R1 Firmware Scoping

The R1`is a smart lacrosse ball that has the PCB from this project embedded within it.

**Project Overview:**

This comprehensive scoping document outlines the scope, objectives, and tasks for developing advanced firmware for the smart lacrosse ball's embedded sensors. In addition to data collection and the wall ball algorithm, the firmware will include innovative features such as shot speed recording, "find my ball" logic using RSSI, and a shake to wake feature.

**Introduction:**

The journey of a lacrosse ball from wind-up to release is a symphony of forces. At every stage, the ball is subjected to a complex interplay of acceleration, rotation, and impact dynamics. Understanding these forces enables us to craft firmware that not only captures data but transforms it into actionable insights. Let's delve into the advanced features that will propel our smart lacrosse ball to the forefront of sports technology.

Please view [this video](https://youtu.be/Z1PEbo-Y8P4?si=LxtDlw8s6XznGbpO) of a lacrosse “shot”

**\*\*2. Objectives:\*\***

- Develop firmware for the R1 Smart Lacrosse Ball that captures sensor data and transforms it into actionable insights.

- Implement shot detection algorithms using accelerometer and IMU data.

- Enhance advertising strategies to ensure efficient data transmission to the dedicated app.

- Optimize power consumption through adaptive advertising intervals via "Shake to Wake" functionality.

- Incorporate sensor fusion techniques for accurate and relevant data representation.

- Enable seamless communication between the smart lacrosse ball and the dedicated app.

- Implement Over-The-Air (OTA) firmware update functionality.

- Enable the detection of whether the Battery is actively being charged (as in it is on the charger and charging). When the board is placed on a charger, the board should wake up and stay awake while charging.This is not outlined in the BLE advertising portion of the document so either create a new string or place it in a string that has room and then update the BLE characteristic map.

- Create a battery management protocol that does not allow the battery to become overly depleted.

**\*\*3. Firmware Features:\*\***

* **\*\*Shot Detection:\*\***
* Utilize accelerometer and IMU data to accurately detect shots and impacts.
* Implement algorithms to determine shot speed, duration, and impact force.
* Update the Last Shot Stats characteristic with shot-related information.
* **\*\*Advertising BLE:\*\***
* Dynamically adjust BLE Advertising intervals based on activity levels.
* Incorporate adaptive intervals to ensure efficient data transmission during active and sleep modes.
* Advertise relevant data, including shot statistics and battery levels, to the dedicated app.
* **\*\*Power Management:\*\***
* Implement power-saving techniques to prolong battery life.
* Transition between "active mode" and "sleep mode" based on accelerometer activity.
* Utilize a "Shake to Wake" feature, with a threshold of 5g's, to activate the ball from sleep mode.
* **\*\*Sensor Fusion:\*\***
* Combine accelerometer and IMU data using sensor fusion techniques.
* Enhance the accuracy of the shot detection and speed calculations. Note that most shots in lacrosse will far exceed the 16g’s on the IMU, but it will still provide information that will allow our app to determine which hand the ball was shot from and the type of shot (overhand, side-arm, etc)
* Please place on a new string for advertising or discuss with me prior
* **\*\*Seamless Communication:\*\***
* Develop a custom service with characteristics for data exchange between the ball and the app.**[SEE BLE CHARACTERISTIC MAP BELOW](#_UUIDs)**
* Ensure compatibility with standard services like Battery Service, Device Information Service, and Nordic DFU Service.
* **\*\*Over-The-Air (OTA) Updates:\*\***
* Enable remote firmware updates through the Nordic DFU Service.
* Implement secure mechanisms for updating firmware over BLE connection.
* Provide instructions for users to perform OTA updates using the dedicated app.

**\*\*4. Specific Implementation Steps:\*\***

* Develop shot detection algorithms that analyze accelerometer and IMU data to identify shots and impacts.
* Configure the Shot Detection Settings characteristic to adjust parameters for shot recognition.
* Implement the Last Shot Stats characteristic to store and update shot-related statistics.
* Design a Sensor Stream characteristic to send raw sensor data to the app.
* Develop Sensor Stream Settings characteristic to control the rate of sensor data transmission.
* Create Sensor Fault characteristic to notify sensor malfunctions and faults.
* Implement Attitude characteristic to provide orientation data of the ball.
* Develop Tare characteristic to reset the orientation to a predefined state.
* Configure advertising intervals based on accelerometer activity levels and power-saving requirements.
* Implement "Shake to Wake" logic with a threshold of 5g's to activate the ball from sleep mode.
* Design Battery Level advertising strategy with adjusted values to prevent inaccurate battery level readings.
* Utilize sensor fusion techniques to combine accelerometer and IMU data for accurate data representation.
* Develop seamless communication between the smart lacrosse ball and the dedicated app using custom and standard BLE services.
* Integrate Nordic DFU Service for Over-The-Air (OTA) firmware updates.
* Implement secure mechanisms for updating firmware remotely.
* Provide clear instructions for users to perform OTA updates using the dedicated app.

