

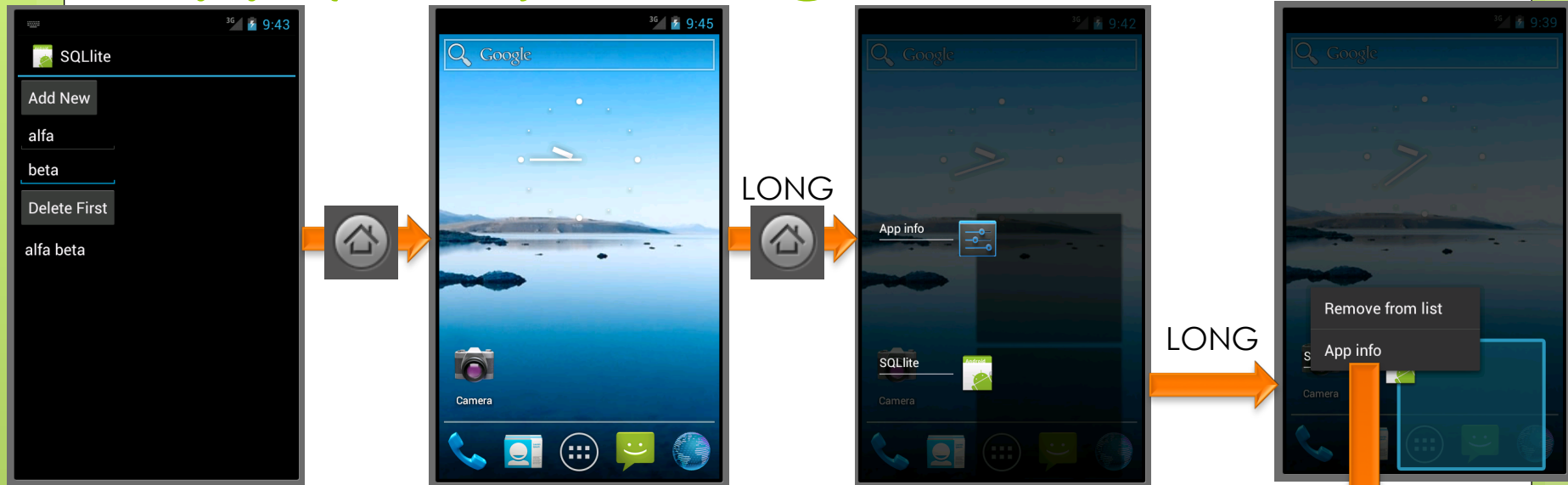


# How to access your database from the development environment

Marco Ronchetti  
Università degli Studi di Trento

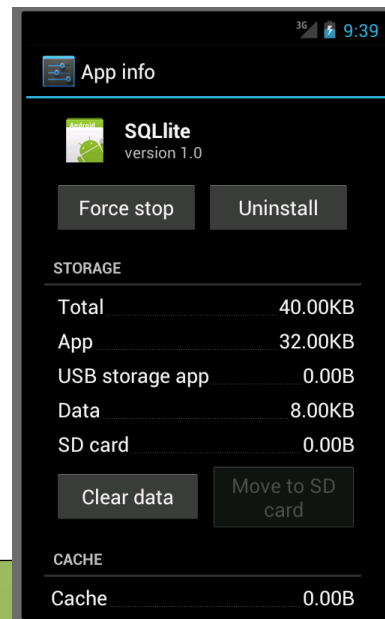
---

# App (data) management

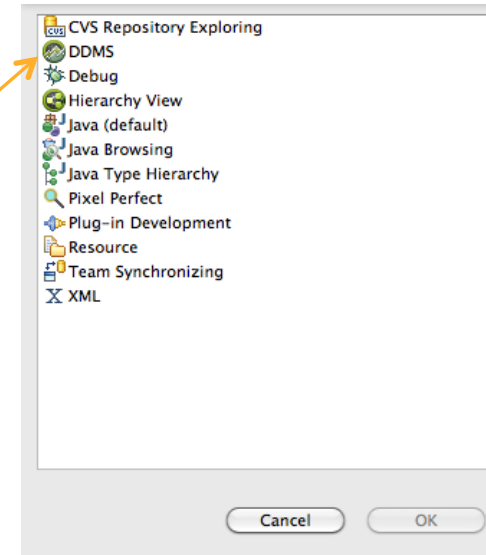
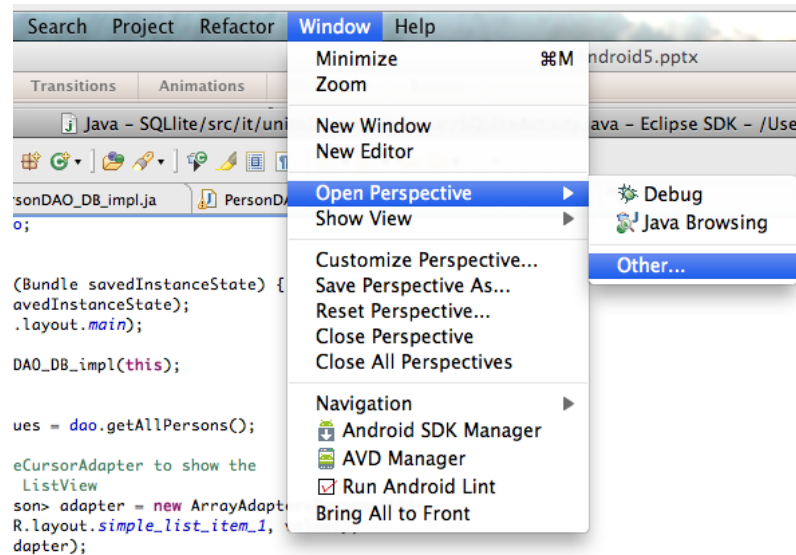


