Real-Time Analytics API Details

Components & Integration Method

- AnalyticsLib.h: ANSI-C Header File
- libStrideAnalytics.a: fat library containing 32-bit, 64-bit and watchOS build for library

Integration into Objective-C is simply through including the header file and adding the library to the linker list.

Integration into Swift:

Follow steps listed as in this web tutorial: https://medium.com/swift-and-ios-writing/using-a-c-library-inside-a-swift-framework-d041d7b701d9

Usage Process

1. When starting analytics, call the **InitUser** function with the user's details. This needs to be done **once**, at the beginning of the application. This initializes all appropriate data objects and sets the Analytics usage context. If no specific user info is collected by the application, simply pass 0 to that argument. Analytics will use appropriate defaults. While other arguments are self-explanatory and can be understood easily through header file, these two need some clarification:
    a. no_of_devices: 2 if both insoles are to be used.
    b. isShod: 1 if the person is using insoles inside shoes; 0 if using outside (as a foot pressure plate)
    c. reportPressure: Set this to 1 if you need pressure, and not load (force). Pressure is reported in KG/cm2
    d. footsize: Provide foot length (tip of toe to end of heel) in cm (again, foot size. Not shoe size). This is used for accurate calculation of weight in different foot regions.

2. After connecting to the insoles in the application, query the "Hardware Revision String" parameter for the device. Before enabling notification, ensure appropriate sensing spec is set up. Since the Analytics library is built to handle multiple versions & SKUs of hardware, we need to communicate the spec for this specific sensor hardware. This needs to be done once, before BLE notification is enabled:

   For 8-position (10-sensor) insoles:

   a. If Hardware Version String is `STRD_INSAE_11`: call :

```
SetSensorSpecNew (20,18,19, 23, 17, 21, 24, 22, 2, 6, 1);
SetSensorSpecNew (20,18,19, 23, 17, 21, 24, 22, 2, 6, 1);
```

   b. If Hardware Version String is `STRD_INSAE_10`: call :

```
SetSensorSpecNew (23,21,19, 20, 17, 18, 24, 22, 2, 6, 1);
SetSensorSpecNew (23,21,19, 20, 17, 18, 24, 22, 2, 6, 0);
```

   c. If Hardware Version String is `STRD_INSAE_05`: call :

```
SetSensorSpecNew (10, 8, 6, 7, 4, 5, 11, 9, 2, 5, 1);
SetSensorSpecNew (10, 8, 6, 7, 4, 5, 11, 9, 2, 5, 0);
```

3. Call **ProcessStride_FSR** for each frame of BLE data that needs to be processed. This function will process the BLE raw data, and populate the respective data structures that will contain the real-time processed values.
   a. BLE raw data needs to be passed as an array of **unsigned char**.
   b. iOS generally stores the bytes from BLE devices as NSData, which can be passed to this function through [NSData bytes] or [NSData getBytes], and the array length will be 20 chars. Android stores BLE raw data as ASCII string of the BLE raw data, and the array length will be 40. The "length" argument communicates to the Analytics whether this is from iOS or Android, and it will process the data appropriately (including converting ASCII to relevant data in case of Android).
   c. **speed_m_s** is speed in meters per sec**, d_m** is distance in meters, and **alt_m** is altitude from sea level in meters. In applications where it is not relevant, 0 can be passed for these arguments.
   d. **isLeft** is to denote whether we are currently processing left foot data or right foot. Pass 1 if this is left foot, 0 if right foot.
   e. **mode** denotes the type of activity we are analyzing. For any non-standard application, this should be set to 0. For run/walk, this should be set to 1.
   f. **isReset** is provided by the application to demarcate an end of single "micro-analysis period", and collect data for that micro-analysis period for eventual averaging or totals. For example, in walking, this would be when a step has ended, or in case of cycling when a full rotation is complete. If not intended by application, this can always remain as 0.
   g. **isInsight** is to tell analytics we are using INSIGHT insoles. Set to 1 if using INSIGHT insoles. Set to 0 otherwise.
   h. **noConstrain** is to turn off automated pattern-intelligence processing by analytics. Simply speaking, by default, if analytics notices unusual or

unreasonable values all of a sudden (could happen spuriously for various reasons), it decides to throw that value and use established pattern to make that decision. Setting this to 1 will turn this behavior off, and will allow spurious high-variation values to be accepted.