**\*\*5. Example Shot Detection Algorithm in C:\*\* (Illustrative Purposes only)**

#include <stdbool.h>

#define GYRO\_THRESHOLD 1200

#define ACCELEROMETER\_PEAK\_FACTOR 0.5

#define MIN\_SHOTS\_DETECTED 10

bool detect\_shot(int\* accelerometer\_data, int\* gyroscope\_data, int data\_length) {

bool shot\_started = false;

bool shot\_ended = false;

int shots\_detected = 0;

int peak\_acceleration = 0;

for (int i = 0; i < data\_length; i++) {

if (abs(gyroscope\_data[i]) > GYRO\_THRESHOLD) {

shots\_detected++;

}

if (shots\_detected >= MIN\_SHOTS\_DETECTED && !shot\_started) {

shot\_started = true;

peak\_acceleration = max(accelerometer\_data[i], accelerometer\_data[i+1]);

}

if (shot\_started && accelerometer\_data[i] < peak\_acceleration \* ACCELEROMETER\_PEAK\_FACTOR) {

shot\_ended = true;

break;

}

}

if (shot\_ended) {

return true;

} else {

return false;

}

}

float calculate\_shot\_speed(int\* accelerometer\_data, int start\_index, int end\_index, float time\_interval) {

float velocity = 0.0;

float acceleration\_sum = 0.0;

// Integrate acceleration data to calculate velocity

for (int i = start\_index; i <= end\_index; i++) {

acceleration\_sum += accelerometer\_data[i];

velocity += acceleration\_sum \* time\_interval;

}

// Convert velocity from m/s^2 to m/s (assuming time\_interval is in seconds)

velocity /= (end\_index - start\_index + 1);

return velocity;

}

**\*\*6. Data Flow to BLE Advertisement:\*\***

When a shot is detected using the example algorithm, the relevant shot data, including shot speed, and duration is calculated. This data is then updated in the Last Shot Stats characteristic. The relevant shot data, battery level, and other information are incorporated into the advertising payload, ensuring efficient transmission to the dedicated app. Sensor fusion techniques are applied to enhance the accuracy and relevance of the advertised shot data.

**\*\*7. Over-The-Air (OTA) Updates:\*\***

The firmware includes support for remote Over-The-Air (OTA) updates using the Nordic DFU Service. This allows users to update the firmware of the smart lacrosse ball wirelessly through the dedicated app. Secure mechanisms are implemented to ensure the integrity and authenticity of the firmware updates. Detailed instructions are provided to guide users through the process of performing OTA updates.

**\*\*8. Visual Aids:\*\***

Include charts and visuals that illustrate the shot detection algorithms, sensor fusion process, advertising intervals, "Shake to Wake" logic, OTA update process, and interaction between the ball and the app during updates.

**\*\*9. Testing and Validation:\*\***

- Conduct thorough testing of shot detection algorithms using different shot scenarios.

- Verify accuracy of shot statistics, speed determination, and sensor fusion.

- Test advertising intervals, "Shake to Wake" functionality, battery level optimization, and OTA update process.

- Validate seamless communication between the smart lacrosse ball and the dedicated app.

- Debug and refine the firmware based on testing results.

**\*\*10. Documentation and Deliverables:\*\***

- Provide detailed documentation for the developed firmware, including explanations of algorithms, characteristics, functionalities, and OTA update process.

- Deliver the finalized firmware code, well-documented and organized for future reference and maintenance.

- Include visual aids, charts, and diagrams used in the project documentation.

- Present a comprehensive guide for integrating the firmware with the smart lacrosse ball hardware.

- Offer recommendations for firmware updates and improvements based on testing outcomes.

R1 v1.0.0 characteristic map

# UUIDs

The custom service and characteristics share the same UUID base:

**1bc5XXXX-0200-b8be-e611-e60c60b7c457**

The two bytes marked as **XXXX** compose the ID of each characteristic and the service.

