TECHNISCHE UNIVERSITÄT
CHEMNITZ

Faculty of Computer Science
Department of Media Computer Science

Seminar on Android

Internal and External Storage

Richard Siegel

Chemnitz, 29th September 2017

Supervisor: Stefan Kahl

Contents

1. Introduction

Taking notes on a piece of paper may be less convenient than typing into your digital calendar or document file, but the paper note will never disappear for any strange reason as long as you hold on to it. However, most people have experienced the loss of their work due to a trivial computing error. Be it due to running out of power, switching apps or even just flipping the screen to landscape, sometimes everything done gets lost. How do we get apps to not forget about our settings or work in progress? Access to the Android file system it the solution. We can simply save all the progress information and settings, and reopen them when ever needed. If well implemented, the user will not even notice this kind of storage use, the app will just work more smoothly. But we also want to save files on external drives like SD cards, or simply the phone itself, to use and alter them later or with other apps. Sometimes we even want to copy them to different devices.
Therefore, this paper describes the basic principles on how to access and organize files within the android file system.

2. A Digital History of Files and Folders

There are no physical folders or files inside phones or storage cards. Digital storage devices simply store ones and zeros. About the inner architecture of SD cards one may reed, that *"Information is comprised of 1s and 0s which are stored as electrons inside the floating-gate transistor."*[1] However, in everyday life we experience SD cards as almost magical tiny cards, which seem to replace huge cabinets of folders and files. This is due to the conceptual model and *"its ability to provide meaning to things."* [2] *"A conceptual model is an explanation, usually highly simplified, of how something works. It doesn't have to be complete or even accurate as long as it is useful."* [2]

In the beginning of computing the raw data, being ones and zeros, was enough to satisfy the needs of a computer user. Calculations which required a set of input numbers and a set of outputs, which was often barely fitted into the storage capacities the computer, had no needs for the concept of computer files. As storage grew the challenge of maintaining data in usable structures grew too. The concept of the computer file was born. *"A file is a named collection of related data that appears to the user as a single, contiguous block of information and that is retained in storage."* [3]
To maintain the growing number of files manageable, eventually the concept of directories was born. *"The term directory is used [...] to refer to what appears to the user to be a container [...] that can hold files and other directories."* [3] To help the public in understanding computer storage, *"The term folder is used as a synonym for directory on the Microsoft Windows and Macintosh operating systems."* [3] Thus, today many people think of all the data on their computers in terms of files and folders, thanks to the easy conceptual model.
If we accept the common way to think of digital data as a conceptual model, rather than the truth about the nature of the data itself, we can allow ourselves to question it. Will files and folders be the best way to present digital data to the user forever? Probably not. On today smart phones photos and videos are often presented as previews of themselves grouped by the location or place where they have been taken. As a result photos on phones are normally not associated with file names or the folders they have been sorted into. Displaying pictures inside of a virtual gallery feels simply more natural. It only confuses smart phone users once they try to copy the photos onto their office computers, which usually displays the data of connected devices as folders and files. Users are presented with many files and folders, which have no connection to the photos they seek. It's not even clear what file or folder names are important to look for. It is therefore not always obvious to the user which folder holds the desired pictures. To overcome this problem, in *Android 7.1.1*, a message asks the phone user how it should present itself to the connected device (see Illustration 1). If the camera option is selected, connected devices should display the photos again, without the need for folder navigation (see Illustration 2).

If smart phone users choose to open apps like gallery or music-players over a file manager app (see Illustration 4) which show all the music and photos side by side in a less convenient way, files and folders may eventually disappear from the user perspective.
New technologies and designs will result in new conceptual models of digital data. The virtual reality environment of the future, partly possible on android devices today, may even result in concepts of data as items in virtual space.

