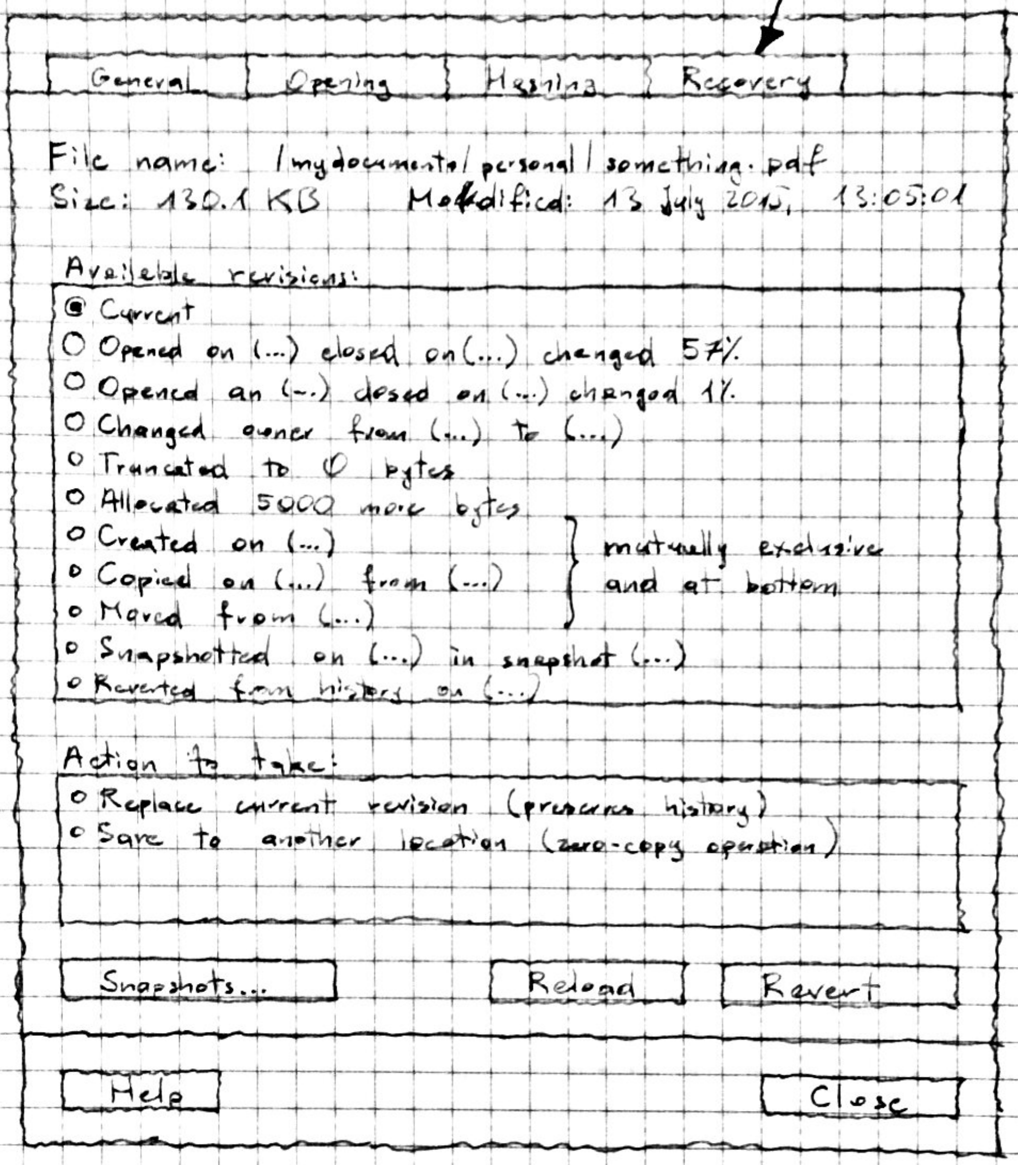
Nautilus-enabled recovery scenario GUI

for a single file!



File name: /mydocuments/personal/something.pdf
Size: 130.1 KB Modified: 13 July 2015, 13:05:01

Available revisions:

- ● Current
- ○ Opened on (...) closed on (...) changed 57%
- ○ Opened an (...) closed on (...) changed 1%
- ○ Changed owner from (...) to (...)
- ○ Truncated to 0 bytes
- ○ Allocated 5000 more bytes
- ○ Created on (...)
- ○ Copied on (...) from (...)
- ○ Moved from (...)
- ○ Snapshotted on (...) in snapshot (...)
- ○ Reverted from history on (...)

