1 Introduction

This document outlines the methodology, functions and possible improvements in the device management. It is divided into sections, each one defining a particular app, and each app divided into subsections, which define any methods or modules that are used.

2 Applications

2.1 file-upload

This is the "boss" application that does nothing but hold settings.py and urls.py, which contain settings and urls. Any new application will have to be included in these files in order to work properly. Can be left untouched.

2.2 fileupload

This is the main "fileupload" application that loads the `ekfile_form` for file upload. The webpage sends a JQuery `getJSON` request that returns a JSON file which is parsed by the webpage to enable/disable USB upload/download buttons.

2.2.1 views.py

`verify_USB`

This method tries to get a value from the `attemptMount` method in `USBFinder.py`. Any value that is not `None` implies an existence of a connected USB device, and this returns a JSON Response that lets `ekfile_form` keep the buttons as is. If there is no USB device attached, the JSON Response tells it to disable the USB buttons.

`download_to_USB`

This method downloads files to USB from a predefined folder that contains the telemetry data, which is defined in `/support_files/res.json`. The method loads files that are present in the directory and satisfy a certain prerequisite [which can be removed if needed] and mirrors the files in the telemetry folder to the USB device.

A result is returned in JSON Format that either implies a success in transfer, or any error which is thrown, such as removal of USB, insufficient disk space, etc.

`split_dirs`

This is a "helper" method that truncates a file name from the entire file path which is passed as an argument. An old implementation, can be replaced by using `rfind`

`transfer`

This is the heart and soul of the USB upload button and is built on patches after patches. This requires extensive optimization, but the core of the method is essentially the same; it loads the file list available for download(`new_files`), compares it with files that are already there (`old_files`) and removes those that overlap. These files are then modelled and kept ready for transfer. Successive GET requests download each file from the USB to the machine and reload the page [can be optimized by converting into a synchronous Jquery method]. If the save is successful, a DB entry is created and if not, `remove_corrupt_files` is called.

If there are no unique files in the USB device [those which don't already exist in the machine] then a message is displayed that informs the user so. If the user does a simple "refresh" after transferring all the files, the above condition does not occur, but instead, the webpage is just refreshed.

`remove_corrupt_files`

This is another helper function that executes a shell command to remove a file, the name of which is passed as an argument.