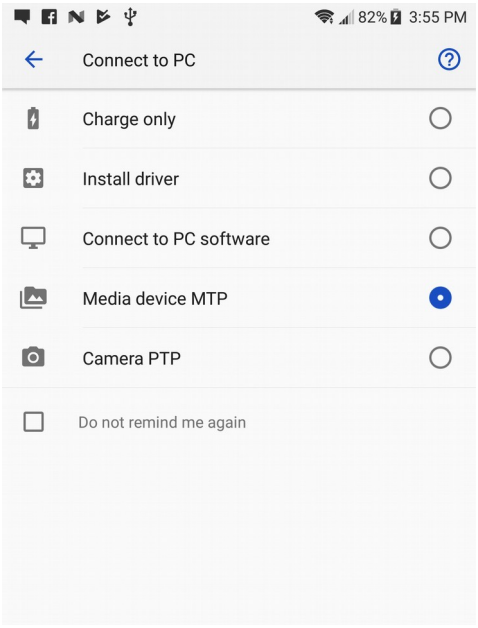

Illustration 1: Android 7.1.1 Connect Dialog

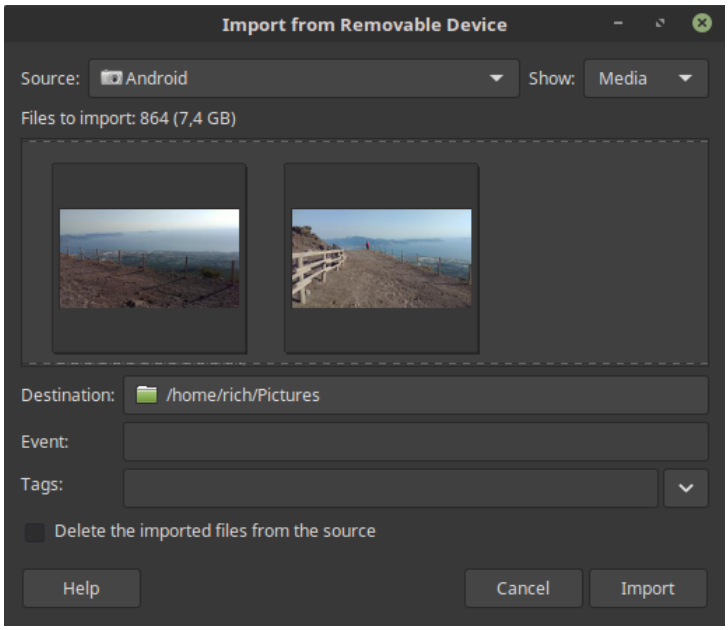

Illustration 2: Pix Photo Import Tool on Linux Mint 18 Sarah

3. Internal and External Storage in Android

3.1. The Permission of Storage Types

Android is based on the Linux kernel and therefore inherits the Unix like file system. This file system was created to be maintained by professionals, who thoughtfully decide when and how to install critical updates or new software. Consequently the users of such systems are denied permission on any attempts to change the system setup. In order to adapted to the needs of the average smartphone user, the permission system had to allow for important updates and app installs to be made. To insure the safety of the Android system, app permissions were introduced. The default storage policy of apps in Android is very restricted. All apps have permission to access a special folder, hidden away from all other apps. This safe space is usually referred to as internal storage, while storage which is shared between the different apps is called external storage. "These names come from the early days of Android, when most devices offered built-in non-volatile memory (internal storage), plus a removable storage medium such as a micro SD card (external storage)." [4] Today a physical removable medium is usually not necessary to utilize external storage. Such removable media are simply emulated.

Internal storage:

- *It's always available.*
- *Files saved here are accessible by only your app.*
- *When the user uninstalls your app, the system removes all your app's files from internal storage.*

Using Internal storage is best when you want to be sure that neither the user, nor other apps can access your files.

[4]

External storage:

- *It's not always available, because the user can mount the external storage as USB storage and in some cases remove it from the device.*
- *It's world-readable, so files saved here may be read outside of your control.*
- *When the user uninstalls your app, the system removes your app's files from here only if you save them in the directory from getExternalFilesDir().*

External storage is the best place for files that don't require access restrictions and for files that you want to share with other apps or allow the user to access with a computer.

3.2. The Basics of Read and Write Implementations in Java

Access to either storage type can be implemented in *Android Studio*. In the example project *UrNote*, writing (See *MainActivity.java* : 113) and reading (See *MainActivity.java* : 138) of both storage types are featured. The principles of storage operations in Java for Android will be explained here:

1. External storage locations are always provided by the *android.os.Environment*. However, internal storage only exists in the *Context* of the application.

