UNCAP Driver description and example

UNCAP drivers are standalone Android applications. Each application has the following metadata:

- package id (string: com.uncap.driver.glucose.vpd2in1)
- service id (string: com.uncap.driver.glucose.vpd2in1.GlucometerService)
- driverType: “foreground” or “background”
- description: “Human readable description for catalog, etc.”
- masterAppButtonLabel: “Blood glucose meter”
- version: (int: 1)
- lastUpdateDatetime: “2015-01-01 11:11:11”
- apkUrl: “https://uncap.eu/drivers/driverGlucose.apk”
- barCodeId: “00000001”
- hasSettings: true
- hasMeasurement: true
- [more to be added if necessary]

Driver catalog

All of this is packed into a JSON array, and stored on the server. This serves as source data for the UNCAP driver catalog, and for the Android application to learn about new drivers and their capabilities.

```
{
  "drivers": [
    {
      "packageId": "com.uncap.driver.mydriver.mycompany",
      "serviceId": "com.uncap.driver.mydriver.mycompany.HeartrateService",
      "driverType": "background",
      "description": "This is the driver for heartrate",
      "masterAppButtonLabel": "HeartRate",
      "version": 1,
      "lastUpdateDatetime": "2015-25-11",
      "apkUrl": "https://dl.dropboxusercontent.com/u/79979620/hr.apk",
      "barCodeId": "00000003",
      "hasSettings": true,
      "hasMeasurement": true
    },
    {
      ...
    },
    {
      ...
    }
  ]
}
```

Current version of the catalog hosted by TRI: http://www.uncap.eu/driver-apk/driver-list.json

The idea is that if this file is updated regularly, it can power both the UNCAP catalog online, as well as all of the UNCAP Android apps.

Master Android app driver installation

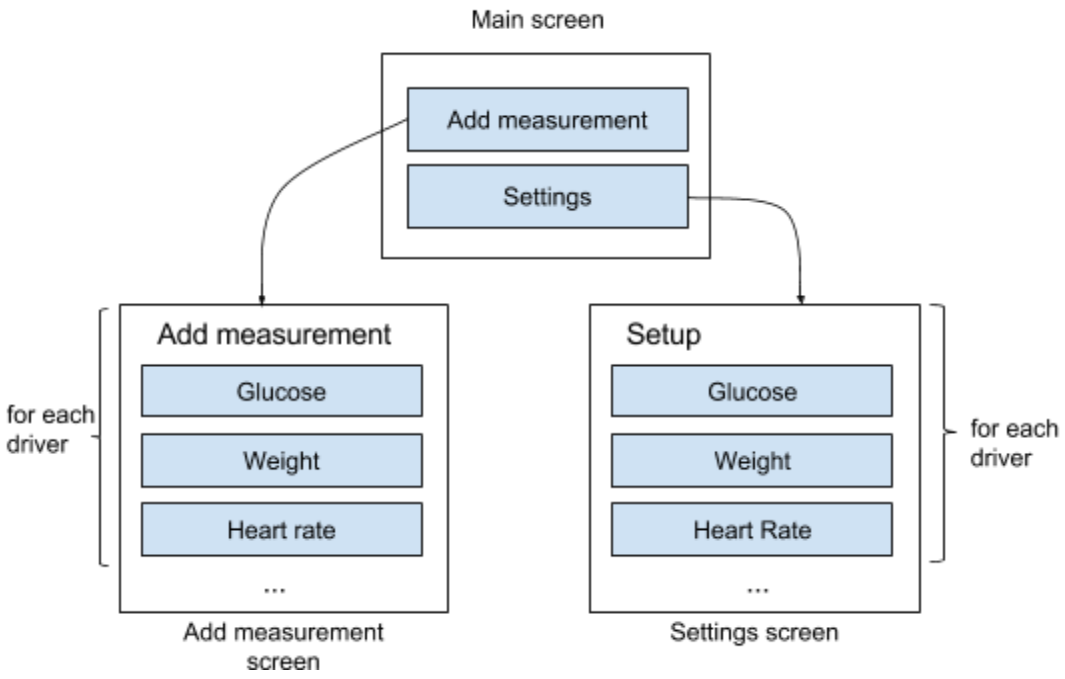
Users can add (install) driver apps in three ways:

