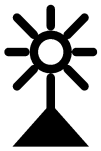
datum



1. a piece of information
2. a fixed starting point of a scale or operation

The goal of this project is to make it easier for students and hobbyists to incorporate sensors into their projects by doing the heavy lifting of translating the language the sensors speak into a format that's easy for humans to read and for machines to parse and process.

Towards that end each datum board emulates a serial port over a USB connection, presents the information and data stored on them in a JSON encapsulated packet, and processes URI style commands to change and retrieve their settings. The datum boards fill the gap between a LEGO Mindstorms sensor and a breakout board.

Hardware Structure

The datum boards are based on the Arduino Zero. They use the same Atmel SAMD21G18 MCU used on the Zero. Each datum board uses a standard micro B USB connector to connect to the host system. The boards identify themselves as USB CDC devices and the host operating system creates a serial port for that board. The serial port settings are 3000000, 8, N, 1. Many terminal programs don't actually care about the settings when a USB CDC device is being addressed but some terminal programs like PuTTY do care.

This means that each datum board can be used on nearly any platform or with any language that can read data from a serial port. There are no libraries that need to be installed. This and human readable data are two of the key features of the datum boards.

Parameter Hierarchy

Information about a particular board is stored in various parameters organized into several higher level categories. The high level categories include device, config, options, and sensors. An example of the structure is shown below.

```
{
  "device": {},
  "config": {},
  "options": {},
  "sensors": {}
}
```

The top level device category stores immutable information about a sensor. Typical parameters stored here are manufacturer, product ID, version information, etc. One of the parameters that will be included is a 64 bit unique user ID (UUID). This acts as both the serial number for the board and as a means to identify this board at a system, household, or even global level. The UUID is the same UUID used for generating MAC addresses in network systems.

The top level config category stores parameters that the user can change. Typical values stored here are the report rate, automatic reporting, compact reports, etc. Another parameter stored here is called 'friendlyName'. This particular parameter allows the user to pick their own

name instead of using the UUID. While it won't necessarily be globally unique values like 'left arm' and 'right arm' are a lot more readable than a big string of numbers.

The options category contains information about the valid values for each of the config parameters. If a range of values can be set the minimum and maximum value will be listed. If there is a pick list of valid values they will be listed under the relevant parameter. Text based parameters like 'friendlyName' will list the minimum and maximum length of the allowable string.

Having the information stored and categorized in this manner makes each datum board essentially self documenting. While the information will always be available on other media and in different formats it is still stored right at hand, where it's needed most.

The sensors category contains more detailed information about individual sensors on a board. The information for each sensor is called out following the same device, config, and option format used in the top level categorization. Items in the sensor category also include a 'data' field where the measurements taken from a particular sensor are stored.

Command Structure

Commands are sent using a URI style syntax similar to that used for REST requests. The response to every command is prefaced by an http response code indicating if a valid command was sent. Commands must also be terminated with a carriage return and an optional line feed.

Requests for information start with a 'get' command followed by the path to the information requested. A command of 'get /' will retrieve all the information in the hierarchy. The command 'get /device' will retrieve the parameters underneath the top level device category. The command 'get /device/manufacturer' will retrieve that specific parameter.

Information from sensors can be collected individually from each sensor but it can also be collected from all the sensors at the same time. A command like 'get /sensors/sensor1/device' would retrieve the device parameters from sensor1. A command like get /sensors/device' would retrieve the device parameters from all the sensors supported by that particular board.

Similarly config settings can be changed using either 'set', 'put', or 'patch'. Patch is the most technically correct but any of the terms work equally well. A typical command to change the report rate would be 'set /config?reportRate=10'. This follows the URI style syntax of using a '?' to separate the resource category from the parameters being changed. An equal signs is used to indicate the new value for a particular parameter.