   ```
   Context appContext = getApplicationContex();
   ```

2. For many read and write operations a *File* object, which points to either a file or a folder, has to be created. To access internal storage the application context is utilized to create *File* objects. Its methods *getFilesDir()* or *getCacheDir()* help to access the internal folders *"files"* or *"cache"*. Be aware: *"If the system begins running low on storage, it may delete your cache files without warning."* [4]

   ```
   File internalFolder = appContext.getFilesDir();
   ```

   To access the root folder of the SD card (External Storage) one may utilize:

   ```
   File externalFolder = Environment.getExternalStorageDirectory();
   ```

   Public directories are also easy to load in the following fashion:

   ```
   File documentsFolder = Environment
      .getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUMENTS);
   ```

   Similar enumerations to *"DIRECTORY_DOCUMENTS"* are provided for music, pictures etc. for example *"DIRECTORY_DCIM"* for pictures.

3. To actually write a file into the directories we have loaded into file variables, we need to create the new files as variables too.

```
File newFile = new File(internalFolder, filename);
```

4. So far *newFile* is just a variable. We can write it into internal storage by creating a *FileOutputStream*. To write into external storage android permissions have to be set, the rest is the same.

```
FileOutputStream fos = new FileOutputStream(newFile);
```

5. We are ready to save bytes. In this simple example we assume that the *fileContent* variable holds a string value. When all bytes have been written, we close the *FileOutputStream*.

```
fos.write(fileContent.getBytes());
fos.close();
```

6. Reading files is in principle very similar. Instead of the *FileOutputStream*, a *FileInputStream* has to be created. The method *read()* loads files bytewise and could therefore be utilized to read single bytes like in the following example:

```
FileInputStream fis = new FileInputStream(loadFile);
int fileContent = fis.read();
fis.close();
```

However, usually we are not trying to read single bytes to store them into integer variables.

3.3. Loading Text Files with a Buffered FileInputStream

To load a text file from storage into our android application, we need to convert the raw bytes of the *FileInputStream* into characters and concatenate them into string format. These are the functions of the *InputStreamReader* and *BufferedReader*:

```
FileInputStream fis = new FileInputStream(textile);
InputStreamReader isr = new InputStreamReader(fis);
BufferedReader br = new BufferedReader(isr);
```

The *BufferedReader* already holds the string values we seek. To collect them we simply iterate through the *BufferedReader* until it holds no more lines. In our Example (see *MainActivity.java* : 140) we make use of a *StringBuffer* to collect the whole text and carry it into a *textView*:

```
StringBuffer sb = new StringBuffer();
String note = "";
while ((note = br.readLine()) != null) {
      sb.append(note + "\n");
}

textView.setText(sb);

fis.close();
```

3.4. Gaining External Storage Permissions

Like all kinds of android permissions, external storage permissions can be requested in the manifest file (*AndroidManifest.xml*). In case we desire to exclusively read the content of the external storage, like music files in music player apps, only read permissions should be requested.

```
<usespermission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"
android:required="true" />
```

If one needs to be able to save files to the external storage the permission should match the following:

```
<usespermission
android:name="android.permission.READ_EXTERNAL_STORAGE"
android:required="true" />
```

"When you declare android:required="false" for a feature, it means that the application prefers to use the feature if present on the device, but that it is designed to function without the specified feature, if necessary." [13]

4. File-Browsing and Mobile-Usability.

4.1. The Android of Invisible Files.

To android developers files and the understanding of storage types are certainly an important tool in the creation of good applications. However, smartphone users sometimes do not even realize that files are presented on their phones. Thus, many applications integrate files so smoothly, that users do not need to understand whether they are using a file. For example, to send a recently taken picture to a friend, most people will simply ask themselves how they may accomplish sending the picture, not the image file. Although they are selecting a file, by tapping on the image, which they want to attach to their message, to them it is simply a picture. The action of adding this picture to their message may be technically very different, but to the user feels extremely similar to adding an emoticon. As there is no need to think about the files, the conceptual model of the user is simplified. Thus, the usability of the application increases due to the invisibility of the files behind the picture.

"Good design is actually a lot harder to notice than poor design, in part because good designs fit our needs so well that the design is invisible, serving us without drawing attention to itself." [2]

