Android Development

Shared Preference, JSON

Android Files

Persistence is a strategy that allows the reusing of volatile objects and other data items by storing them Into a permanent storage system such as disk files and databases.

File IO management in Android includes –among others- the familiar IO Java classes: Streams, Scanner, PrintWriter, and so on.

Permanent files can be stored *internally* in the device's main memory (usually small, but not volatile) or *externally* in the much larger SD card.

Files stored in the device's memory, share space with other application's resources such as code, icons, pictures, music, etc.

Internal files are called: **Resource Files** or **Embedded Files.**

Choosing a Persistent Environment

Your permanent data storage destination is usually determined by parameters such as:

- size (small/large),
- location (internal/external),
- accessibility (private/public).

Depending of your situation the following options are available:

1.Shared Preferences Store private primitive data in *key-value* pairs.
2.Internal Storage Store private data on the device's main memory.
3.External Storage Store public data on the shared external storage.
4.SQLite Databases Store structured data in a private/public database.
5. Network Connection Store data on the web.

Preferences

- **Preferences** is an Android lightweight mechanism to store and retrieve *key-value* pairs of primitive data types (also called *Maps*, and *Associative Arrays.*
- Typically used to keep state information and shared data among several activities of an application.
- On each entry *<key-value>* the key is a string and the value must be a primitive data type.
- Preferences are similar to Bundles however they are persistent while Bundles are not.

Preferences

- **Using Preferences API calls**
- You have three API choices to pick a Preference:

1. **getPreferences()** from within your Activity, to access activity specific preferences
2. **getSharedPreferences()** from within your Activity to access application-level preferences
3. **getDefaultSharedPreferences()**, on *PreferencesManager*, to get the shared preferences that work in concert with Android's overall preference framework

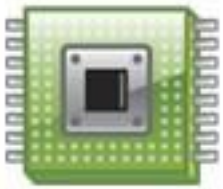
Shared Preferences

SharedPreferences files are good for handling a handful of Items. Data in this type of container is saved as **<Key, Value>** pairs where the *key* is a string and its associated *value* must be a primitive data type.

KEY	VALUE

This class is functionally similar to Java Maps, however; unlike e *permanent*.

Data is stored in the device's internal main memory.



PREFERENCES are typically used to keep state information and sha among several activities of an application.

- acct
- cache
- charger
- config
- d
- data
  - adb
  - app
  - app-asec
  - app-lib
  - app-private
  - backup
  - bootchart
  - bugreports
  - dalvik-cache
  - data
    - AndrodApp1.AndrodApp1
    - Mono.Android.DebugRuntime
    - Mono.Android.Platform.ApiLevel_27
    - MyMovieApp.MyMovieApp
      - cache
      - code_cache
      - files
      - lib
      - shared_prefs
        - MyMovieApp.MyMovieApp_preferences.xml