App management

Data management

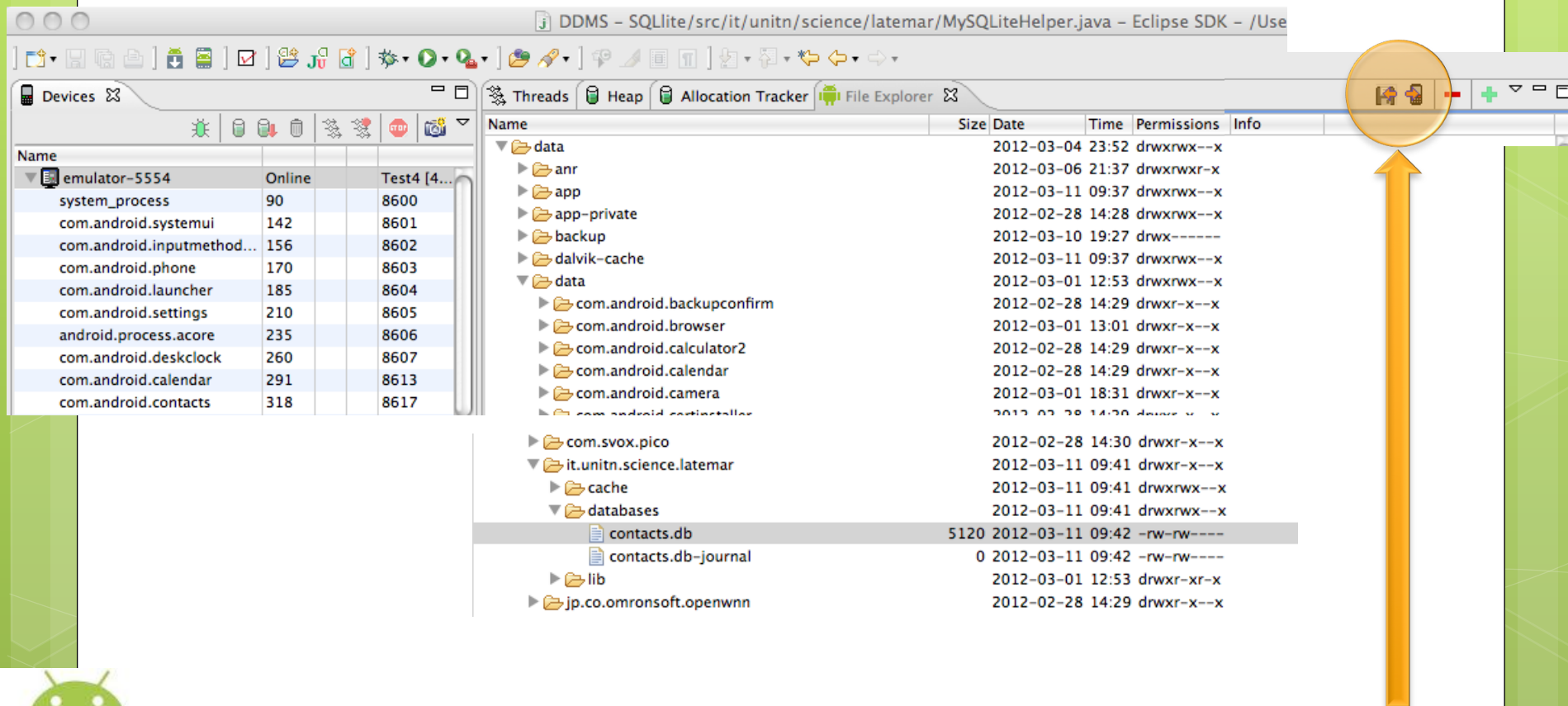


# Open the DDMS Perspective



# Find your DB

1) Look into `data/data/YOURPACKAGE/databases/YOURDATABASE.db`



The screenshot shows the Eclipse IDE with the DDMS File Explorer. The file explorer is open to the 'data' directory of an Android emulator. The 'databases' folder is expanded, showing 'contacts.db' and 'contacts.db-journal'. An orange arrow points from the 'contacts.db' file to a circular icon in the top right corner of the File Explorer window, which represents the pull action to the host.