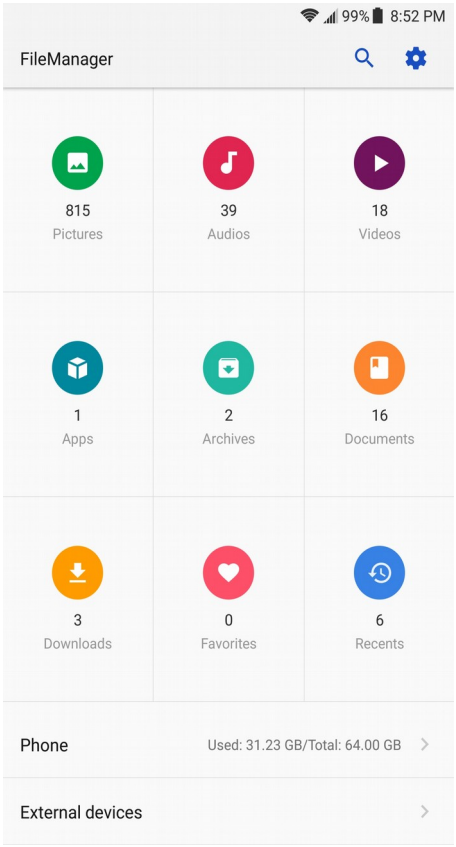

Illustration 3: FileManager (Android 7.1.1)

During the research for this paper, an article with the title "*How to Access Stored Files on an Android Phone*", was found. It illustrates how some users may think about conscious file management on their phones.

"You probably didn't get an Android phone because you enjoy managing files on a computer and wanted another gizmo to hone your skills. Even so, you can practice the same type of file manipulation on a phone as you would on a computer. Is there a need to do so? Of course not! But if you want to get dirty with files, you can." [5]

Even if *"you want to get dirty with files"* [5], the application *"FileManager" (Android 7.1.1)* attempts to present us a tamed file system by grouping similar files. Directory like container structures are reduced or eliminated within the resulting groups. The *"Phone"* button would allow us to enter the file system, but colorful icons convey the user into utilizing the simplified structure. (see Illustration 3)

4.2. The Relation Between Applications and Files

Desktops computers usually have a directory named *"Desktop"* which holds all files the users see on the screen as soon as they are logged into their account. Applications are started by selecting a file which indicates the application it is supported by. Alternative a link to the actual application can often be clicked. Android is different. No directory is associated with the icons visible on the phones home screen. Only applications are accessible through home screen icons. Due to the relative ease of application access on the home screen files are seldom accessed through file manager applications. If the android user desires to open a file, a click onto the right application icon is the foreseen way. Thus, applications for android are expected to incloud file browsing tools, to open the desired file. To enhance the usability, the knowledge about the desired actions, revealed through the choice of application, can be utilized to discriminate files. If well implemented, the remaining files are then displayed in the context of the application, such that they are often preserved as pictures, songs or other items.
Consider the following example: Within the directory "*Phone/documents/TestFiles*" six files of three different media types have been created. Four different applications are tested for their method of making relevant content available to the user. All six files are visible within the *FileManager* application (see Illustration 4). In *Gallery* (see Illustration 6) only two image files are visible as pictures within the *TestFiles*

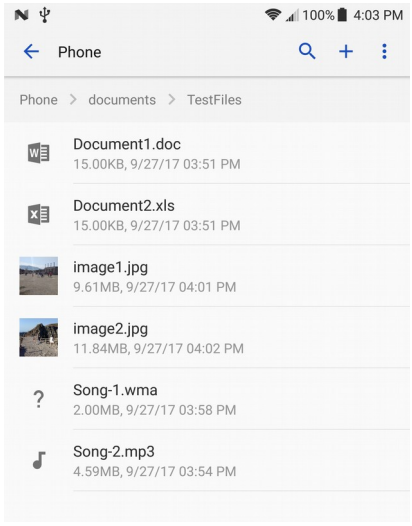

Illustration 4: FileManager view of the TestFiles directory (Android 7.1.1)

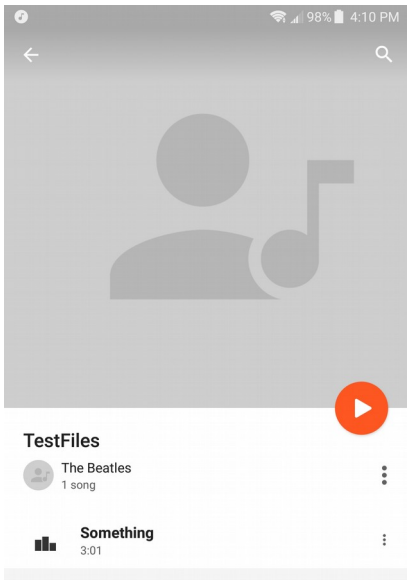

Illustration 5: Play Music view of the TestFiles directory (Android 7.1.1)

