipped with 5 sensors:

- MPU9250: accelerometer, gyro, compass, temperature combo
- HDC100: humidity and temperature combo
- SPH0641LM4H-1: one bit microphone

[1] CRC 2032 batteries are typically tested under lower discharge current. Here is a quote from a manufacturer's sheet: "210mAh (on continuous discharge at 20℃ under 15kΩ load to 2.0V end-voltage)". My quick test using a new battery that I pulled out of my drawer yielded about 2 hours of continuous streaming. This is a very poor score suggesting that CRC 2032 is not a good battery for experiments and, probably, not a good battery for the target application.

[2] Experiments are needed for a better estimate. AA batteries should last over a year in this mode.

- OPT3001: ambient light sensor
- BMP280: air pressure and temperature combo

Technical details are provided in separate documents.[3] Here I only include rudimentary description needed to understand the functionality of the application at hand. From now on, the sensors will be referred to by their selectors, i.e., names used in the application, which are, respectively: IMU, HUMIDITY, MICROPHONE, LIGHT, PRESSURE.

Three sensors, IMU, HUMIDITY, and PRESSURE, combine multiple functions (they consists of multiple, selectable, semi-independent components). IMU is the most prominent (most complex) representative of this group. Note that there are three different temperature components.[4] Not a big surprise: many sensors provide temperature components on the side, so this is expected when you have a random bunch of them.

The sensors can be independently configured. Also, for the three sensors consisting of multiple components, the components can be selected independently. Then, each sensor can be independently turned on and off. The operations of configuring a sensor and turning it on and off are separated.

The present application is a bit messy because the program running in the Tag caters to numerous tests and implements miscellaneous hooks that I needed to try out (experiment with) various things. Things can all be cleaned up when (if) we know what we want. If any *actual* functionality is expected from the thing, it is easier to remove and trim than to create and refine.

Sampling and streaming

There are two ways to collect sensor data which we shall refer to as *sampling* and *streaming*. Sampling is available for all sensors, and for combinations of their selections at once, meaning that groups of data coming from different sensors can be sampled simultaneously as compound samples. Streaming is presently only available for the accelerometer component of the IMU. It provides for high-reliability, timed (synchronous) data collection from the accelerometer and is intended for research in canine behavior. Streaming looks like an extension of sampling that went a bit sideways and has been implemented as a separate function.

For sampling, the sensors can be configured and turned on independently. Usually, a sensor is configured before being turned on, and it cannot be configured while being turned on, so the drill is to turn the sensor off, configure (or reconfigure) it, and turned it on again. The action of configuring basically prepares a set of parameters to be used when the sensor is turned on, so the operation of turning the sensor on is simple (and takes no parameters). There is a shortcut for starting the streaming operation where the IMU (the only sensor that can be streamed) is configured and turned on at the time when the streaming is commenced.

For sampling, we go through these steps:

1. configuring the sensors,
2. turning on those sensors that we want to deliver data (contribute to the sampling),

[3] As mentioned earlier, there are two levels of such documents: 1) PicOS documentation (device drivers), 2) sensor datasheets provided by the manufacturer.

[4] There is a fourth, i.e., the temperature sensor implanted into the microcontroller itself. Note that those sensors return the die temperature, as opposed to the ambient temperature, but in many cases the die temperature directly reflects the ambient temperature, e.g., if the sensor has been idle prior to the measurement.

3. starting sample collection,

Following step 3, all the sensors that have been turned on will be sampled according to their configuration sending data over the RF channel to the Peg. The operation will continue until explicitly stopped by a command from the Peg.

The command to start sampling accepts the sampling rate as its single argument. The amount of data arriving in a single sample depends on the set of sensors that have been turned on and the selection of their components. For example, each of the three main components of the IMU, i.e., accelerometer, gyro, and compass, sends triplets of 16-bit values interpreted as vectors in 3-space, and the temperature component sends a single 16-bit value. Thus, if all four components of the sensor are selected, the data contributed by the sensor amounts to $3 \times 3 \times 2 + 1 \times 2 = 20$ bytes. If multiple sensors are on, e.g., the IMU (configured with its all components) and the LIGHT sensor, then the data sent in a single sample will amount to $20 + 4$ bytes (the additional four bytes contributed by the LIGHT sensor). The readings arriving in a sample are tagged with sensor identifiers, so they can be interpreted without the knowledge of sensor configuration.