Name	Size	Date	Time	Permissions	Info
data		2012-03-04	23:52	drwxrwx--x	
anr		2012-03-06	21:37	drwxrwxr-x	
app		2012-03-11	09:37	drwxrwx--x	
app-private		2012-02-28	14:28	drwxrwx--x	
backup		2012-03-10	19:27	drwx-----	
dalvik-cache		2012-03-11	09:37	drwxrwx--x	
data		2012-03-01	12:53	drwxrwx--x	
com.android.backupconfirm		2012-02-28	14:29	drwxr-x--x	
com.android.browser		2012-03-01	13:01	drwxr-x--x	
com.android.calculator2		2012-02-28	14:29	drwxr-x--x	
com.android.calendar		2012-02-28	14:29	drwxr-x--x	
com.android.camera		2012-03-01	18:31	drwxr-x--x	
com.android.contacts		2012-03-01	14:29	drwxr-x--x	
com.svox.pico		2012-02-28	14:30	drwxr-x--x	
it.unitn.science.latemar		2012-03-11	09:41	drwxr-x--x	
cache		2012-03-11	09:41	drwxrwx--x	
databases		2012-03-11	09:41	drwxrwx--x	
contacts.db	5120	2012-03-11	09:42	-rw-rw----	
contacts.db-journal	0	2012-03-11	09:42	-rw-rw----	
lib		2012-03-01	12:53	drwxr-xr-x	
jp.co.omronsoft.openwnn		2012-02-28	14:29	drwxr-x--x	

2) Pull the file on your PC

3) Use sqlite on your PC (in your\_sdk\_dir/tools)



# Access your DB

Use the following script, and

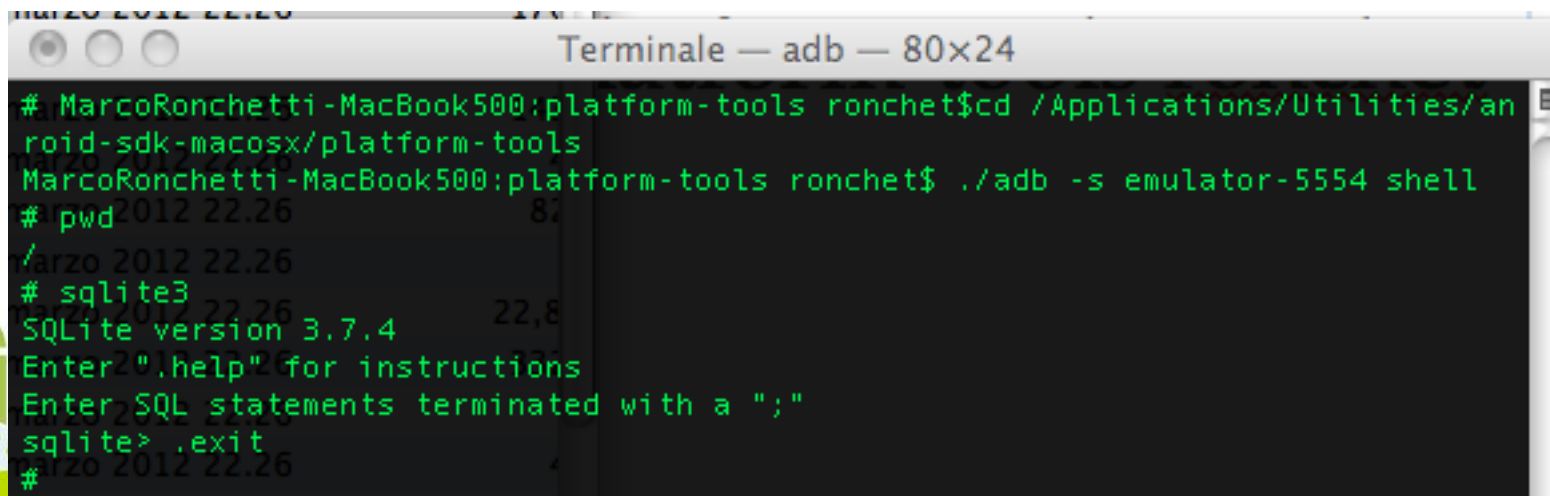
```
#!/sh
adb shell "chmod 777 /data/data/com.mypackage/databases/store.db"
adb pull /data/data/com.mypackage/databases/store.db
```