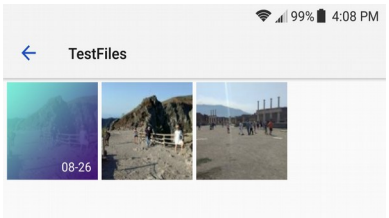

Illustration 6: Gallery view of the TestFiles directory (Android 7.1.1)

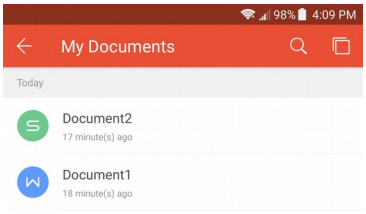

Illustration 7: WPS Office showing content of the TestFiles directory (Android 7.1.1)

container. The *Play Music* application shows just the audio file which is compatible with it and declares it part of the album *TestFiles* (see Illustration 5). Both office documents are listed in the category *My Documents* of *WPS Office* (see Illustration 7). Because of the well implemented discrimination of available files, every app in this example made only the relevant files immediately visible. (Of course *FileManager* is designed to skip file discrimination.) As a result users should be able to quickly find their desired content.

"Two of the most important characteristics of good design are discoverability and understanding. Discoverability: Is it possible to even figure out what actions are possible and where and how to perform them? Understanding: What does it all mean? How is the product supposed to be used?" [2]

4.3. Files and Googles Material Design

Material Design is a design language for android created by *Google* to be intuitively understandable. It aims to create a subconscious conceptual model of user interfaces, which compares them to layers of *paper* with different contents. The color and texture of the content is called *ink* as if it had been printed on actual paper. Contents could be title bars, articles or many other app components. *"In material design, the physical properties of paper are translated to the screen. The background of an application resembles the flat, opaque texture of a sheet of paper, and an application's behavior mimics paper's ability to be re-sized, shuffled, and bound together in multiple sheets."* [6]
An important part of *Material Design* is the feedback. Donald Norman, who insists on the importance of immediate feedback in design, writes, *"Feedback is critical to managing expectations, and good design provides this."* [2] The visual feedback provided in *Material Design* is discreet and pays attention to detail like shadows which change with the hight of *paper*. A description of feedback from the card (a component made of *paper*) reads, *"Ink ripples confirm user input by immediately expanding outward from the point of touch. The card lifts to indicate an active state."* [7]

Employing *Googles Material Design* is an easy way to create professional app designs. Therefore the question, "How to best organize files in *Material Design*?", may arise. However, the example in 4.1. shows how much the interpretation of files may differ depending on their content and the design of the individual application. To create the right visual file representation *Google* provides a set of guidelines (see [8]). Commonly applied components with the ability to display a set of items are: *Cards, Grid lists* and *Lists*.

Name	*Cards*	*Grid lists*	*Lists*
Description	*"Cards may contain a photo, text, and a link about a single subject. They may display content containing elements of varying size, such as photos with captions of variable length.* *A card collection is a layout of cards on the same plane."* [9]	*"A grid list consists of a repeated pattern of cells arrayed in a vertical and horizontal layout."* [10]	*"Lists are made up of a continuous column of rows. Each row contains a tile. Primary actions fill the tile, and supplemental actions are represented by icons and text.* *Lists are best suited for similar data types."* [11]
style	*"Used for content with inconsistent formatting, such as photos with captions of variable length, or data sets with*	*"A grid list is best suited to presenting homogeneous data, typically images, and is optimized for visual*	*"Lists are best suited to presenting a homogeneous data type or sets of data types, such as images and text. They are optimized for*

	heterogeneous content, such as a mixed collection of photos and videos and books." [10]	*comprehension and differentiating between similar data types."* [10]	*reading comprehension while differentiating either between similar data types, or qualities within a single data type."* [11]
usage	*"If more than three lines of text need to be shown in list tiles, use cards instead"* [11]	*"If the primary distinguishing content consists of images, use a grid list"* [11]	If the item caption is the most essential part for differentiating multiple items.

If necessary, navigational elements may be employed to differentiate file categories or directories and containers. Fewer such categories or short category names suggest the use of *Tabs*. *Tabs* keep the options always visible and therefore easy to find. Long category names or a larger number of them are better suited for the *Navigation Drawers*. However, a separate activity should be considered (see Illustration 3). One should try to avoid useless categories or navigational elements.