The sampling rate is specified as the number of samples per minute. It is not extremely precise (the strobing clock is not accurate), although the Tag tries to adjust the actual rate dynamically to maintain a consistent long-term rate (so the long-term rate will tend to converge to the specified rate). The maximum rate that can be specified is 15360 samples per minute, i.e., 256 samples per second, with the effective reachable rate of about 160 samples per second. The minimum rate is 1 sample per minute.

For streaming, the sampling frequency is precise and determined by a sensor parameter (as explained below).

IMU modes

The IMU can operate in two (or three, depending how one counts) modes. In the passive mode, the sensor only returns the readings of its selected components when explicitly polled for the next sample, according to the schedule determined by the sampling command in effect. This is also how all the remaining sensors *always* behave if they are on during sampling.

Another sampling mode of the sensor is the *motion detection* mode. When the IMU is configured this way, the accelerometer becomes its only active component, and the sensor triggers events on a configurable acceleration threshold. Configuring the IMU in this mode automatically selects the accelerometer component and disables the remaining components. The data sent by the sensor consists of the acceleration vector (3×2 bytes) augmented by two more bytes (8 bytes total). The extra two bytes amount to a 16-bit unsigned integer value returning the number of motion events triggered from the last readout.

An additional option available in the motion detection mode is the *report option*. When the option is off, the motion data is only collected via scheduled sampling, as for any other sensor. If the report option is on (note that the option is only effective in the motion detection mode), a motion event will automatically trigger a spontaneous sample report sent to the Peg. That sample arrives out of band and contains the IMU data only (8 bytes, as explained above), even if other sensors are also active (their values being collected and included in regular samples).

The third mode of the IMU sensor is used exclusively in streaming (one may call it the *synchronous* data extraction mode). In this mode, the sensor triggers events at prescribed regular intervals when new data becomes ready for acquisition. The mode provides the

Tag with precise strobes driving the streamed samples, so the collection rate can be accurate. The available rates start at the high end of 1024 samples per second and go down to 4 samples per second. They are internally determined by a *divider* of the base rate of 1024 Hz, whereby a value of D between 0 and 255 indicates the number of ticks of the base clock to be skipped before triggering the event. For example, when $D = 3$, the rate is $1024/(3 + 1) = 256$ (the event occurs on every fourth tick of the base clock). Similar to the motion detection mode, the only active component of the IMU in the streaming mode is the accelerometer. The values arriving from the sensor are the three coordinates of the acceleration vector reduced to 10 bits each and sent in packages (blocks) of 12 readings. The blocks are sent using a special communication protocol that tries to compensate for occasional packet losses in the radio channel with retransmissions. How far we can go with effective streaming rates is to be determined (we need to experiment with different RF rates, distances, as well as the parameters of the streaming protocol), but something like 128 (reliable) samples per seconds appears to be easily achievable without trying too hard.

HUMIDITY

The sensor comes with two components: the *actual* humidity sensor and a temperature sensor. Either component returns two bytes of data, i.e., a 16-bit value. Once turned on, the sensor is internally sampled by the Tag at the interval specified as one of the configuration parameters (even if no scheduled sampling is in progress). The same approach is followed for PRESSURE and LIGHT. The official (scheduled) sample returns the sensor's last internally sampled value. The role of this internal pre-sampling is to:

1. Prevent the delays in physical sampling (readouts) of the sensor, which may be significant for slow sensors (whose values are not meant to change very fast), from impacting the scheduled sampling. Generally, it is assumed that scheduled sampling is delay-free.
2. Decouple the readouts of slow sensors from scheduled sampling, such that those readouts can be made formally available at high rates (at which one may want to sample other sensors) without overtaxing the slow sensors. This is in fact a slightly different formulation of point 1.