Multiple commands may be sent with one set command as well. Multiple commands are separated with the '&' symbol. All the new settings must be in the same high level category. The following command would set the report rate to 10 and enable automatic reporting:

```
set /config?reportRate=10&automaticReporting=true.
```

Sensor config values may be set individually or en mass similar to the operation of the get command. Changing the sample rate and data rate of all sensors on a board can be accomplished with the following command:

```
set /sensors/config?sampleRate=10&dataRate=5
```

Commands can also be abbreviated. This is a departure from the URI standard syntax but it greatly eases testing and debugging. This is particularly true when commands are being entered manually. Commands can be shortened to as few as one character. Care must be taken with parameters that start with the same letters. Enough characters must be provided to clearly distinguish which parameter is being changed or unexpected results could occur.

Example of how commands could be shortened are shown in the table below.

Full Command	Abbreviated Command
get /config	get /c
get /sensors/config/dataRate	get /s/c/d
set /config?automaticReporting=false	set /c?a=f
set /sensors/config?filterType=RMS&dataRate=5	set /s/c?f=RMS&d=5

Data Format

Taking measurements with one of the boards is a three step process. The first step is collecting a sample from the sensor itself. Samples are taken at the rate specified in the 'sampleRate' parameter. This information is stored in a 16 element ring buffer.

The second step is transferring the measurement from the sample ring buffer to the data ring buffer. The data ring buffer is also 16 elements. The elements in the data ring buffer are populated at the rate specified in the 'dataRate' parameter.

Collecting samples and transferring them to the data buffer in this way allows various filters to be applied to the measurements as they pass through the buffers. Each time an element in the data ring buffer is populated all the available samples in the sample ring buffer are processed through the filter method selected.

The filters currently available include none, mean, min, max, and RMS. For example, if ten samples are available in the sample ring buffer and the filter type is set to mean the average value of those ten samples will be stored in the data ring buffer. The min filter type would store the minimum of the ten values, the max filter type would store the maximum of the ten values, and the RMS filter would compute the root mean square value of the ten values.

The 'none' filter type just moves the most recent value from the sample ring buffer to the data ring buffer. If the sample rate and data rate for a particular sensor were the same this would be the most appropriate filter to use. A median filter is available in the list of filter types but it hasn't been implemented yet. Other filter types such as an IIR or FIR filter with custom coefficients may be implemented in the future.

The third step is reporting the data stored in the data ring buffer to the host system. This happens at the rate specified in the 'reportRate' parameter. All of the data available in the data ring buffer is processed and formatted for transmission to the host system. Data is returned in a JSON packet with a report timestamp in the header and individual timestamps for each point of data collected. An example of this from the datum-IMU board is shown below.


```
{
  "timestamp": 24042.9,
  "accelerometer": {
    "time": [24042.45, 24042.55, 24042.65, 24042.75, 24042.85],
    "x": [0.82843, 0.828125, 0.828125, 0.827454, 0.828247],
    "y": [0.041138, 0.041626, 0.042053, 0.042419, 0.041443],
    "z": [-0.565735, -0.566162, -0.565735, -0.565308, -0.566895]
  },
  "gyro": {
    "time": [24042.45, 24042.55, 24042.65, 24042.75, 24042.85],
    "x": [0.006195, 0.006226, 0.006134, 0.006165, 0.006165],
    "y": [0.003388, 0.003235, 0.003265, 0.003113, 0.003235],
    "z": [0, 0.000153, -0.000031, -0.000031, 0.000031]
  },
  "magnetometer": {
    "time": [24042.46, 24042.56, 24042.66, 24042.76, 24042.86],
    "x": [0.126011, 0.125736, 0.12653, 0.125889, 0.126652],
    "y": [0.041597, 0.040956, 0.041963, 0.040864, 0.042116],
    "z": [0.068026, 0.068056, 0.069063, 0.070071, 0.070254]
  }
}
```


Prototypes

There are four prototype boards currently available, datum-Distance, datum-IMU, datum-Light, and datum-Weather. Each prototype features a different chip using their own unique communication interface and language. They are meant to showcase the capabilities of each chip while highlighting the common command structure and format for reporting the information collected.