"Many products defy understanding simply because they have too many functions and controls." [2]

It is often possible to guess a probable frequent desire of the general app user. The actions needed to fulfill such desires are primary actions in an application. Creating a new item or file, to start working with a blank object, could be one such wish. In application design we should create an affordance, which helps users to achieve their goals (e.g. creating a blank file) easily.

"An affordance is a relationship between the properties of an object and the capabilities of the agent that determine just how the object could possibly be used. A chair affords ("is for") support and, therefore, affords sitting. Most chairs can also be carried by a single person (they afford lifting), but some can only be lifted by a strong person or by a team of people. If young or relatively weak people cannot lift a chair, then for these people, the chair does not have that affordance, it does not afford lifting." [2]

If an affordance has been created (e.g. a function which creates a file in the current directory) its availability has to be communicated to the user.

"Affordances exist even if they are not visible. For designers, their visibility is critical: visible affordances provide strong clues to the operations of things." [2]

In *Material Design*, the *Floating Action Button* is a good signifier for primary actions. "*Shaped like a circled icon floating above the UI, it changes color upon focus and lifts upon selection. When pressed, it may contain more related actions."* [12] *"Only one floating action button is recommended per screen to represent the most common action."* [12]

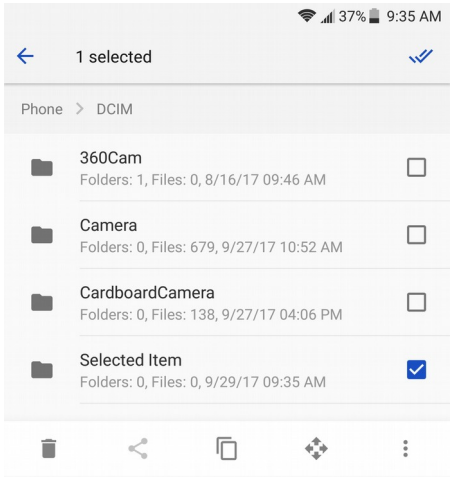

Illustration 8: FileManager with Bottom Navigation Bar (Android 7.1.1)

If one desires to communicate secondary actions associated with file selection (e.g. the ability to delete a file), a *Bottom Navigation Bar* is a good choice (see Illustration 8). It provides easily comprehensible icons, while remaining hidden as long as it is not needed. A common practice to activate it, is on the selection of one or multiple files. Holding an item pressed for one second, instead of tapping it, often engages selection mode. In selection mode other items are added to the selection by tapping (see Illustration 8). To create useful feedback the selected items, and selection mode, should always be communicated with signifiers (e.g. checkboxes). Signaling mode changes to avoid errors is crucial to remain a clean and usable design. *"Mode errors are inevitable in*

anything that has more possible actions than it has controls [...] that is, the controls mean different things in the different modes. This is unavoidable as we add more and more functions to our devices." [2] Always avoid to provide more functions than needed and insure the quality and visibility of the provided functions.

Note that signifiers do not always need to attract much attention. To give an example, in the *FileManager* (see Illustration 4), the nondescript line *"Used: 31.23 GB/Total: 64.00 GB"* helps to attract the attention of experts or knowledgeable users while representing useful information. As previously described (see 4.1.) less technically versed users may prefer to avoid the classical file system. Thus, favorably the technicality of the described line, attracts only the users who seek to proceed to the file system. In support of the gray arrow, the contrast of the text line (behind the *Phone* option) to the void (behind the *External devices* option) also signifies the affordance to access the file system.

5. Visions for a more Natural Interaction with Files

5.1. Abolish the Need to Save Files

In reality files only need to be saved if something threatens to destroy them. Digital systems constantly require us to save them. This way of file handling is unnatural and can be frustrating. To minimize damage caused by unexpected file loss many programs already automatically create backup files and suggest to recover the last backed up file state. Of cause this is helpful, but it still feels unnatural. Why do we force users into saving their files? Design should take care of saving files for them. The nature of our reality tempts people to create a conceptual model of computer files, which is more similar to real files. No one is afraid of loosing the lines written into a traditional paper letter while writing. Of cause why should anyone be worried of that? It does not happen in the real world, but we let it happen in the virtual world we created. A well designed application should aim to recover every change made to a document even if the device crashed during its very creation. To ask the user whether he or she wants to recover the document is unnecessary. Undo-functions could provide the option to return to the previous stages in which the document was found when it was opened before.
New users should be happier with the described approach because it dose not require to be learned. Mobile Usability has also been increased, since running out of power, or closing the application in a hurry becomes less of a thread to the accomplished work. For those users who expect the necessity to save a file manually, the absence of the option to do so is probably confusing and unpleasant. Moreover the save option should be available for those who want to use it. It dose not have to be a placebo button, but could provide the user with a positive feedback. It should be clearly stated, that the file has been saved (already). For conscious users a "read more" option could provide an explanation of the simplification and the option to save the file to a specific directory. Users who keep using the save button will also continue to feel satisfied due to the conformation. Additionally the save option could create save states, which may be returned to with the undo option as well. The new undo options should only be revealed as possibilities to the user, once the undo button has been pressed at least 3 times in a row, as pressing it frequently would be a natural attempt to return to a much older version of the document. The important option of choosing a file location (e.g. on a thumb drive), should be presented as a variation on sharing the file. After all one could try to share the file via e-mail or could-storage with ones own accounts.