PRESSURE

Like HUMIDITY, the sensor consists of two components: air pressure and temperature. Each component returns a 4-byte (32-bit) value.

LIGHT

The sensor has a single component and returns two 16-byte values: light intensity and status.

MICROPHONE

The sensor is a standard one-bit microphone which I have crudely transformed into a counter accumulating the imbalance between zeros and ones in consecutive 8-tuples of bits (bytes) acquired from the microphone. This imbalance can be interpreted as a measure of acoustic noise in the neighborhood. The value returned by the sensor consists of two unsigned 32-bit numbers (8 bytes total). The first value returns the accumulated imbalance counted from the previous readout (sample), the other returns the total number of bits received from the sensor since the previous readout was made. The imbalance should be interpreted relative to the total bit count.

The present driver for the MICROPHONE sensor is a stub opening the sensor for potentially interesting applications, like barking recognition, to be implemented later.

Setting things up

For GUI development and testing, the Tag can be powered from a USB cable through the DevPack interface (see Figure 4). This is a small board about the same size as the Tag PCB which can be connected (piggybacked) *onto* the Tag board to provide access to the Tag for programming and debugging. When powered through DevPack, the Tag behaves in the same way as when powered from the battery.

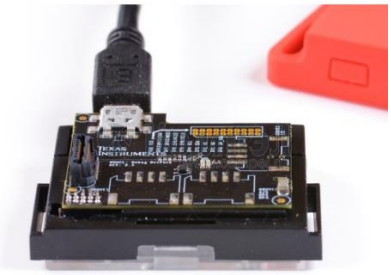

Figure 4. CC1350STK with attached DevPack board

The Peg must be connected to a PC to be operable. The USB interface of the Peg appears as two serial ports on the PC. Under Windows, these are COM ports; under Linux (say Ubuntu or Fedora), the devices are named /dev/ttyUSBACMx where x is a digit. One of those ports is for programming and debugging, the other maps to the UART of the Peg's microcontroller. This is the port that the program run on the computer and talking to the Peg is going to use to communicate with the device.

Both the Tag and the Peg have been programmed in PicOS. The present interface to the Peg consists of a command-line program (a Tcl script) extended with a simple GUI (Tk widgets). The command-line script is intended to provide full access to all capabilities of the Tag, while the GUI can only be used for streaming.

The script consists of two parts. The first (main) part is the generic OSS module whose intention is to provide a framework for interfacing OSS (Operational Support System) programs to PicOS master nodes. The second part extends the generic module by defining the specific set of commands and messages for the application. For the project at hand, the two parts come together in two files: ossrun.tcl (this is the generic part) and ossi.tcl (the specific part) that must be present the same directory.[5] The whole thing is run by executing:

```
./ossrun.tcl
```

The "main" script will find the specific part and "evaluate" it in the proper context. The script can run on Windows (e.g., under ActiveState Tcl/Tk) as well as on Linux (e.g., Ubuntu) using the standard Tcl/Tk packages.

The interface

When invoked properly (see above), the script will produce a window looking as shown in Figure 5 (this is how the window appears on Windows). The script must connect to the

[5] This is all described in separate documents. We have an SDK (development platform) for PicOS where the ossrun.tcl part belongs to the platform and the "project" provides its specific ossi.tcl part. The whole thing can be run independently of the platform by putting the two parts together.

Peg before we can start conversing with the setup. The present version is set to automatically connect to the Peg on startup. If the Peg is available, the connection should be automatic, i.e., the script identifies the serial device into which the Peg's UART has been mapped by polling all serial ports for a known response code. The button in the bottom right corner can be used to disconnect from (or reconnect to) the Peg.