OR

Run remote shell

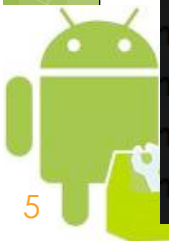
```
$ adb -s emulator-5554 shell
$ cd /data/data/com.yourpackage/databases
$ sqlite3 your-db-file.db
> .help
```

adb -s **<serialNumber>** **<command>** to access a device



The screenshot shows a terminal window titled "Terminale — adb — 80x24". The user is at a Mac (MarcoRonchetti-MacBook500) and navigates to the Android SDK platform-tools directory. They then run the command `./adb -s emulator-5554 shell` to get a remote shell on the emulator. From there, they run `pwd` to confirm the location is `/`, then `sqlite3` to start the SQLite command-line interface. The SQLite version is 3.7.4. They enter `.help` to see instructions, and finally `.exit` to leave the SQLite shell. The terminal output is as follows:

```
# MarcoRonchetti-MacBook500:platform-tools ronchet$ cd /Applications/Utilities/Android\ Studio.app/Contents/Resources/defaults/Android Studio 2.2/sdk-macosx/platform-tools
MarcoRonchetti-MacBook500:platform-tools ronchet$ ./adb -s emulator-5554 shell
# pwd
/
# sqlite3
SQLite version 3.7.4
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .exit
#
```



# adb

adb is in your **android-sdk/platform-tools** directory

It allows you to:

- Run shell commands on an emulator or device
- Copy files to/from an emulator or device
- Manage the state of an emulator or device
- Manage port forwarding on an emulator or device

It is a client-server program that includes three components:

- A **client**, which runs on your development machine.
- A **daemon**, which runs as a background process on each emulator or device instance.
- A **server**, which runs as a background process on your development machine and manages communication between the client and the daemon.

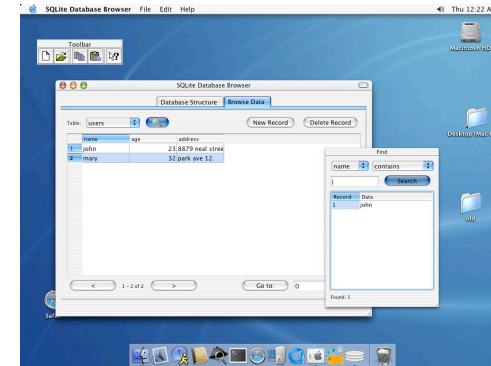
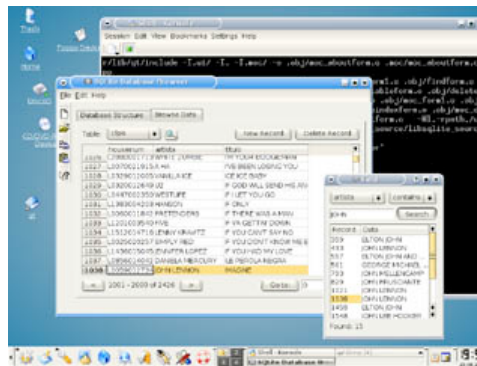
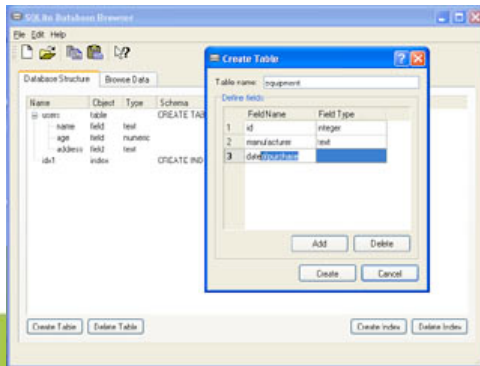
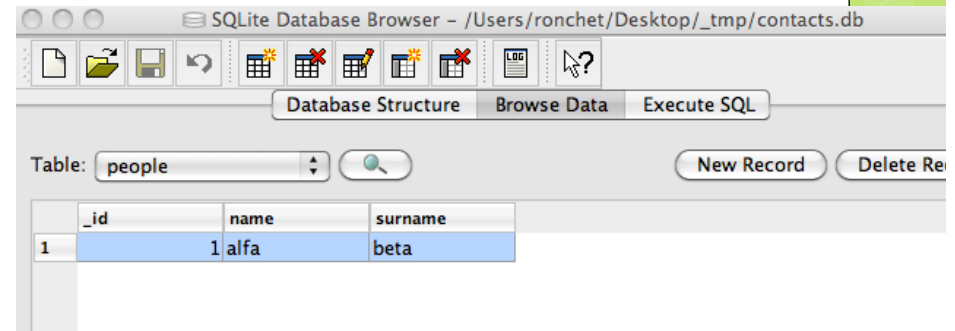
See <http://developer.android.com/guide/developing/tools/adb.html>