File Explorer

Name

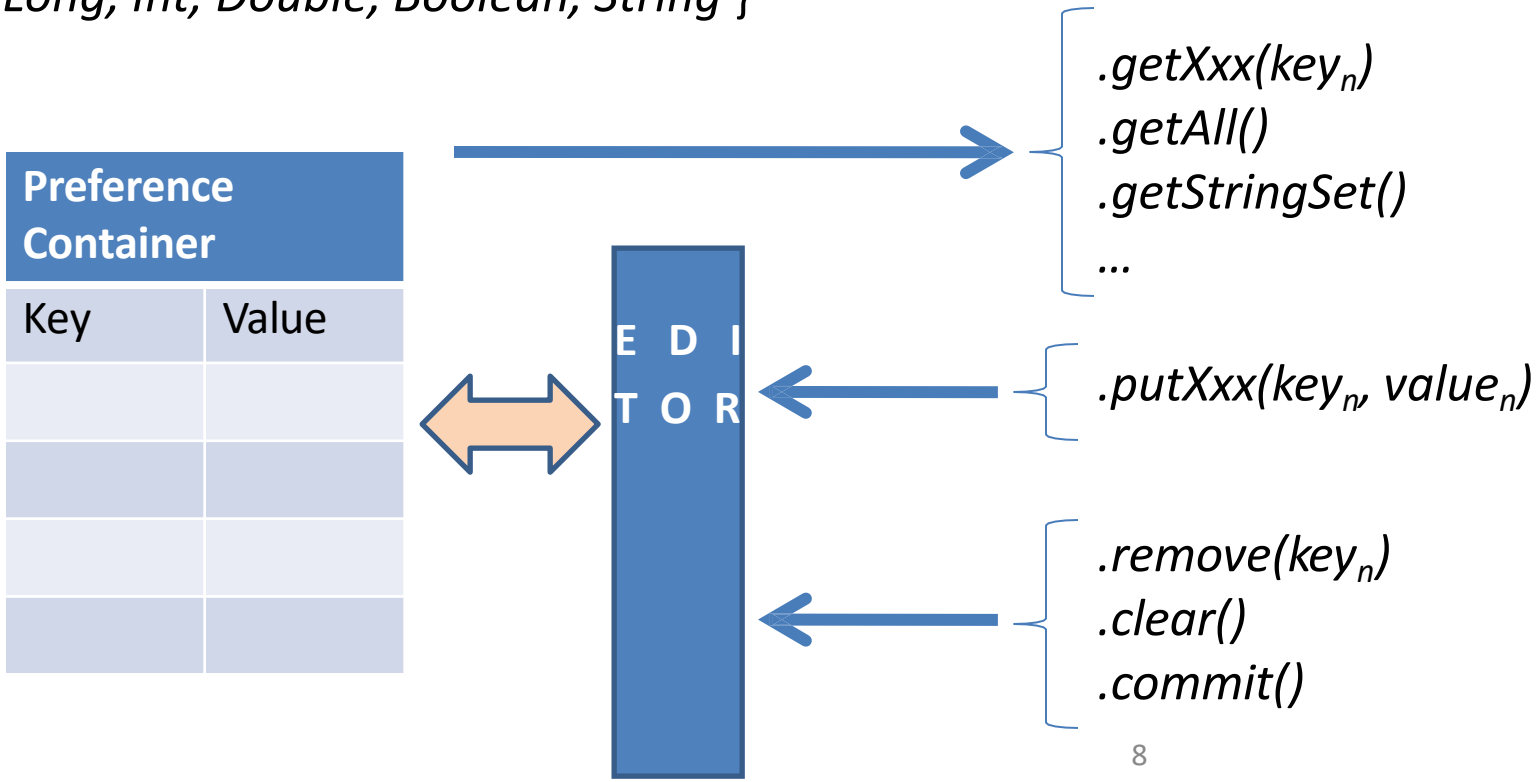
- com.android.mms
- com.android.music
- com.android.netspeed
- com.android.packageinstaller
- com.android.phone
- com.android.preferenceDemo
  - lib
  - shared_prefs
    - com.android.preferenceDemo_p
    - my_pref.xml
    - pref_2.xml

Files & Preferences

Using Preferences API calls

Each of the Preference mutator methods carries a typed-value content that can be manipulated by an *editor* that allows *putXxx… and getXxx…* commands to place data in and out of the Preference container.

Xxx = { *Long, Int, Double, Boolean, String* }



Preferences

- **Example**

1. In this example a persistent *SharedPreferences* object is created at the end of an activity lifecycle. It contains data (name, phone, credit, etc. of a fictional customer)
2. The process is interrupted using the "*Back Button*" and re-executed later.
3. Just before been killed, the state of the running application is saved in the designated *Preference* object.
4. When re-executed, it finds the saved *Preference* and uses its persistent data.