   i. In case mode is set to 1 (run/walk), non-zero return value is obtained when a step completes, and 0 is obtained intra-step. In case mode is set to anything other than 1, return value of this function is 0 in case of bad inputs or any other bad calculations. It is recommended that application only use calculated results when the return value is 1.

4. Once analysis has been completed, and it needs to be restarted by clearing off all internal states, call **ResetStrideInfo**.
   a. **ResetStrideInfo** does NOT clear up user info, so UserInfo does not need to be initialized again.
   b. **ResetStrideInfo** takes argument isShod, which can set/reset the mode of usage (whether with shoes or without), in case user changed the mode.

Accessors for processed values

In iOS Objective-C, all accessors return the pointer to the **struct** object specific to that accessor. Analytics does not prevent mishandling of the object by the application, and in that sense, care must be taken to ensure application does not corrupt the **struct** objects.

- **GetUserInfo** returns the **struct UserInfo** pointer, where user data is stored.
- **GetRunInfo** returns the **struct RunInfo** pointer, where run info is stored (not relevant for applications other than walking/running).
- **GetNewStrideInfo** returns the **struct NewStrideInfo** pointer, which always stores the CURRENT stride information. On next call to **ProcessStride_FSR**, all the fields in this struct will get updated. Any persistent storage of the data if needed, needs to be handled by the application by copying the values into it's own data objects.
- **GetStressInfo** returns the **struct StressInfo** pointer, which always stores the CURRENT stress information. On next call to **ProcessStride_FSR**, all the fields in this struct will get updated. Any persistent storage of the data if needed, needs to be handled by the application by copying the values into it's own data objects. The weight/force measured shall reflect in this data object. For INSIGHT insoles, "stress" simply refers to Load (measured in KGs).
  - **hallux** will point to Big Toe
  - **front** will point to metatarsal 1,2
  - **front_2** points to metatarsal 3,4,5
  - **mid** points to outside of the midfoot
  - **arch** points to the inside arch of the foot
  - **heel** points to the heel area of the foot. In 8-position insoles, this will be lateral (outer) heel. In 6-position, this is at the center of the heel.

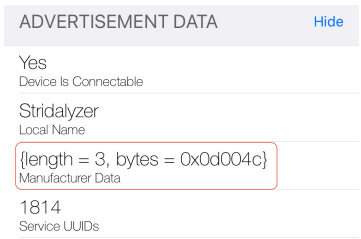
- **heel2** points to the medial (inner) heel in 8-position insoles. In 6-position insoles, this will simply be equal to "heel".
  - To get overall heel loading, heel and heel2 values must be added.
  - To get pressure on heel in 6-position insoles, heel2 can be ignored.
- **toes** points to the smaller toes; more specifically the base of the 4th toe.
- **lr_*** refers to Loading Rate (KG/s).
- **peak_*** refers to peak values obtained during the analysis.

- **GetRevNumber** returns the revision number of the analytics library
- **SetDebugMode** sets printing of certain debug message in the library, if an argument of 1 is provided.