# A graphical sqlite browser

<http://sqlitebrowser.sourceforge.net/index.html>

- Create and compact database files
- Create, define, modify and delete tables
- Create, define and delete indexes
- Browse, edit, add and delete records
- Search records
- Import and export records as text
- Import and export tables from/to CSV files
- Import and export databases from/to SQL dump files
- Issue SQL queries and inspect the results
- Examine a log of all SQL commands issued by the application





# Testing and deploying on your device

Marco Ronchetti  
Università degli Studi di Trento

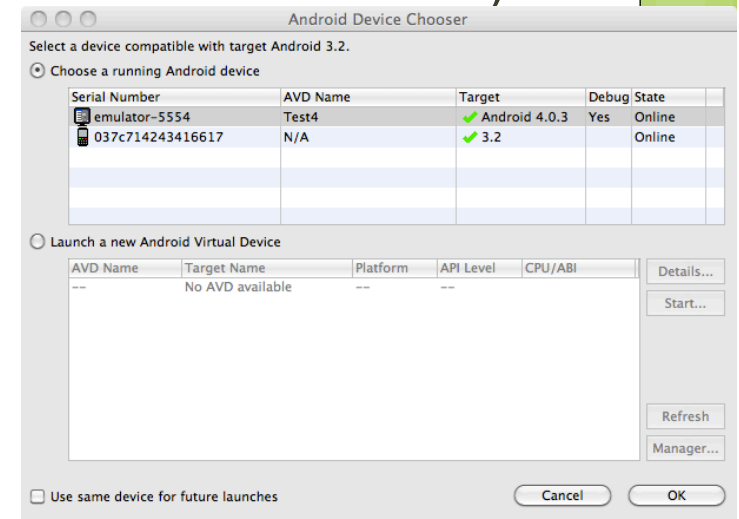
---



# Configure device

- 1) **Turn on "USB Debugging"** on your device.  
On the device, go in
  - Android <4: **Settings > Applications > Development**
  - Android >=4: **Settings > Developer options**and enable **USB debugging**
- 2) **Load driver on PC** (win-linux, on Mac not needed)
- 3) Check in shell: **adb devices**
- 4) In Eclipse, you'll have the choice

Make sure the version of OS is correct both in project properties  
And in manifest!



See <http://developer.android.com/guide/developing/device.html>



BY

# Alternative, simple way to deploy

e.g. to give your app to your friends

Get Dropbox both on PC and Android device

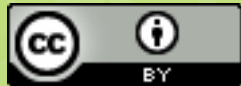
Copy your apk from bin/res into dropbox (on PC)

Open dropbox on Android device, and open your apk

By sharing your dropbox with others you can easily pass your app.

[www.dropbox.com](http://www.dropbox.com)





# DAO Implementation File System

Marco Ronchetti  
Università degli Studi di Trento

---

# The Java-IO philosophy

## 1) Get a (raw) source

```
File f; ... ; InputStream s = new FileInputStream(f);  
Socket s; ... ; InputStream s=s.getInputStream();  
StringBuffer b; ... ; InputStream s = new StringBufferInputStream(f);
```

## 2) Add functionality

```
Reader r=new InputStringReader(s); //bridge class  
DataInputString dis=new DataInputString(s); //primitive data  
ObjectInputString ois=new ObjectInputString(s); //serialized objects
```

## 3) Compose multiple functionalities

```
InputStream es=new FilteredInputStream(  
    new BufferedInputStream(  
        new PushBackInputStream(s)));
```



# Choose the type of source!

You can choose among four types of basic sources:

	BYTE		CHARACTER	
SOURCE	InputStream	OutputStream	Reader	Writer

Both file and directory information is available via the File class, or the classes (like Path) in the nio package.



# I/O Table

	Byte Based		Character Based	
	<i>Input</i>	<i>Output</i>	<i>Input</i>	<i>Output</i>
<b>Basic</b>	InputStream	OutputStream	Reader InputStreamReader	Writer OutputStreamWriter
<b>Arrays</b>	ByteArrayInputStream	ByteArrayOutputStream	CharArrayReader	CharArrayWriter
<b>Files</b>	FileInputStream	FileOutputStream	FileReader	FileWriter
	RandomAccessFile	RandomAccessFile		
<b>Pipes</b>	PipedInputStream	PipedOutputStream	PipedReader	PipedWriter
<b>Buffering</b>	BufferedInputStream	BufferedOutputStream	BufferedReader	BufferedWriter
<b>Filtering</b>	FilterInputStream	FilterOutputStream	FilterReader	FilterWriter
<b>Parsing</b>	PushbackInputStream		PushbackReader	
	StreamTokenizer		LineNumberReader	
<b>Strings</b>			StringReader	StringWriter
<b>Data</b>	DataInputStream	DataOutputStream		
<b>Data - Formatted</b>	PrintStream		PrintWriter	
<b>Objects</b>	ObjectInputStream	ObjectOutputStream		
<b>Utilities</b>	SequenceInputStream			