5.2. Perfecting File Representation

"It is the duty of machines and those who design them to understand people." [2]. As described in 4.1. an 4.2. files are often not represented or understood as files. Smartphons are perfectly capable of presenting files of two-dimensional images graphically in a gallery. Users can slide pictures to the side, similar to moving physical photos on a table. Sound files, videos or office documents, are not as conveniently represented by todays standards. However, without restrictions to our creativity, how would humans naturally expect to experience different kinds of data? Let us Imagen truly endless possibilities and all possible kinds of data:

Images: No matter if big or small, gray or colorful, two-dimensional or three- dimensional images are a purely visual media. The natural way to preserve images is to see them with our eyes. The whole spectrum of colors and brightness or dimness should be available to realistically present images, as long as it causes no damage to the eyes. To gather visual information in nature we simply look or walk around to change our perspectives. A virtual gallery of our images would be great for active personalities in humans. Turning around and even walking into surrounding three-dimensional images would be possible. We may also move the objects we experience visually. The perfect way of browsing images could be a card, cube or another object, which displays an image on every side and changes the naturally invisible images on its back depending on the way in which we rotate it. Ideally the object would be easily resizable and thereby enable us to zoom into the images for greater detail.

Music and Sounds: Everything hearable is caused by vibration. We preserve as deep sounds what is caused by slow and vibrations. Beyond the spectrum of hearable vibrations we can feel deep vibrations with our whole body. Sound is thus, preserved with both auditory and tactile senses. In nature we can move towards sounds, and away from them. Sound interacts with the shape and texture of its environment. Air, watter and solids of all kinds and surface structures have an effect on sound, as they act as a filter for vibrations. Consequently a supernatural environment could create the realistic illusion of sound sources which we can walk or simply look into. The experience should acoustically feel similar to an easy journey through music halls of various kinds. Extreme sounds which damage human hearing should be avoided.

Videos: Videos combine images with sound. Characteristic previews of the now continuously changing stream of images should be repeated while browsing through the videos is possible. Surrounding three-dimensional videos may be miniaturized and maybe only sections of such videos would be visible while browsing.

Navigation: Navigation should, in the best of cases, be intelligent enough, to present us only a set of very few desired data entities. If navigation in a directory like container structure is required, it should be associated with directional movements. No matter which direction we move to enter a container, leaving the container should be associated with an intuitive moment into the opposite direction. Navigation may always be supported by intelligent movement of the environment.

No doubt, smartphones and tablet computers are already extremely close to fulfill image browsing experiences similar to the described objects and virtual reality may enable us to realize many of these ideas. The navigation through sound and video items, especially mixed content, could be improved very much. Even the navigation on user-friendly file browsers requires to press the learned return-arrow symbol instead of intuitive directional movements. There is always room for improvement.