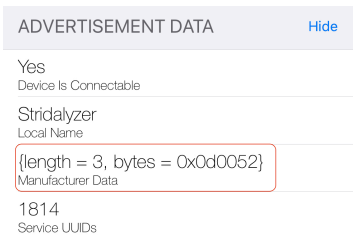
Setting up the Insoles

Stridalyzer Insoles connect over BLE 4.0 protocol, and send data to the application. Here's the process to connect to the insoles and read data:

1. Set up BlueTooth LE Protocol and connection manager.
2. Scan for device with substring "Stridalyzer" in its name, and supporting service UUID 1814.
3. Connect, and discover services and characteristics.
4. The insoles send out advertising data which mark the LEFT and RIGHT specifically. Your app can read these advertising packets BEFORE connecting, so you can detect which insole this is, even before connecting. To ensure LEFT and RIGHT devices are recognized appropriately by the app, use the following logic:
   a. Read the advertising data.
   b. Check for the "Manufacturer data" field. It should be a 2 or 3 character field depending on the OS.
   c. The last character will be 'L' (ASCII value 4c) for the LEFT insole, and 'R' (ASCII value 52) for RIGHT insole.



LEFT



RIGHT

5. Battery data notification can be enabled at this point if application wants to show battery level.

6. When application is ready to receive and process relevant biomechanics data, set up notification for characteristic 2A53 on all connected devices.
7. Another characteristic, **FF03** is provided that sends step-specific data – at every step the data changes, and will update step count, and the **peak forces** during that step. This is useful in case the application does not want to or need to process instantaneous streaming data. To use data from this characteristic, call **ProcessStride_FSR_step** instead of **ProcessStride_FSR** as mentioned above.
8. For Stridalyzer INSIGHT default data rate is 5Hz, and Performance devices, default data rate is 2Hz. Device data rate is adjustable through software, through the following steps:

   The characteristic **FF02** in Service UUID **1814** can be written to, to program the sampling rate.

   The command for changing sampling rate is **0x000B**, and needs to be programmed to bytes 0 & 1
   The opcode is data sampling period in milliseconds (represented in HEX), and should be programmed in byte 5

   An example of sending this command is below:

```
int period = 1000/dataRateInHz;
unsigned char data[6] = {[0 ... 5] = 0x00};
data[1] = 0x0B;
data[5] = (unsigned char)(period&0xFF);
```

   Then you can send the char array **data** to the characteristic FF02.

9. If device disconnects for some reason, and reconnects, make sure to re-enable notification.

NOTE: Stridalyzer devices do NOT implement AES or pairing-based security. Any client can connect to the devices.

LightBlue app on iOS and BLEMonitor App on Android are good debugging apps to be able to connect to the devices and understand various supported characteristics and parameters.

Support & Help

For reporting bugs and errors, or for seeking assistance for APIs, please send an email to support@retisense.com. ReTiSense Team commits to acknowledge and respond within 24 hours, and provide an assessment of severity and potential ETA.

Terms and Conditions

This document, the API header source file and the compiled libraries are provided pursuant to the Non-Disclosure agreement. Any efforts to reverse engineer or decompile shall be considered a violation of that agreement.

Document History

Apr 10, 2019:
- Added support for New version of devices.

Dec 10, 2018:
- Added support for New version of devices.

Aug 20, 2018:
- Added support for 8-position insoles and associated data structures.
- Added documentation for FF03 characteristic

Dec 29, 2017:
- Added sensor mapping for latest devices.
-

Nov 5, 2017:
- Bug fixes for pressure value scaling at higher force ranges.
- Increased LUT range for higher force ranges.

Oct 8, 2017: MAJOR UPDATES:
- Added data object NewStrideInfo – this data object is compatible with new analytics of intra-stride and gait phase analysis. More to come in future versions.
- Added better pressure <-> weight transformation. For that purpose, need foot size, which was added as argument in InitUser.
- Added argument “reportPressure” to InitUser to expose pressure values if a developer wants.

Sep 17, 2017: Added documentation for option “isShod”

Aug 4,2017: Added “SetSensorSpec” function call.

Jul 19, 2017: Added “Setting up Insoles” section, and removed dev_type parameter from InitUser.

Jul 10, 2017: Updated to latest analytics rev. Added Swift flow weblink.

Mar 12, 2017: Initial Version detailing main flow for upcoming INSIGHT insoles.