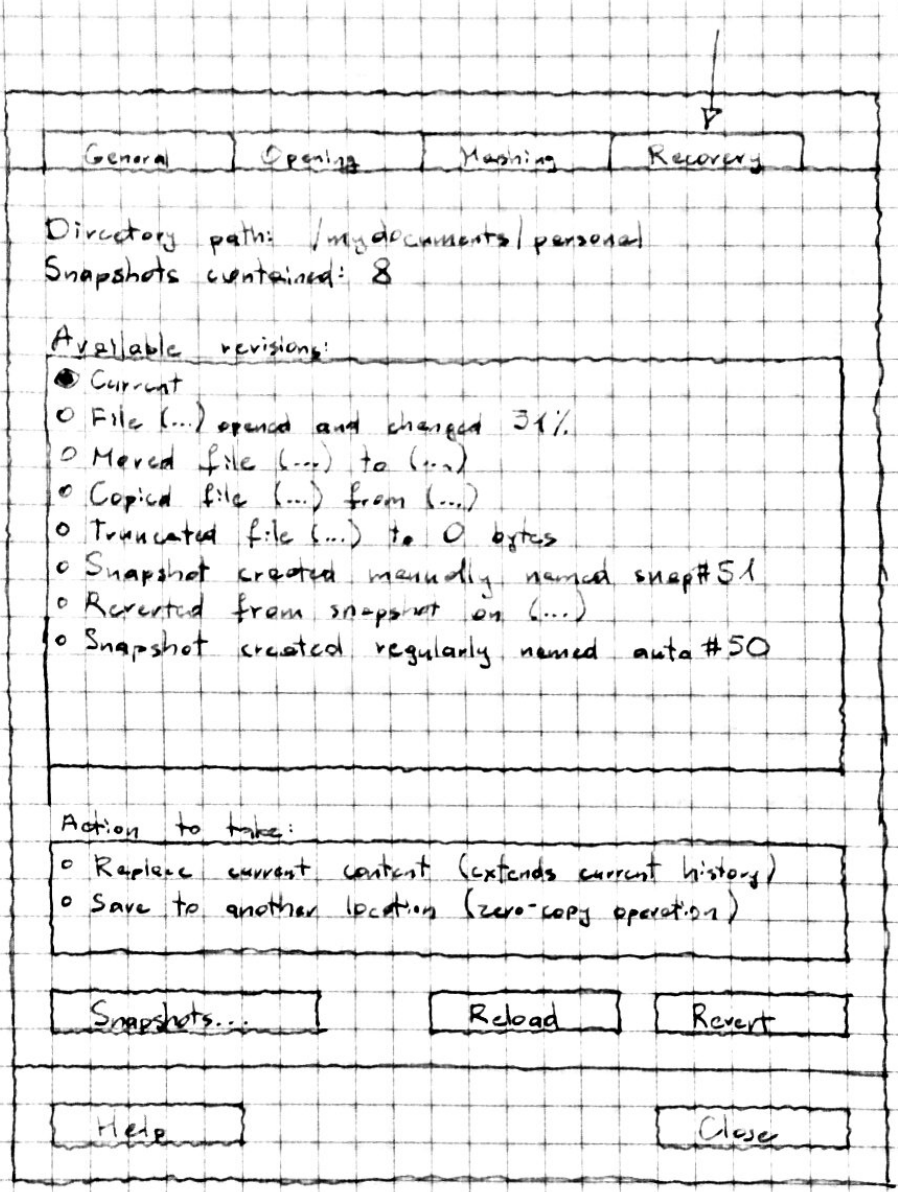
} mutually exclusive and at bottom

Action to take:

- ○ Replace current revision (preserves history)
- ○ Save to another location (zero-copy operation)

Nautilus-enabled recovery scenario GUI

for an entire folder



Directory path: /mydocuments/personal
Snapshots contained: 8

Available revisions:

- ● Current
- ○ File (...) opened and changed 31%
- ○ Moved file (...) to (...)
- ○ Copied file (...) from (...)
- ○ Truncated file (...) to 0 bytes
- ○ Snapshot created manually named snap#51
- ○ Reverted from snapshot on (...)
- ○ Snapshot created regularly named auto#50