- **[The preferred way]** By scanning the barcode / QR code (or manually entering the ID) through the UNCAP master app, which automatically downloads the driver app and prompts user to install it.
- By finding the link in the online UNCAP catalog, which takes the user to the APK
- By manually finding the app in the Google Play store and installing it

Master Android app driver detection

The master app:

- Regularly fetches the driver catalog (JSON) and stores the last version locally
- Based on the packageId, the master app checks if the packageId is installed
    - This happens when user presses [Settings] or [Add measurement] buttons in the master app
- When user presses the [Settings] button:
    - The master app loops through JSON, checking which driver apps are installed
        - For each installed app, it shows the settings button
        - This only happens, if the driver has “hasSettings” flag set to true
        - The button that appears, has the text specified in “masterAppButtonLabel”
- When user presses the [Add measurement] button:
    - The master app loops through JSON, checking which driver apps are installed
        - For each installed app it shows the “Add measurement” button
        - This only happens, if the driver has “hasMeasurement” flag set to true
        - The button that appears, has the text specified in “masterAppButtonLabel”



Communication with the driver

Communication with the driver happens in three ways

1. by opening a HTML-based interface in the webview of the master app
    - actions in this interface are triggered using JS, and relayed back to the driver using intents
2. by opening the driver application directly from the master app
    - this is meant primarily for media-rich and interactive applications (featuring video, animations, or apps that cannot be easily ported to HTML GUI)
3. and with the driver sending the data to master app in the background using intents

When each option is used:

- For settings, option 1 is always used.
- For measurement, either option 1 or option 2 is used
    - this is based on the "driverType" field in the JSON metatada
    - "driverType": “background” means option 1 will be used
    - "driverType": “foreground” means option 2 will be used
- For communicating data from driver app back to master app, android intents are always used.

Each option is described in more detail below.

Option 1: HTML GUIs (for communication master → driver)

Each driver comes with a handful of static HTML (CSS, JS) files in the `assets` directory. On first launch, it copies the files to a folder `(sdcard)/UncapFiles/`.

- For this each driver creates a subdirectory with the name of its package id
    - example:
      `(sdcard)/UncapFiles/com.uncap.driver.glucose.vpd2in1`

Each driver comes with two main HTML files (both residing in that directory); both files can include additional CSS or JS files, or have the CSS/JS code inline.

- settings.html → here a settings GUI is defined
- measurement.html → a measurement GUI