# Android internal file I/O

```
String FILENAME = "hello_file";  
String string = "hello world!";
```

```
FileOutputStream fos = openFileOutput(FILENAME,  
                        Context.MODE_PRIVATE); // called in a Context
```

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



# Using temporary files

```
File file = new File(getCacheDir(), "temp.txt");
try {
    file.createNewFile();
    FileWriter fw = new FileWriter(file);
    BufferedWriter bw = new BufferedWriter(fw);
    bw.write("Hello World\n");
    bw.close();
} catch (IOException e) {
    Toast.makeText(this,
        "Error creating a file!"
        ,Toast.LENGTH_SHORT).show();
}
```

When the device is low on internal storage space, Android may delete these cache files to recover space.

You should not rely on the system to clean up these files for you.

Clean the cache files yourself

stay within a reasonable limit of space consumed, such as 1MB.





# Other useful methods

## `getFilesDir()`

Get the absolute path where internal files are saved.

## `getDir()`

Creates (or opens an existing) directory within your internal storage space.

## `deleteFile()`

Deletes a file saved on the internal storage.

## `fileList()`

Returns an array of files currently saved by your application.



# The DAO interface

```
package it.unitn.science.latemar;  
  
import java.util.List;  
  
public interface PersonDAO {  
    public void open();  
    public void close();  
  
    public Person insertPerson(Person person) ;  
    public void deletePerson(Person person) ;  
    public List<Person> getAllPerson() ;  
}
```



```
package it.unitn.science.latemar;  
import ...
```

# The DAO implementation - FS

```
public class PersonDAO_FS_impl implements PersonDAO {  
    DataOutputStream fos;  
    DataInputStream fis;  
    Context context=MyApplication.getAppContext();  
    final String FILENAME="contacts";  
  
    @Override  
    public void open() {  
        try {  
            fos=new DataOutputStream(  
                context.openFileOutput(FILENAME, Context.MODE_APPEND)  
            );  
        } catch (FileNotFoundException e) {e.printStackTrace();}  
    }  
  
    @Override  
    public void close() {  
        try {  
            fos.close();  
        } catch (IOException e) {e.printStackTrace();}  
    }  
}
```

This should  
never happen



# The DAO impl. – data access 2

@Override

```
public Person insertPerson(Person person) {  
    try {  
        fos.writeUTF(person.getName());  
        fos.writeUTF(person.getSurname());  
    } catch (IOException e) { e.printStackTrace(); }  
    return person;  
}
```

write as  
Unicode

@Override

```
public void deletePerson(Person person) {  
    Log.d("trace", "deletePerson DAO_FS – UNIMPLEMENTED!");  
}
```



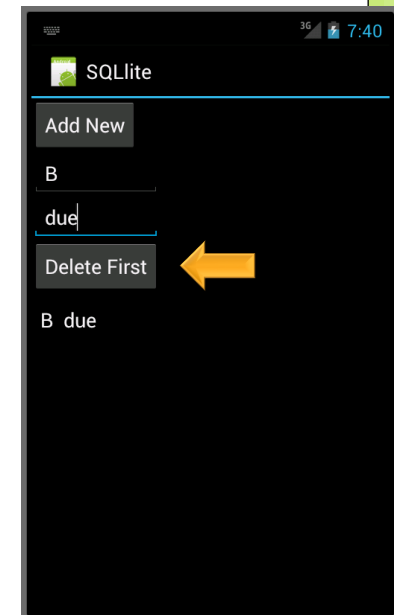
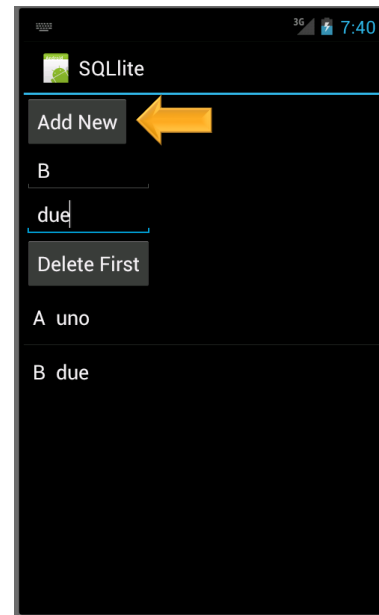
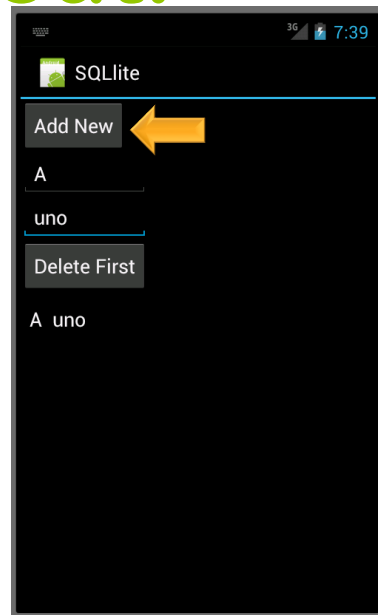
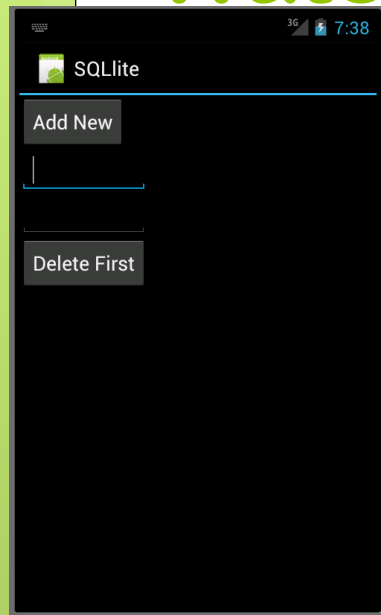
# The DAO impl. – data access 3