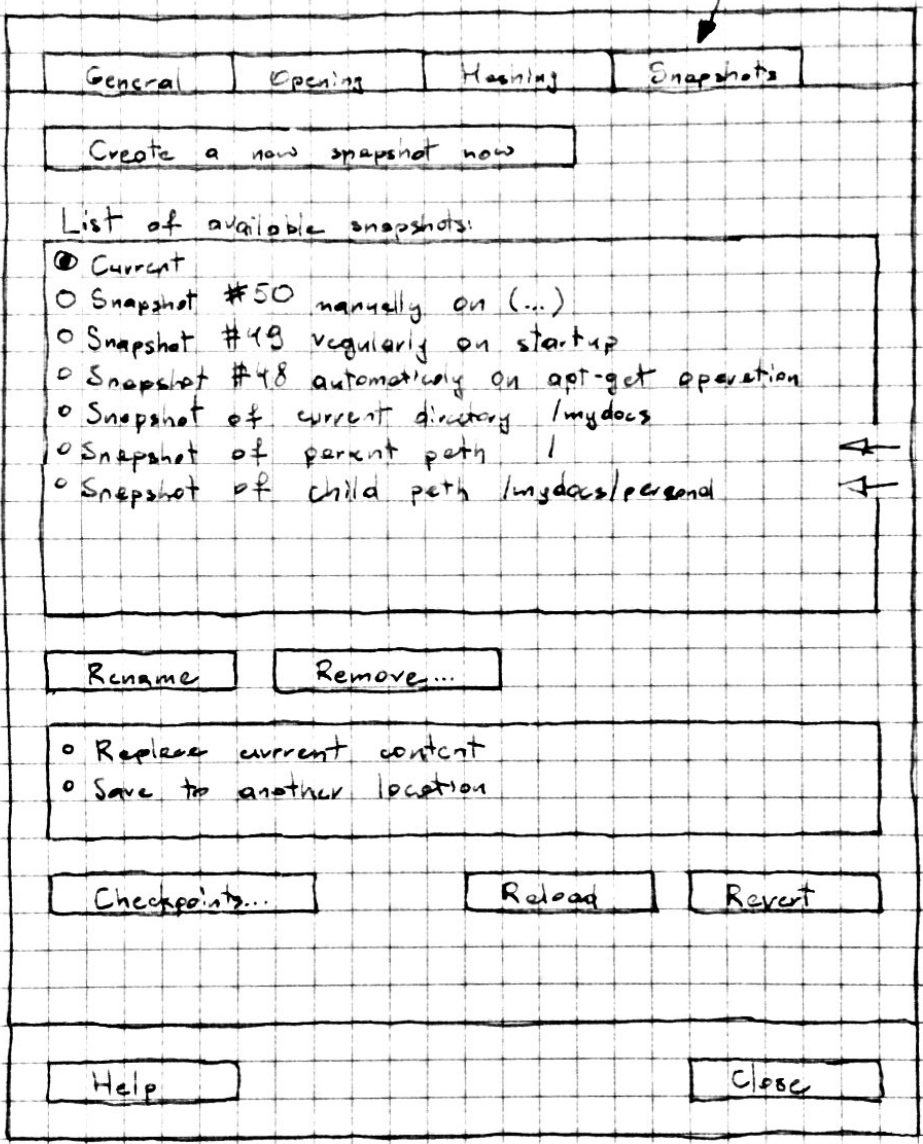
Action to take:

- ○ Replace current content (extends current history)
- ○ Save to another location (zero-copy operation)

- File moved into folder would be removed without recreating original file outside the folder. File moved outside the folder would be recreated without removing external one.
- Changes made after "Current" (last) checkpoint will be discarded as well as those listed.