The service ID is **0x001**, so its **UUID** is:

**1bc50001-0200-b8be-e611-e60c60b7c457**

# Map

| **Name** | **ID** | **Access** |
| --- | --- | --- |
| [Session](#_w0dzyyhco2al) | 0x1100 | Read/Write |
| [Last shot stats](#_xirh2tf3qmc5) | 0x1101 | Read/Notify |
| [Shot detection settings](#_arpw3iqfpvfx) | 0x1102 | Read/Write |
| [Name](#_x60us7uk8adz) | 0x0133 | Read/Write |
| [Sensor stream](#_r0sd6wyev78y) | 0x0011 | Notify |
| [Sensor stream settings](#_j01un21oh44w) | 0x0012 | Read/Write |
| [Sensor fault](#_6yofhicncuh6) | 0x0013 | Read/Notify |
| [Attitude](#_y0zrpb4hrjb3) | 0x0102 | Notify |
| [Tare](#_sqbduxvmpkx) | 0x0129 | Write |

# Characteristics’ description

## Session

The session characteristic can be used to control the training session. It holds 4 bytes of data that can be used to indicate the session ID or start timestamp. The value is not used by the device, but it can be read by the app.

Writing the Session characteristic clears the shot/bounce counter in [Shot stats](#_cipg0ica2pwd)

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Description** | **Type** | **Size [bytes]** |
| IdOrTimestamp | Session ID or start timestamp | uint32 | 4 |
| **Total** | | | **20** |

The characteristic supports reading and writing.

## Last shot stats

The characteristic holds the statistics of the last detected shot. Its value is updated for every shot detected.

The characteristic holds the [Shot stats](#_cipg0ica2pwd) structure.

The characteristic supports reading and emits notifications.

## Shot detection settings

Shot detection settings configure the shot detection and flight length calculator algorithms.

The characteristic holds a structure of data:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Description** | **Default value** | **Type** | **Size [bytes]** |
| AccFilterCoeff | The coefficient for filtering ACC data | 0.05 | float | 4 |
| AccBaselineCoeff | The coefficient for calculating baseline in shot detection algorithm | 0.005 | float | 4 |
| AccSamplesAboveMinBeforeShotEnd | Determines how many ACC samples above (greater than) a local minimum is needed to recognize the local minimum as a shot end | 4 | uint16 | 2 |
| GyroThreshold | The minimum spin required to detect a shot. During flight, the ball must spin as least as fast for the shot to be recognized. | 2500 | uint16 | 2 |
| GyroStableRotationSamplesRequired | Minimum length of the flight after a shot for it to be recognized. In gyro samples.  The value of 80 samples means that the ball must fly for at least 0.1s for the motion to be recognized as a shot. | 80 | uint16 | 2 |
| GyroStableRotationMaxDeviation. | The maximum deviation of the spin during flight.  If the spin rate changes more than this threshold then a flight end is detected. | 15 | uint16 | 2 |
| **Total** | | | | **20** |

The characteristic supports reading and writing. When written, the settings are saved and applied immediately.

## Name

The name characteristic can be used to read or write the name of the device. Writing the characteristic will save the name and apply it immediately. The name is included in advertising.

The characteristic has variable length. It holds the name encoded in ASCII without the trailing null-terminator. The maximum length of the name is 16 bytes.

## Sensor stream

The sensor stream characteristic sends raw sensor data from the device. The rate at which data is sent can be configured via [sensor stream settings characteristic.](#_j01un21oh44w)

The characteristic streams data in notifications. Each notification contains one or more samples. The notifications should be treated as a continuous stream of data. It doesn't matter in which notification a particular sample came.

### Sample structure

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Description** | **Type** | **Unit** | **Size [bytes]** |
| Index | The sample index. Incremented by 1 for each sample.  The most significant bit tells if the sample comes from ACC or GYRO:   * 1: ACC * 0: GYRO | uint16 | 1 | 2 |
| X | Raw sensor measurement in the X axis. | int16 | LSB | 2 |
| Y | Raw sensor measurement in the Y axis. | int16 | LSB | 2 |
| Z | Raw sensor measurement in the Z axis. | int16 | LSB | 2 |
| **Total** | | |  | **8** |

The sensor data is sent in raw form, without any preprocessing. They can be converted to more useful units:

* ACC: Acceleration = LSB \* 400 / 32767 [g]
* GYRO: Rotation rate = LSB \* 4000 / 32767 [dps]

When configured at full speed, the samples are sent at the same rate as they are collected from the sensors:

* ACC: 1000 Hz
* GYRO: 800 Hz

## Sensor stream settings

The sensor stream settings characteristic can be used to limit the rate at which the sensor stream characteristic sends data.