@Override

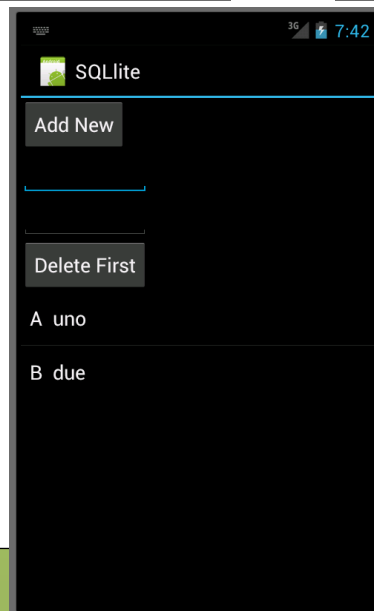
```
public List<Person> getAllPersons() {  
    List<Person> list=new ArrayList<Person>();  
    try { fis=new DataInputStream( context.openFileInput(FILENAME) );  
    } catch (FileNotFoundException e) {  
        e.printStackTrace(); return list;  
    }  
    while (true) {  
        try {  
            String name=fis.readUTF();  
            String surname=fis.readUTF();  
            Person p=new Person(name, surname);  
            list.add(p);  
        } catch (EOFException e) { break;  
        } catch (IOException e) { e.printStackTrace(); break; }  
    }  
    try { fis.close(); } catch (IOException e) { e.printStackTrace(); }  
    return list;  
}
```



# Watch out!



Restart...

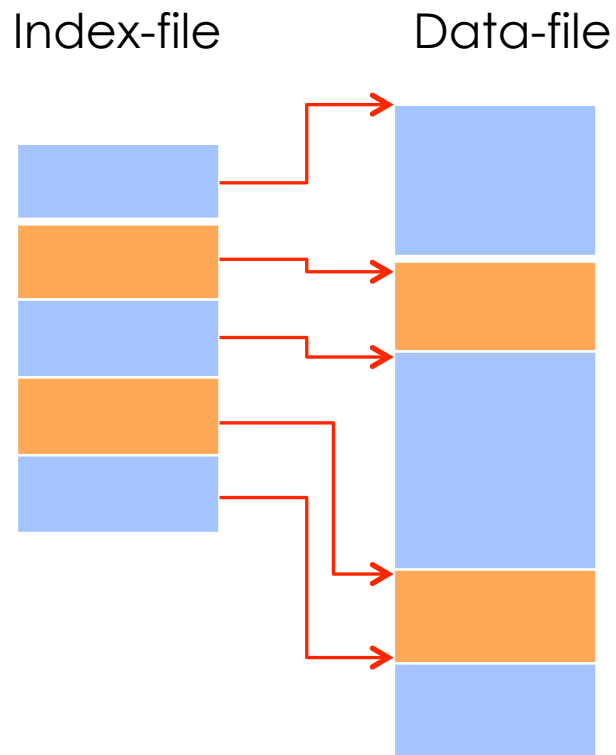


## Why so?



## Serializing any-size objects to a random access file

<http://blog.donkersautomatisering.nl/2011/06/29/serializing-arbitrarily-sized-objects-to-a-random-access-file/>



See `java.io`  
**Class `RandomAccessFile`**





# External Files

Marco Ronchetti  
Università degli Studi di Trento

---



# External storage

Every Android-compatible device supports a shared "external storage" that you can use to save files.