Nautilus - enabled
recovery scenario GUI

~~Mange~~ Management of snapshots

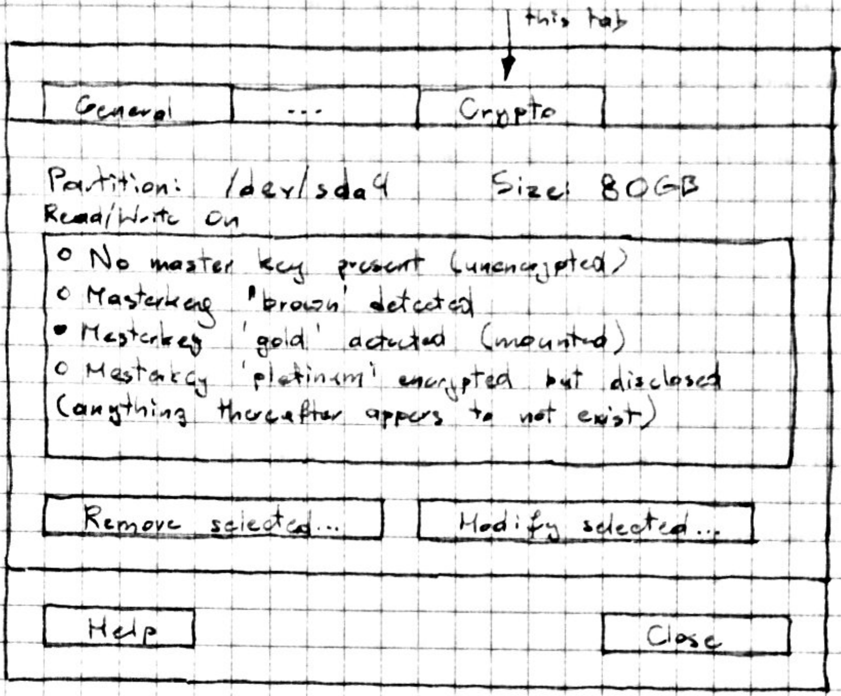


List of available snapshots:

- ● Current
- ○ Snapshot #50 manually on (...)
- ○ Snapshot #49 regularly on startup
- ○ Snapshot #48 automaticaly on apt-get operation
- ○ Snapshot of current directory /mydocs
- ○ Snapshot of parent path / ← blue color
- ○ Snapshot of child path /mydocs/personal ← green color

- ○ Replace current content
- ○ Save to another location

Encryption:



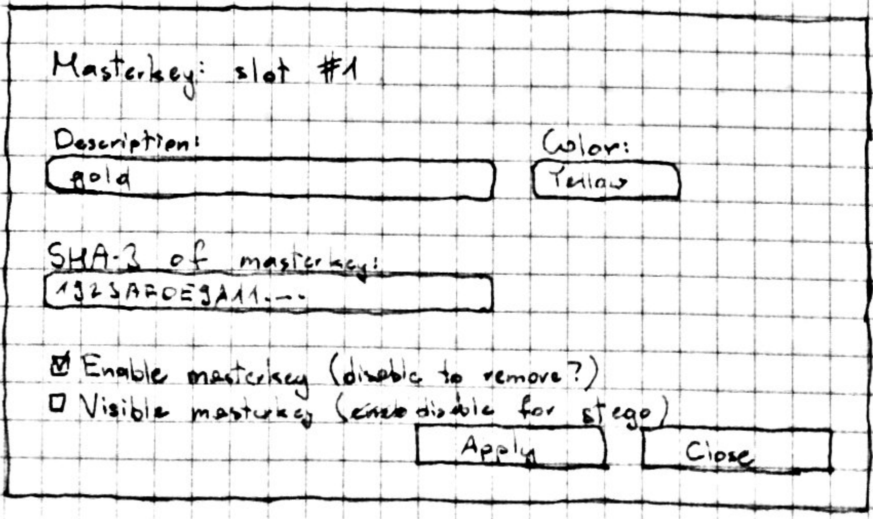
Partition: /dev/sda4 Size: 80GB
Read/Write on

○ No master key present (unencrypted)
○ Masterkey 'brown' detected
● Masterkey 'gold' detected (mounted)
○ Masterkey 'platinum' encrypted but disclosed
(anything thereafter appers to not exist)

- Add button is missing

○ Masterkeys can be either visible ~~for~~ (cryptographic confidentiality) or hidden (steganographic confidentiality).

~~• User Owner can opt out of encryption making all files unencrypted (speeding it up). Other Further masterkeys may exist despite of it.~~



Masterkey: slot #1

Description: gold

Color: Yellow

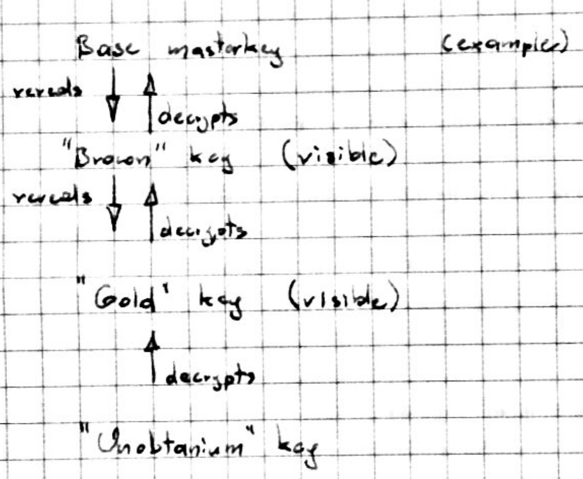
SHA-3 of masterkey: 1923AFOE9A11....