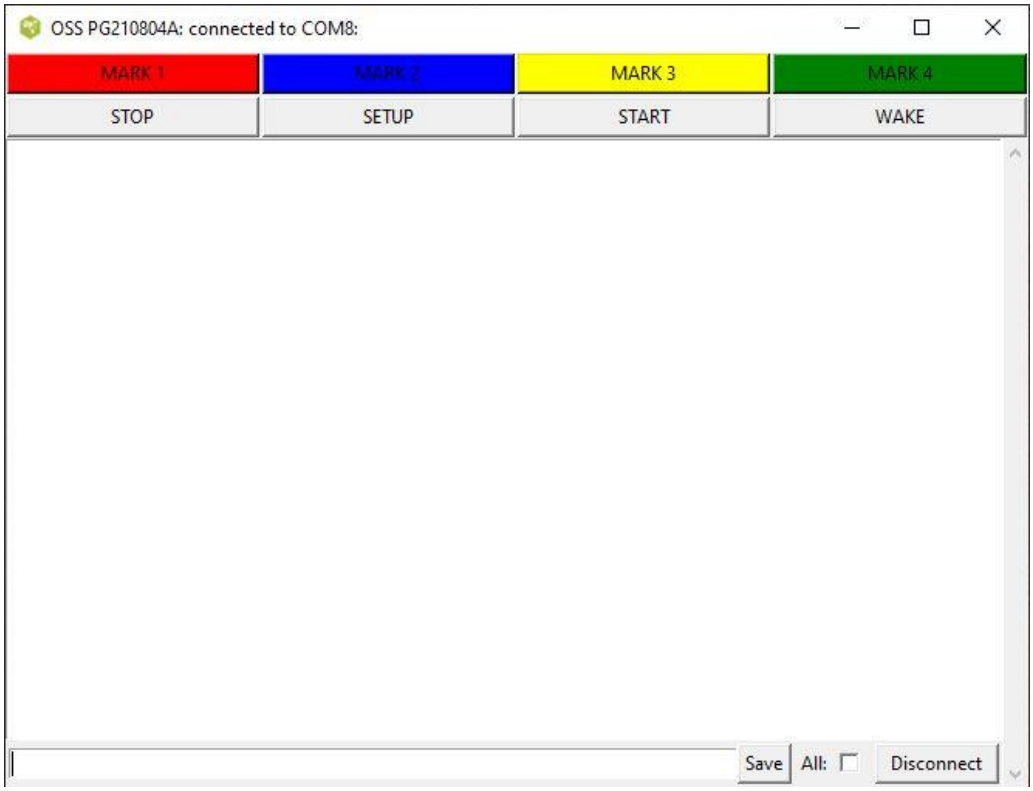

Figure 5. The OSS window

Once a connection has been established, one can converse with the Peg (and thus with the Tag) by entering commands into the input field in the left bottom area of the window. The buttons at the top can be used for easy streaming without delving too much into the technicalities of the sensor interface. We shall start from the command line interface.

Command syntax

A command starts with a keyword which can be followed by *selectors* and *parameters*. A selector is a word that *selects* something, e.g., a sensor. A parameter is a pair: –name value. The name of a parameter is preceded by – (the minus sign). The value of a parameter can be a character string, representing a selection of options, or a number.

Here is a sample command:

configure imu –range 1 –components at

The piece of string following the command keyword is a *selector*: it selects the sensor (IMU) to which the subsequent parameters apply. The value of –range represents one

numerical parameter of the sensor, specifically the so-called full range setting. The value of –components is the string “at” indicating two components of the sensor: the [a]ccelerometer and the [t]hermometer.

Any keyword can be abbreviated as far as to be still distinguishable from another keyword that might legally appear in its place. In particular, the above command can be shortened to:

c i –ran 1 –c at

The reason why –range cannot be shortened to –r (or even to –ra) is that –rate is also a legit parameter for the sensor. Note that “at” is not a keyword but a string of letters indicating the sensor components.

Practically all parameters are optional. When an optional parameter is not specified, it means either “default” or “unaffected”, i.e., retaining the last set value.