6. File Browsing in VR-Envirnoments

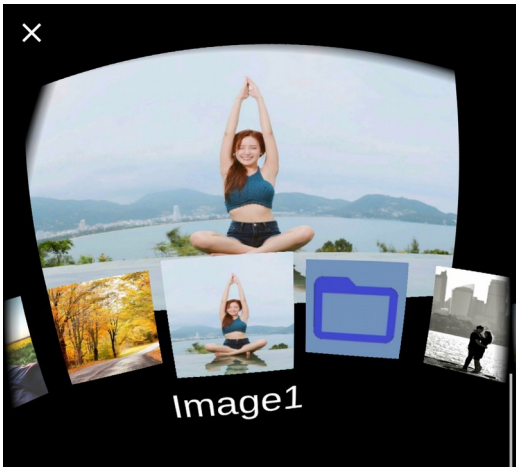

Illustration 9: VR-Files-Demo.apk running inside Google Cardboard

The potential of virtual reality seems promising, but as of 2017 productive work within virtual reality is hardly ever heard of. Games and entertainment content seem to be the main target for virtual realty devices. To support three-dimensional prototype design or medical training within virtual reality to gain popularity, the basics of virtual realty computing layout should be created. Even if it may be avoided most of the time, the navigation within a file system should be manageable at a productive level. A demonstration of the basics of a usable design for file management in virtual reality has been created to illustrate the concepts of natural engagement with the file system (see *VR-Files-Demo.apk* file). To insure productivity within a system similar to the demonstration, file management as suggest would need to be further developed, and most importantly integrated into a system which would allow the user to enter other VR applications within the VR-Goggles. It is recommended to install the apk file (see *VR-Files-Demo.apk* file) for *Google Cardboard* on a compatible device to enhance the understanding of the following statements. Once the application has been started a row of multiple pictures and one folder icon appear in the foreground. The pictures represent image files. The big image behind the files is called a *virtual screen*. If the user looks towards the *screen* the content of the last selected file is visible. To make room for the content the files are automatically lowered. By simply looking down towards the icons, which will rise up into the view of the user, it is possible to select other items. The concept is designed for users who want to quickly go through there files just by looking to the right and left sides. Once the desire to view a file in detail arises, users will automatically look up because the preview change is always evident. Unlike this demonstration, real file systems may contain files which will take a while to load to the *screen*. In such cases the *screen* should display only a blurred out preview until the user looks up and thereby decides to load the file more clearly. Note that turning the head to the sides, to view other items, triggers the items to move against the turning direction of the head. This makes it easier to reach many items with comfortable head movements. Like many things the design of directories in the demonstration should be improved, however the functionality well demonstrates how directories could be animated in Virtual Reality. The directory content is floating in from above, to close the folder again the user is supposed to simply look down towards the parent directory. Sound files are hearable as a preview when they are selected.

7. Summery

Some users dislike to work with classical file systems. File discrimination in the context of the application saves time, which users would otherwise need to spend by searching through useless files. Alternative methods of displaying or symbolizing files should meet the natural expectations of application users. Application interfaces should emphasize only primary actions, while secondary actions should remain hidden until they are needed. Consequently it can be summarized that the key to the usability of an application lies in the simplicity of its conceptual model.
Future technologies will increase the possibilities for intuitive design. Only sufficient conceptual models will remain popular.

Referances

[1] A Flash Storage Technical and Economic Primer. Access: 28.10.2017. Link: http://www.flashstorage.com/flash-storage-technical-economic-primer/

[2] Don Norman “The Design of Everyday Things”. Access: 28.10.2017. Published by Basic Books, 2013

[3] Directory Definition. Access: 27.10.2017. Link: http://www.linfo.org/directory.html

[4] Developer Android. Access: 28.10.2017. Link: https://developer.android.com/training/basics/data-storage/files.html

[5] Dan Gookin “HOW TO ACCESS STORED FILES ON AN ANDROID PHONE”. Access: 27.10.2017. Link: http://www.dummies.com/consumer-electronics/smartphones/droid/how-to-access-stored-files-on-an-android-phone/

[6] Material Motion. Access: 29.10.2017. Link: https://material.io/guidelines/layout/principles.html)

[7] Material Motion – how does material move. Access: 27.10.2017. Link: (https://material.io/guidelines/motion/material-motion.html#material-motion-how-does-material-move)

[8] Material Design – guidelines. Access: 28.10.2017. Link: https://material.io/guidelines

[9] Material Design – components. Cards. Access: 29.10.2017. Link: (https://material.io/guidelines/components/cards.html)

[10] Material Design – components. Grid-Lists. Access: 25.10.2017. Link: (https://material.io/guidelines/components/grid-lists.html)

[11] Material Design – components. Lists. Access: 27.10.2017. Link: (https://material.io/guidelines/components/lists.html)

[12] Material Design – components. Buttons. Floating action button. Access: 29.10.2017. Link: . (https://material.io/guidelines/components/buttons-floating-action-button.html)

[13] Uses-feature. Access: 26.10.2017. Link: https://developer.android.com/guide/topics/manifest/uses-feature-element.html

Appendix

- VR-Files-Demo.apk
- AndroidManifest.xml
- MainActivity.java