☑ Enable masterkey (disable to remove?)
□ Visible masterkey (~~can~~ disable for stego)

○ When edited, Apply turns on and Close turns into Revert

Concept of Masterkeys:

A filesystem has a hierarchy of masterkeys. They can be seen as different aspects (RubberhoseFS term) or ~~profiles~~ subvolumes (Btrfs term) or ~~whatever~~ "hidden volumes" in TrueCrypt. Base masterkey is unprotected and stored plainly in the header. Second masterkey IS the first password-protected and also allows to decrypt all masterkeys above/before. List of masterkeys goes on for unspecified ~~length. Var~~ length. Every MK has a name.



Every masterkey can be set visible ~~or not.~~ or hidden. Base MK is always visible by design. When unmounted, MKs are visible up to below first hidden masterkey (excluding it). When mounted, MKs are visible upto excluding first hidden key above mounted MK.

Having no password, 'gold' is visible. Having 'brown' password, 'gold' is visible. Having gold password, still, gold is visible. Having unobtanium, same is visible.

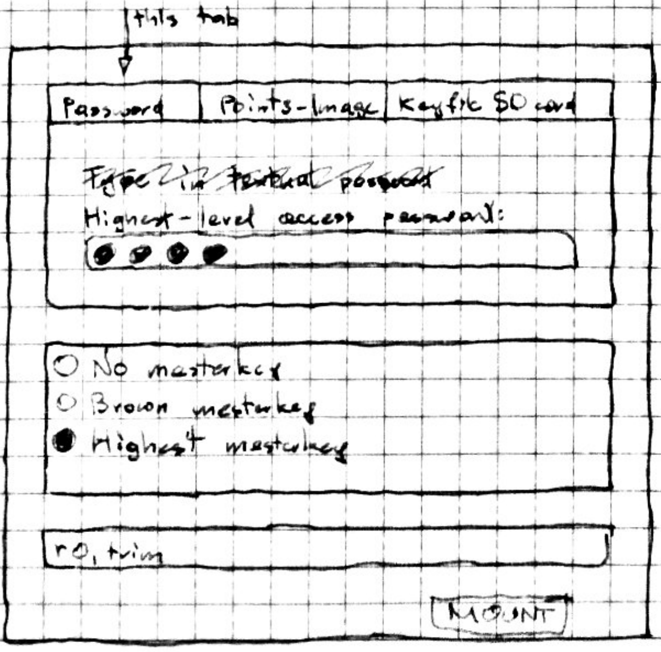
Motivation for masterkeys:

- Confidentiality and Authenticity guarantees are founded ~~on~~ (based) on an assumption that every bit of data is either unencrypted with no need to or encrypted with masterkey itself directly, another key derived with one-way function independent or encrypted with masterkey. By assumption, masterkey is not obtainable to an attacker, ~~mater~~ dragging cryptographic properties over further keys and then over ~~data~~ content itself.

- This creates purposfully a single point of failure. If needed masterkey can be destroyed making all ~~the~~ dependant data unaccessible in almost ~~no time~~, constant time.

Operational guide:

- ~~[illegible]~~ When mounting, user can be presented with visible masterkeys to choose ~~the~~ level of access he opts for, or just for informational ~~in~~ purposes.