Command interpretation

Most commands issued to the Peg are directly transformed into commands to the Tag and expedited by the Peg over the RF channel. Two exceptions are: “ap”, which is intended exclusively for the Peg (no message is sent to the Tag at all), and “wake”, which causes the Peg to issue a wake-up sequence to the Tag (which takes more than simply forwarding the original request).

The packets exchanged between the Peg and the Tag are identified with a specific 16-bit Tag Id. It is possible to use the same Peg to talk to Tags belonging to different setups by changing the Tag Id with the “ap” command.

As explained earlier, the Tag will switch to the WOR mode if there has been no communication from the Peg (and no outgoing traffic from the Tag) for 30 seconds. In the WOR mode, the Tag will not respond to any command[6] except for “wake” which is intended to wake the Tag up and switch it to the fully active (receptive) mode (at least for the next 30 seconds). Thus, when we see that the Tag does not respond, it makes sense to try “wake” to bring the Tag into the receptive mode. Note that this command is also available from a button in the upper frame of the window.

To increase the reliability of communication (in the receptive mode), the Peg can be set to repeat every command a prescribed number of times (see below). This costs little and comes at no risk that the same command will be interpreted multiple times by the Tag, because the commands are numbered.

The Tag is expected to respond to every (non-duplicate) command, even one that requests it to become dormant. If the command, as seen by the Peg, succeeds (meaning the Tag has responded indicating success and the Peg has received the response), we should see the text “<OK>” in the OSS windows. The lack of (any) response usually means that the Tag can’t be reached or is irresponsive (because it is too far, off, dormant, or in the WOR mode). Responses other than “<OK>” (encapsulated in < … >) indicate errors or problems.

The traffic between the Tag and the Peg involves commands (sent by the Peg to the Tag), responses (ACK codes sent by the Tag to the Peg in reply to commands), and messages sent by the Tag to the Peg, e.g., containing sampled (or streamed) data from the sensors.

[6] Strictly speaking, the Tag will be *unlikely to respond* to a command because it would have to fall into the narrow window when the Tag’s RF module is active. The Tag will exit from WOR (at least for the next 30 seconds) when it receives any packet within that window.

All messages arriving from the Tag are shown in the OSS window in a formatted and legible fashion. This is what the “Save” button is for. When the “All” box is additionally checked, the saved data also includes the commands typed in the input box at the bottom of the OSS window.

Streaming data is always directed to a file which is separate from the (optional) save file for the contents of the OSS window (see below). The contents of the streamed blocks are not shown (as such) in the window, but the block numbers and the “end-of-train” messages (indicating the boundaries of acknowledgeable chunks of data) are.

Here are two (sometimes useful) extra features of the command input interpreter:

1. The exclamation sign (!), entered as a complete command, has the effect of re-entering the previous (last-typed) command.
2. If the string typed in as a command starts with a colon (:), the remaining portion of the string will be executed as a Tcl script within the context of the interpreter (this may be useful for debugging).

Command: ap

This command (“ap” stands for “access point”) is the only command addressed exclusively to the Peg. Its syntax is:

ap –node *nn* –retries *nr* –loss *pl*

All parameters are optional, and their values are nonnegative numbers. Their meaning is as follows:

node	Specifies the Id of the Tag (as explained above). The parameter can be used to switch among the different Tags serviced by the same Peg. The default Tag Id is
retries	The number of transmissions for a regular command send to the Tag. Zero and one both mean a single attempt. The default is 2, i.e., every command is transmitted twice.
loss	This parameter is for testing (emulating) packet losses in the Tag → Peg direction in those situations when they cannot be easily triggered in the natural way, e.g., by separating the two devices physically.[7] The default is zero. The parameter gives the average number of Tag packets per 1024 to be randomly dropped (ignored) by the Peg.

If the command is issued without arguments, it polls the Peg for the current setting of the above parameters.

Command: wake

This command takes no arguments. It instructs the Peg to issue a sequence of wake packets to the Tag intended to switch the Tag from WOR to the fully attentive mode.