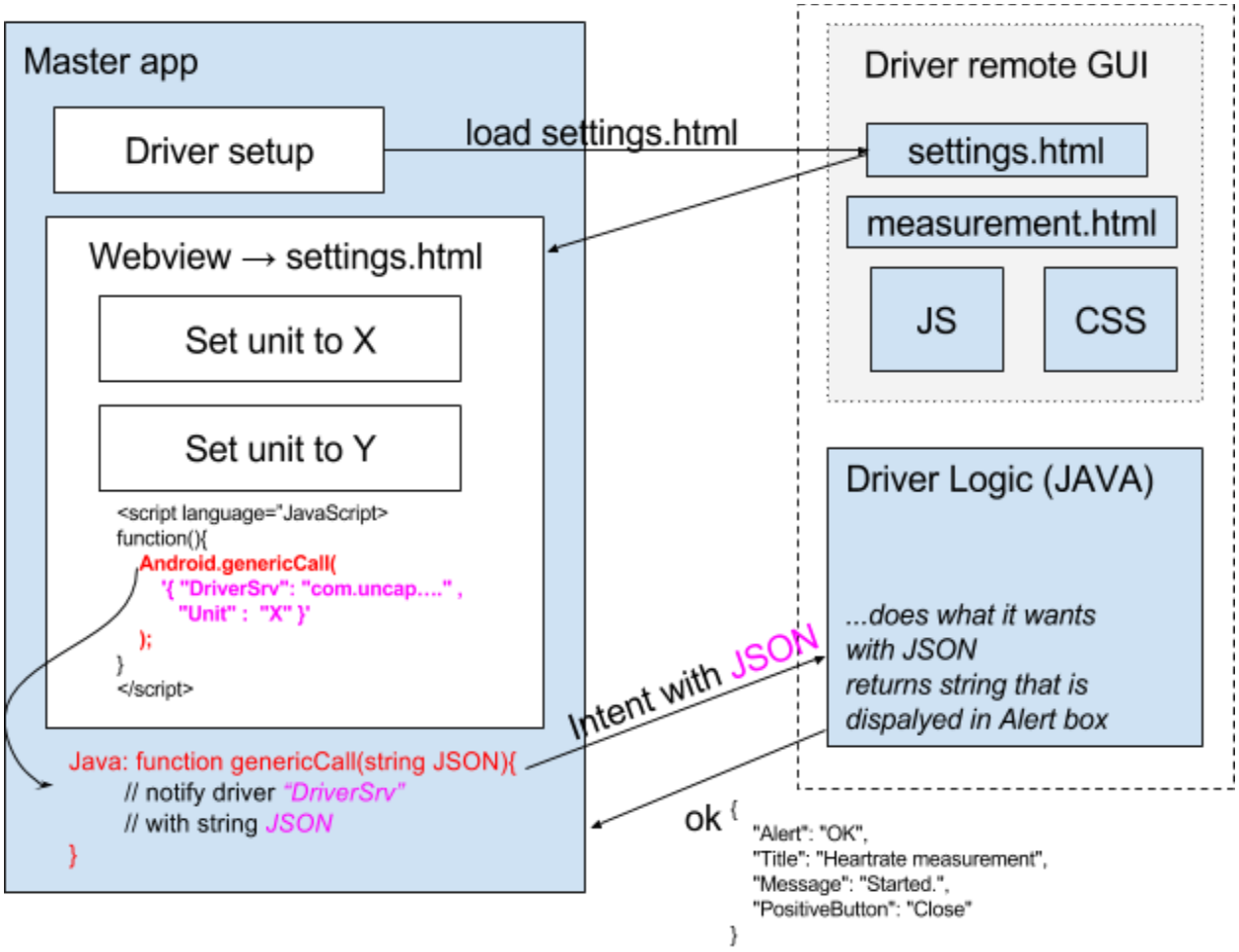
HTML interface guidelines

Each of both HTML files represents a GUI; two main GUIs are foreseen:

- Configuration GUI (settings.html)
- Measurement GUI (measurement.html)

Each interface packs all the data it wishes to communicate back to the driver (= back to itself) in JSON format. Then, it sends the payload to the method *genericCall()* defined in the master app. The payload is then relayed by the master app back to the driver, which unpacks it and parses it.

For an example, see figure below.



Option 2: Opening the driver app (master → driver)

If app has the "driverType" == “foreground”, the actual driver app is opened in the foreground. When measurements are done, the user is returned back to the master app.

Option 3: Communicating back the results (driver → master)

When the driver app obtains the results from the device, it sends the data in JSON form by using intents.

To do this, driver binds to **com.uncap.box.main.COM**

Four methods are exposed for sending data to the master application

- pushMeasure is used for sending measurement data.

- The driver is expected to pack the measurement data as an openmhealth payload, with added measurement type (openmhealth schema id, e.g. `omh:heart-rate:1.0`), like this:

```
{
      "type": "omh:heart-rate:1.0",
      "payload": {
            "effective_time_frame": {
                  "date_time": "2015-12-08T16:37:25Z"
            },
            "heart_rate": {
                  "value": 104,
                  "unit": "beats\/min"
            }
      }
}
```

  - the received JSON string is equipped with user_id (collected from login details), raptor stream name, raptor device id, and is pushed to the message bus
  - all data that appears on message bus is picked by the sync module of the master app
  - Sync module forwards the data to the server; if there is no connectivity, data is stored and sent later, when connectivity is back
  - Raptor receives this and relays it to chino, CEP, etc.

- pushAlarm is used for raising an alarm
- pushPosition
- pushPhr