Best Practices

Version 1.1

So, this document is to give an idea of the kind of thing that we try to adhere to in all of our software. A lot of this is just about being clean when you write code, and a lot of this can be cured by a mind set that realizes that **many more people than yourself** **will have to read your code** (this statement should both frighten and motivate you). If you can keep that in mind, then most of this will already be achieved.

# Use SysMan

Please add the SysMan library. Just do it, don’t think about, do it. If you have constants you use in your app then consider adding a class there with your constants. Try **NOT** to create constants that are used in your application and in DiAs Service or other places and just add them as globals. If you have these cases then add them to SysMan, that way if we change things you know that all the applications will have the modification and you won’t have to change it in seventeen different places. This is basically the point of the SysMan library.

# Don’t just Log, Debug instead!

Debug lines should be used, and used a lot (other than putting statements in high frequency thread loops we pretty much never frown on this). However, you need to adhere to the system of Debug statements that we use. We have overridden the built in “Log” function and use the “Debug” function from SysMan. It works basically the same way, except it takes three arguments not two. The format is as follows:

Debug.[*i,d,w,e*](*ClassTag*, *FunctionTag*, *Message*);

**Every** class must have a TAG string and **every** function, where you debug, must have a FUNC\_TAG string. The debug statements will automatically format the spacing between the function tag and message so there is no need to do that.

Example: Debug.w(TAG, FUNC\_TAG, “An error has occurred!”);

# Naming stuff

Use naming conventions. One does not simply choose a naming convention at random (see Boromir, reference vacation to Mordor…). There are many ways to name your variables: caps, lower case, camel case, underscores, etc. Ideally try to group like variable types with like conventions. For instance, maybe all your globals are all caps, that would make it easier to pick them out in a long set of code, or all your local variables are lower case, or parameters for a certain device all have that device name with an underscore and the parameter name. Anyways…you get the idea…

# Format, format, format

If it looks cleaner, it is easier to follow and therefore easier to debug when it inevitably breaks down. We understand that people write code quickly and when you’re on a roll, there’s no time to stop and check your indentation, but when you are going to commit to the server at least make your last commit look nice. Remove whitespace, line up your brackets, erase things that you aren’t going to use that are commented, etcetera, etcetera.

# Do. Or do not. There is no ‘try’.

The try/catch…yes, we’ve all heard of it, and in many cases it is required to catch illegal exceptions. It is **NOT** meant to be used as a safety net that covers bugs or errors in code. If you need try/catches to keep code from crashing, then that code needs to be written so it doesn’t rely on them to function.

# Tell us about what you do

Commit notes. When you push code to the repo, the commit notes are used to describe the changes you made to other users. If you are to commit changes to the good ol’ SVN repository, here are a few rules to follow when it comes to write those notes.

* 1. Any line starts with either a ‘+’ or a ‘-‘ symbol. ‘+’ is used to describe a **major** functionality change in a module, either because it changes the way the module works or because the modifications will have an impact on other applications in the system. ‘-‘ is used for **minor** changes, such as bug fixes, cleaning of the code or any modification that has no or few impact on the rest of the system.
  2. Precise **on each line the name of the application** in which the changes were made. If you were working on a script or file that is not part of any application or module, simply write its name. If you want to make a general comment, use the keyword ‘GENERAL’.
  3. Write your comment in a **clear and concise** way. Notes shall summarize what you actually modified/added so that any other user can understand globally what you did without having to dig into the code.

Here is an example of commit notes correctly formatted:

Keep in mind that you are identified by your username when you commit any change to the repo, so if you do it wrong, we can track you down, find you and terminate you. No pressure.

*+ BiometricsContentProvider: Removed all data fields and now using json\_data for storing information from exercise sensors*

*+ ExerciseService: Data is now parsed from JSON Objects not specific columns in the table*

*- ExerciseService: Added the notion of device types to parse the correct data*

*+ HR\_Driver: Converted so that data is stored directly to the DB from the drivers in JSON string form*

*- HR\_Driver: Cleaned up UI, formatting date, removing unused buttons, unused options, etc.*

*- installall.py: Added the new PumpService to the install apps in the script*

# Take what you need, leave the rest

Optimize queries to DB Tables. When you have to query data from the database’s table, try to be as efficient as possible. Android Cursors have a lot of options to help you perform queries. An optimized query means less processing time, less use of memory, and the guaranty to get the result you expect.

Here is an example of a BAD way to query data. The objective is to get the latest “hypolight” value from the STATE\_ESTIMATE table.

Cursor c=getContentResolver().query(Biometrics.*STATE\_ESTIMATE\_URI*, **null**, **null**, **null**, **null**);

**if** (c.moveToLast()) {

**int** hypolight = c.getInt(c.getColumnIndex("hypolight"));

}

You might think **“What’s wrong, this query is totally working, why should I care?”**

Not only this query is not effective but there are also several elements that also make it at least dangerous, if not wrong.

* We query the whole table in order to get a single row… Not really great.
* We go to the last of the queried rows assuming it will be the more recent. It might happen to be the case, but how sure are we about that?
* We query all the fields of the table when we only need one. If you are reading this file, you probably know how big the State Estimate table is: that’s not really good either.
* What if the row we get does not have the “hypolight” data? Don’t look at me, I don’t know!

**“Okay, so … what should it be then?”**

Keeping those elements in mind, let’s have a look at how to improve the same query:

Cursor c=getContentResolver().query(Biometrics.*STATE\_ESTIMATE\_URI*, **new** String[] {"hypolight"}, "hypolight IS NOT NULL", **null**, "time DESC"+" LIMIT 1");

**if** (c.moveToLast()) {

**int** hypolight = c.getInt(c.getColumnIndex("hypolight"));

}

How is that better?

* We sort the result of the query by descending value of “time”, so we know that the first row will actually be the latest one. This happens in the last argument of the function: *order by*.
* Better: We limit the result to one row, so we end up with the only row that we are interested in! There is no ‘limit’ argument in the function, so it goes at the end of the *order by* value, the last to be read to build the query.
* We know we only need the “hypolight” field? Let’s use the *projection* argument to only grab this field from the database.
* Oh, and since we’re at it, let’s just make sure we will get a value, so we are going to only search for rows where the field is not null. That’s the *selection* argument.

Here is what those two queries look like in SQL.

The bad one: *“SELECT \* FROM <State Estimate Table>;”*

The good one: *“SELECT ‘hypolight’ FROM <State Estimate Table> WHERE ‘hypolight’ IS NOT NULL ORDER BY ‘time’ DESC LIMIT 1;”*

That’s cleaner, isn’t it? Of course, it’s a bit longer, but in the end, it comes to the very deep meaning of programming and engineering: *what can we do now to have less to do later*?

## Thanks for your time :)