Technically, the WOR mode at the Tag is implemented by a duty-cycle loop where the receiver is turned on for a short interval (about 10 ms) to listen for a packet, then turned off for about 1.5 s, and so on. The duration of the receive cycle is about 0.7% which means that the effective average current drain in the WOR mode is below 0.1 mA. Any packet received during the short reception period will trigger an exit from the WOR mode, but the

[7] The parameter is intended for testing the streaming protocol.

chance that a random packet will make it through is slim. In response to “wake”, the Peg will quickly send a back-to-back sequence of short packets, taking about 2 ms each, for about 2 seconds. This practically guarantees that at least one of those packets falls into the 10 ms reception interval.

Command: configure

The command configures a sensor (or several sensors) at the Tag. The list of arguments consists of a sensor selector (one of: imu, humidity, microphone, light, pressure) followed by the list of parameters specific to the sensor. Any sensor selector can be abbreviated to a single letter.

When issued without arguments, the command polls the Tag for configuration information on all sensors. This information is presented in the OSS window in a self-explanatory manner.

The command is cumulative in the sense that the parameters not mentioned with the current command retain their last settings. When the command is entered without arguments it polls the Tag for the current setting (parameter values) of all sensors. That information will arrive from the Tag as a packet whose contents will be presented (hopefully in a legible fashion) in the OSS window.

A single “configure” command can apply to multiple sensors. The list of parameters of a sensor can be followed by another sensor selector,[8] followed in turn by the list of parameters for the sensor. For example, this command:

```
conf imu –options mr –thresh 32 humid –opt h –com h
```

configures the sensors IMU and HUMIDITY in a single go.

A sensor parameter can be of one of two types:

option	Such a parameter is a string where every letter selects an option. Some options may be exclusive or incompatible. This is not diagnosed at the time the command is entered, but the sensor defines rules for prioritizing and reconciling conflicting options. For example, a component selection for a multiple-component sensor is an option parameter. In that case, all options are independent and cause no conflicts.
number	This is a nonnegative integer value whose minimum and maximum depend on the specific parameter.

Below we explain the configurations of parameters for every sensor to the extent required to understand the operation of the configure command. More detailed information about configuring the sensors can be found in the documents mentioned earlier.

One parameter, –sampling, is applicable to all three “sampled” sensors: HUMIDITY, LIGHT, and PRESSURE, and refers to the background sampling frequency of the sensor (as described earlier). Valid values are between 1 and 8192 (inclusively). The parameter is interpreted as the number of PicOS milliseconds separating two consecutive samples. The largest inter-sample interval is 8 seconds. Note that, in contrast to the remaining parameters, –sampling does not map to a physical parameter of the respective sensor.

[8] Note that a sensor selector is not preceded by –, so it cannot be mistaken for a parameter.

IMU

name	type	description
options	o	The legitimate option letters are: l (this is el like in llama) standing for low-power, s for synchronous, m for motion detection, and r for reports. Options s and m are exclusive. If both are selected then m takes precedence over s., Option r is only meaningful together with m and selects motion report events, as described earlier. Option s select synchronous read (needed for streaming) where the sensor generates events on data availability. Option l is automatically forced by m (so it's selection together with m is redundant). The default value is none.
threshold	n	Sets the threshold for motion detection. This parameter is only relevant together with the m option (see above). Legitimate values are between 0 and 255 and directly represent the acceleration threshold in 4 mg increments. The default value is 32.
lprate	n	Selects the internal sampling rate of the sensor when operating in the low-power mode (including motion detection). Legitimate values are from 0 to 11 and translate into the following rates in Hz: 0.24, 0.49, 0.98, 1.95, 3.91, 7.81, 15.63, 31.25, 62.5, 125, 250, 500. The default value is 6 corresponding to 15.63 Hz.
range	n	This is the so-called full range setting (sensitivity) which comes in four options. For the accelerometer, they are 2, 4, 8, and 16 g, corresponding to the parameter values from 0 to 3. The default value is 0 corresponding to 2 g.
bandwidth	n	This parameter selects the bandwidth of the low-pass filter. Its value is between 0 and 7 (eight steps) translating into 5-460 Hz. The default value is 3 corresponding to 41 Hz.
rate	n	This parameter selects the strobing rate for reading the accelerometer output while streaming, i.e., the frequency of events generated by the sensor with the s option in effect (see above). The specified value is between 0 and 255 and it is interpreted as the divisor of the basic 1024 Hz rate – 1. For example, the default value of 7, translates into 128 Hz = 1024/(7+1). The parameter does not apply when the sensor operates in the low power mode (the l option is selected, see above). Then, the strobing rate is directly determined by lprate, and the rate setting is ignored.[9]