Changing the streaming rate is done by changing the prescalers for each of the two sensors. The prescaler should be interpreted as the count of samples that will be skipped for any sample that’s sent. E.g

|  |  |  |  |
| --- | --- | --- | --- |
| Prescaler value | Comment | ACC rate [Hz] | GYRO rate [Hz] |
| 0 (default) | All samples sent  (0 samples skipped for 1 sent) | 1000 | 800 |
| 1 | Every second sample sent. (1 sample skipped for 1 sent) | 500 | 400 |
| 2 | 2 samples skipped for 1 sent | 333 | 266 |
| 99 | 99 samples skipped for 1 sent | 100 | 80 |

### 

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Description** | **Type** | **Size [bytes]** |
| AccPrescaler | Prescaler value for the ACC | uint8 | 1 |
| GyroPrescaler | Prescaler value for the GYRO | uint8 | 1 |
| **Total** | | | **2** |

## Sensor fault

The sensor fault characteristic is used to signal if the sensors (ACC and GYRO) work correctly

The characteristic holds a single byte, where the two least significant bits indicate the faults of sensors, ACC and GYRO for the first and second bit respectively.

* 0x00 - no fault, the sensors work correctly
* 0x01 - ACC malfunction
* 0x02 - GYRO malfunction
* 0x03 - Both sensors malfunction

If any malfunction is signaled, then the device won’t operate properly.

The characteristic supports reading and emits notifications.

## Attitude

The attitude characteristic sends data describing the current device attitude. It can be used e.g. to draw a 3D model representing the devices orientation in space.

The orientation reported in the Attitude characteristic is relative, not absolute. A device lying flat on a table won’t necessarily report 0 pitch and 0 roll, it will rather report a steady but arbitrary orientation. The [tare characteristic](#_sqbduxvmpkx) can be used to reset the orientation.

The characteristic streams data in notifications. Each notification contains one or more samples. The notifications should be treated as a continuous stream of data. It doesn't matter in which notification a particular sample came.

### Sample structure

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Description** | **Type** | **Size [bytes]** |
| Timestamp | The index of the last collected ACC sample. Can safely be ignored. | uint32 | 4 |
| W | The components of the orientation quaternion. | float32 | 4 |
| X | float32 | 4 |
| Y | float32 | 4 |
| Z | float32 | 4 |
| **Total** | | | **20** |

## Tare

Tare characteristic can be used to tare the [attitude](#_y0zrpb4hrjb3). After taring, the attitude in current orientation will read 0, the quaternion will be set to identity (w=1, x=0, y=0, z=0).

The characteristic is write-only. Writing a single byte set to 1 to the characteristic will tare the attitude reading.

# Advertising data

The SFM R1 broadcasts [shot stats](#_cipg0ica2pwd) data in the scan response data. The data is encoded in manufacturer specific data with manufacturer ID of 0xfffe.

# Shot stats

The shot stats structure is used in both the [Last shot stats](#_xirh2tf3qmc5) characteristic and in [Advertising data](#_hlhjwpnbxwjl).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Description** | **Unit** | **Type** | **Size [bytes]** |
| Shots (wall-bounce) count | The count of shots detected. Same as the count of wall bounces. | 1 | uint16 | 2 |
| Speed | The maximum speed | 0.01 m/s | uint16 | 2 |
| MaxAcc | The maximum acceleration registered during the shot. | 0.0025 g | uint16 | 2 |
| Duration | The duration of the shot (throw). I.e. how long it took from the start of a throw until the ball left the stick. | 0.001 s | uint16 | 2 |
| TimeOfFlight | The time of flight, i.e. how long it took between the shot and impact on wall/target. | 0.00125 s | uint16 | 2 |
| AccSamples | Samples of acc (see below) | 0 to MaxAcc | uint4 (two samples per uint8) | 10 |
| **Total** | | | | **20** |

## ACC samples

The last 10 bytes of the Shot stats structure holds a rough preview of the ACC samples collected during a shot. No matter how long the shot took, the data is always resampled to 20 4-bit samples.

Each byte of the preview encodes two 4-bit samples. The first sample is encoded on the less significant 4-bits and the second is encoded on the most significant 4 bits of the byte. E.g. a byte value of 0x21 (33 dec) encodes samples: 1 & 2.

# Standard services

Besides the custom service, the SFM R1 supports the following standard services:

* **Battery service** - used to read the battery state
* **Device information service** - used to read information about the device: name, firmware version, manufacturer
* **Nordic DFU service** - Used for OTA firmware update