~~Type in textual password~~
Highest-level access password:
●●●●

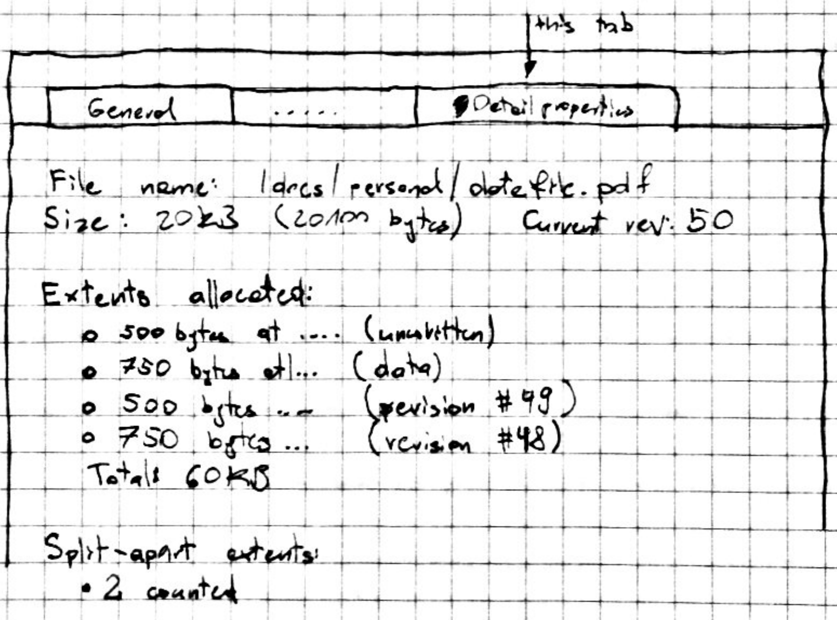
○ No masterkey
○ Brown masterkey
● Highest masterkey

r0, trim

~~obvious Compliance guerentee~~

~~Every call should either be compliant with POSIX/Linux standard or not, and clearly state non-compliance in documentation.~~

Nautilus enabled GUI for
file propeitics view



File name: /docs/personal/datafile.pdf
Size: 20kB (20100 bytes) Current rev: 50

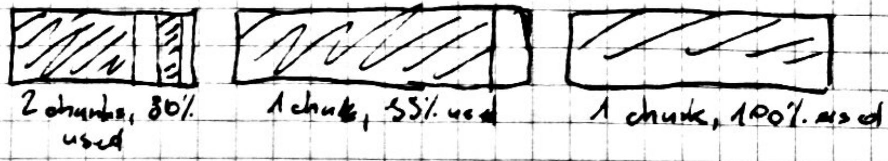
Extents allocated:

- 500 bytes at .... (unwritten)
- 750 bytes at ... (data)
- 500 bytes ... (revision #49)
- 750 bytes ... (revision #48)

Total: 60kB

Split-apart extents:

- 2 counted

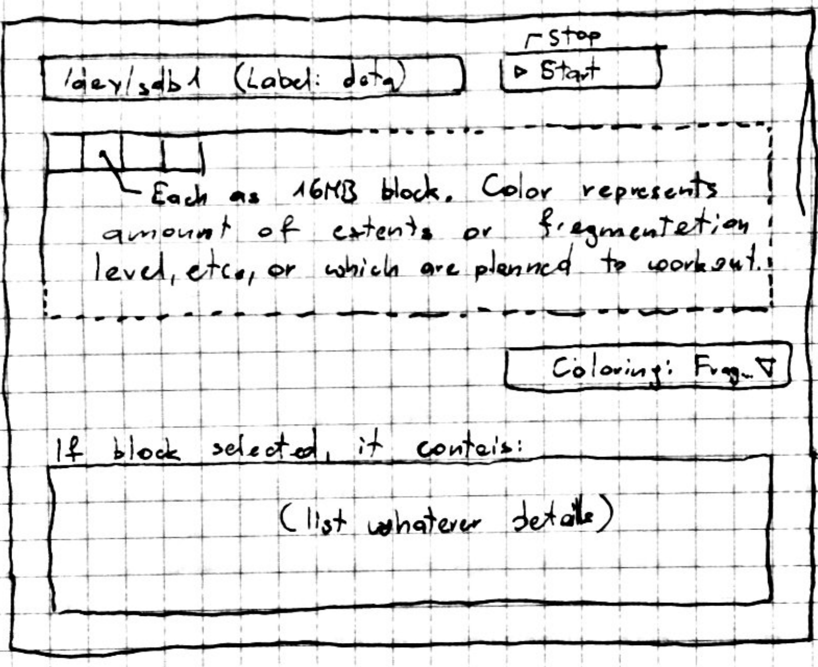


Total: 4 chunks, 88% used (180kB of 200kB)

On-boot profetching

~~Filesystem should be able to record which files are particularly important to quick bootup and allow to pretetch these files on mount. Its should be switchable, and either on/off by default.~~

Online defraigmentation GUI



Each as 16MB block. Color represents amount of extents or fragmentation level, etc., or which are planned to workout.

If block selected, it conteis:

(list whatever details)

Emergency deallocation tool:

To counteract a "clogged" filesystem, a situation where there is no free space left to remove any files (sounds paradoxical), user could select files to be deallocated (and overriden soon enough) which would allow the fs. to "unclog". Profetching-used extents could also be used this way.