It can be:

- a **removable storage media** (such as an SD card)
- an **internal (non-removable)** storage.

Files saved to the external storage

- are **world-readable**
- **can be modified** by the user when the USB card storage is moved **on a computer!**



# Possible states of external media

`String Environment.getExternalStorageState();`

## `MEDIA_MOUNTED`

- media is present and mounted at its mount point with read/write access.

## `MEDIA_MOUNTED_READ_ONLY`

- media is present and mounted at its mount point with read only access.

## `MEDIA_NOFS`

- media is present but is blank or is using an unsupported filesystem

## `MEDIA_CHECKING`

- media is present and being disk-checked

## `MEDIA_UNMOUNTED`

- media is present but not mounted

## `MEDIA_SHARED`

- media is in SD card slot, unmounted, and shared as a mass storage device.

## `MEDIA_UNMOUNTABLE`

- media is present but cannot be mounted.

## `MEDIA_REMOVED`

- media is not present.

`boolean Environment.isExternalStorageEmulated()`  
`boolean Environment.isExternalStorageRemovable()`

## `MEDIA_BAD_REMOVAL`

- media was removed before it was unmounted.



# Standard directories (constants):

## DIRECTORY\_DOWNLOADS

- files that have been downloaded by the user.

## DIRECTORY\_MOVIES

- movies that are available to the user.

## DIRECTORY\_PICTURES

- pictures that are available to the user.

## DIRECTORY\_DCIM

- The traditional location for pictures and videos when mounting the device as a camera.

Places for audio files:

- **DIRECTORY\_MUSIC**

- music for the user.

- **DIRECTORY\_ALARMS**

- alarms sounds that the user can select (not as regular music).

- **DIRECTORY\_NOTIFICATIONS**

- notifications sounds that the user can select (not as regular music).

- **DIRECTORY\_PODCASTS**

- podcasts that the user can select (not as regular music).

- **DIRECTORY\_RINGTONES**

- ringtones that the user can select (not as regular music).



# Other Environment static methods

static File **getRootDirectory()**

- Gets the Android root directory (typically returns /system).

static File **getDataDirectory()**

- Gets the Android data directory (typically returns /data).

static File **getDownloadCacheDirectory()**

- Gets the Android Download/Cache content directory. Here go **temporary** files that **are specific to your application**. If the user uninstalls your application, this directory and all its contents will be deleted. You should manage these cache files and remove those that aren't needed in order to preserve file space.

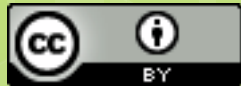
static File **getExternalStorageDirectory()**

- Gets the Android external storage directory. Here go files that **are specific to your application**. If the user uninstalls your application, this directory and all its contents will be deleted.

static File **getExternalStoragePublicDirectory**(String type)

- Get a top-level public external storage directory for placing files of a particular type. This is where the user will typically place and manage their own files. Here go **files that are not specific to your application** and that should *not* be deleted when your application is uninstalled





# Rooting a device

Marco Ronchetti  
Università degli Studi di Trento

---

# Rooting

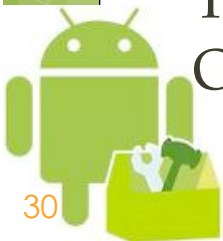
The process of allowing users of Android devices to get root access. Varies widely by device, as it usually exploits a security weakness in the firmware shipped from the factory.

Goal:

- to overcome limitations imposed by that carriers and hardware manufacturers
- to alter or replace system applications and settings
- to run specialized apps that require administrator-level permissions
- to perform other operations that are otherwise inaccessible to a normal Android user.

The process of rooting

On the iphone: **jailbreaking**



## e.g.: CyanogenMod

a replacement firmware. Offers several features, like:

- an OpenVPN client,
- a reboot menu,
- support for Wi-Fi, Bluetooth, and USB tethering,
- CPU overclocking and performance enhancements, app permissions management

Over 1.5 M installations



# Is it legal?

On July 26, 2010, the U.S. Copyright office announced a new exemption making it **officially legal to root a device and run unauthorized third-party applications**, as well as the ability to unlock any cell phone for use on multiple carriers.





# Industry reaction

- concern about improper functioning of devices running unofficial software and related support costs.
- offers features for which carriers would otherwise charge a premium

Technical obstacles have been introduced in many devices (e.g. locked bootloaders).

In 2011 an increasing number of devices shipped with unlocked or unlockable bootloaders.



# The HTC case

“HTC is committed to listening to users and delivering customer satisfaction. We have heard your voice and starting now, we will allow our bootloader to be unlocked for 2011 models going forward.

*It is our responsibility to caution you **that not all claims** resulting or caused by or from the unlocking of the bootloader **may be covered under warranty**.*

*We strongly suggest that you do not unlock the bootloader unless you are confident that you understand the risks involved.”*