[9] Streaming is currently not available in the low power mode. The Tag app must be reparametrized (recompiled) for that.

components	o	Selects the sensor's components. The legitimate option letters are a, g, c, t. When the sensor is turned on, at least one component must be selected. By default, if no components have been explicitly selected, the accelerometer is selected as the only component. The s options implies that any components other than the accelerometer are automatically deselected. The default value is "a".

Defaults: none, 32, 6, 0, 3, 7, a

HUMIDITY

name	**type**	**description**
options	o	The single legitimate option is h selecting the sensor's internal heater.
accuracy	n	Selects one of four accuracy levels from 0 to 3.
sampling	n	This is the external sampling frequency, as described earlier.
components	m	Selects the components which can be h (humidity) and t (temperature).

Defaults: none, 1, 4096, h

MICROPHONE

name	**type**	**description**
rate	n	The frequency of bit sampling from 100 to 2475 directly translating into kilohertz.

Defaults: 1500 (corresponding to 1.5 MHz)

LIGHT

name	**type**	**description**
options	o	The single legitimate option is c (for continuous). It selects continuous internal sampling as opposed to on-demand calculation.
accuracy	n	Selects one of two accuracy levels 0 or 1.
sampling	n	The external sampling frequency, as explained above.

Defaults: none, 0, 4096

PRESSURE

name	**type**	**description**

options	o	The single legitimate value is f selecting the forced mode for internal operation of the sensor where the sensor only responds to direct readouts. In normal mode, the sensor constantly evaluates the pressure, and its response is faster and more accurate. The rate and bandwidth parameters (see below) only apply if f is not set.
accuracy	n	Selects the accuracy of the result determined by the internal oversampling rate in five steps from 0 to 4.
rate	n	Selects the internal sampling rate for the sensor (the reciprocal of the standby interval) in discrete steps between 0 and 7.
bandwidth	n	Selects the filtering rate used to smooth out the results based on previous values in five discrete steps from 0 to 4.
sampling	n	The external sampling rate, as described earlier.
components	o	Selects the components of the sensor. The legitimate letters are p (for pressure) and t (for temperature).

Defaults: none, 0, 2, 2, 4096, p

Command: on

The command turns the indicated sensors on. The syntax is:

on *sen sen … sen*

where the arguments are sensor selectors. Initially all sensors are off. If no sensors are specified, then *all* sensors are turned on.

Command: off

The command turns the indicated sensors off. The syntax is:

off *sen sen … sen*

where the arguments are sensor selectors. If no sensors are specified, then *all* sensors are turned off.

Command: status

The command polls the Tag for its status. It takes no arguments. The message arriving from the Tag in response returns this information (which is shown in the OSS window with legible headers):

1. The up time of the Tag in seconds, i.e., the number of seconds elapsed since the Tag was last reset.
2. The battery voltage (for a healthy battery it should be around 3V).
3. The loss status from the last streaming operation. These are four values labeled F, M, Q, and P (see below).
4. Memory usage in longwords (for the heap): current free (F) and minimum free (M) seen so far (probably not very relevant).
5. Current activity status of the Tag which can be IDLE, SAMPLING, or STREAMING.

6. The list of active sensors, i.e., the ones that are currently on.
7. The collection status from the current (last) sampling or streaming operation. These are two numbers as explained below.

Item 3 is only relevant if the Tag has finished a streaming session and returns information about losses perceived both by the Tag and the Peg. F is the number of FIFO overflow events in the sensor, i.e., moments when the node couldn't retrieve data from the sensor on time. This number is always zero unless there's a bug in the Tag's firmware. M is the number of malloc failures, i.e., situations when the Tag was unable to allocate memory for an outgoing block of sensor data. This number should also be normally zero. Q is the number of cases when the Tag had to drop an unacknowledged packet (data block) because there was no room to accommodate new packets. All three event types represent losses recognizable by the Tag with the last one possibly resulting from the high error rate in the RF channel (as opposed to processing congestion at the node). The last value, P, is a counter provided by the Peg and representing packet losses assessed by the Peg. It is incremented by 1 each time a packet is removed from the Peg's window of outstanding (old) blocks that have not been received.

Item 7 consists of two values shown as n1 @ n2 where n1 is the number of received samples and n2 is the rate. For sampling, n1 is the number of combined samples received so far (by a sample we mean a full set of values for all sensors being sampled), and n2 is the rate in samples per minute. For streaming, n1 refers to triplets of accelerometer values. Recall than one block (packet) of data consists of 12 such triplets.

Command: sample

The command instructs the Tag to start sampling the sensors that are currently on at the specified frequency. The syntax is:

sample –frequency *freq*

The only (optional) parameter specifies the sampling frequency in samples per minute. It defaults to 60 (i.e., one sample per second).

Having received a sample command, the Tag will respond with sample reports taken at the prescribed intervals. A sample arrives at the Peg as an RF packet whose size depends on the active sensors and their selected components. The data is presented in a formatted fashion, by sensor, in a way that should be immediately legible. A spontaneous report arriving from the IMU sensor operating in the motion detection mode is shown in the same manner, except that the data refers to a single sensor only. The sampling continues until a stop command (see below) is issued and accepted by the Tag.

Command: stream

The command instructs the Tag to start streaming the IMU. The syntax is:

stream –file *fn* –limit *nb imu configuration parameters*

All arguments are optional. The command may specify configuration parameters for the IMU sensor (as for config imu, see above) which will be assumed before the streaming operation commences. The command, when properly received by the Tag, will reset the IMU sensor turning it off and on, with the current (possibly updated) configuration of parameters, and start a streaming sensor. The streaming rate is determined by the rate parameter of the sensor (see the config command). If no specific configuration parameters are specified, the current (last-defined or default) set of values is assumed. Note that the

set of options for the sensor must include “s” as otherwise the Tag will refuse to start the operation complaining that the sensor is improperly configured.

The –file parameter specifies the file where the sensor data is to be written. If the parameter is not provided, then no sensor data will be stored. The script will still show the received block numbers in the window (which may be useful for testing), but the data itself will be lost.

Data is written to the file in a textual form with each block presented in one line. The line starts with the time stamp in milliseconds (counting from the beginning of the session), followed by the string “B:”, followed by 12 hexadecimal values representing packed value triplets. The file is meant to be interpreted by programs or scripts. End of train packets are represented by lines marked with the string “E:”.

The –limit parameter can be used to specify the limit on the number of blocks to be collected in the session. The script will automatically issue a stop command (see below) when the goal is reached. Note that this is the number of blocks, not samples. The actual collected number can be exceeded as the script will try to account for the possibly pending blocks that may still be retransmitted (out of order) based on the maximum allowable window size.

Command: stop

The command stops sampling or streaming in progress. It takes no arguments, so the syntax is just:

```
stop
```

The buttons

The buttons in the top frame of the OSS window provide for a simple way to control streaming:

WAKE	Wakes up the Tag. Equivalent to the wake command issued from the command line (see above).
START	Starts a streaming session.
SETUP	Configures the parameters for a streaming session. When pressed the button presents a dialog where the parameters of the IMU sensor (only those that are relevant for streaming) can be preset. Also, the name of the file to contain the streamed data can be specified. This should be done before pressing START.
STOP	When pressed the button issues a stop command to the Tag.

The four colored upper buttons make it possible to insert marks into the stream of data (stored in the collection file) to identify specific points/areas of interest.