Preferences

- **Example2:** Saving/Retrieving a SharedPreference Object

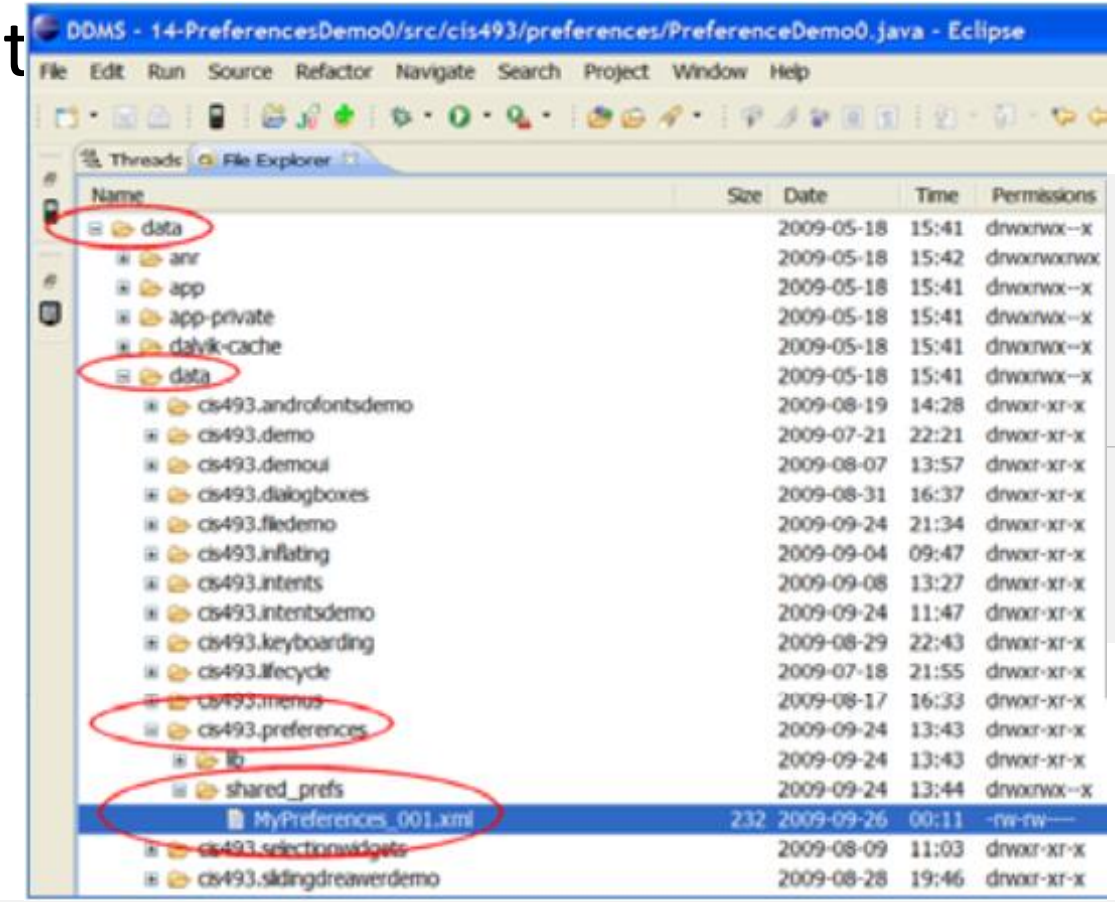


Image of the preference file (obtained by pulling a copy of the file out of the device).

Using DDMS to explore the Device's memory map. Observe the choices made by the user are saved in the *data/data/Shared_prefs/* folder as an XML file.

SharedPreferences

- SharedPreferences sharedpreferences = getSharedPreferences(MyPREFERENCES, Context.MODE_PRIVATE);
- **MODE_APPEND:** This will append the new preferences with the already existing preferences
- **MODE_ENABLE_WRITE_AHEAD_LOGGING:** Database open flag. When it is set , it would enable write ahead logging by default
- **MODE_PRIVATE:** By setting this mode, the file can only be accessed using calling application
- **MODE_WORLD_READABLE:** This mode allow other application to read the preferences
- **MODE_WORLD_WRITEABLE:** This mode allow other application to write the preferences

SharedPreferences

- Editor editor = sharedpreferences.edit();
- editor.putString("key", "value");
- editor.commit();
- **apply():** It is an abstract method. It will commit your changes back from editor to the sharedPreference object you are calling
- **clear():** It will remove all values from the editor
- **remove(String key):** It will remove the value whose key has been passed as a parameter
- **putLong(String key, long value):** It will save a long value in a preference editor
- **putInt(String key, int value):** It will save a integer value in a preference editor
- **putFloat(String key, float value):** It will save a float value in a preference editor
- **contains(String key):** This method is used to check whether the preferences contains a preference.
- **getAll():** This method is used to retrieve all values from the preferences.
- **edit():** This method is used to create a new Editor for these preferences, through which you can make modifications to the data in the preferences and atomically commit those changes back to the SharedPreferences object.

SharedPreferences

- **getBoolean(String key, boolean defValue):** This method is used to retrieve a boolean value from the preferences.
- **getFloat(String key, float defValue):** This method is used to retrieve a float value from the preferences.
- **getInt(String key, int defValue):** This method is used to retrieve an int value from the preferences.
- **getLong(String key, long defValue):** This method is used to retrieve a long value from the preferences.
- **getString(String key, String defValue):** This method is used to retrieve a String value from the preferences.
- **getStringSet(String key, Set defValues):** This method is used to retrieve a set of String values from the preferences.

Storing Data

```
val pref = applicationContext.getSharedPreferences("MyPref", 0) // 0 - for private mode

val editor: Editor = pref.edit()
editor.putBoolean("key_name", true) // Storing boolean - true/false

editor.putString("key_name", "string value") // Storing string

editor.putInt("key_name", "int value") // Storing integer

editor.putFloat("key_name", "float value") // Storing float

editor.putLong("key_name", "long value") // Storing long

editor.commit() // commit changes
```

Retrieving Data

```
val pref = applicationContext.getSharedPreferences("MyPref", 0) // 0 - for private mode

pref.getString("key_name", null) // getting String

pref.getInt("key_name", -1) // getting Integer

pref.getFloat("key_name", null) // getting Float

pref.getLong("key_name", null) // getting Long

pref.getBoolean("key_name", null) // getting boolean
```

Clearing or Deleting Data

- editor.remove("name"); // will delete key name
- editor.remove("email"); // will delete key email
- editor.commit(); // commit changes

- editor.clear();
- editor.commit(); // commit changes

What is JSON?

- **What is JSON?**
- JSON is used for data interchange (posting and retrieving) from the server.
- JSON is the best alternative for XML and its more readable by human.
- A JSON response from the server consists of many fields.

What is JSON?

- An example JSON response/data is given below.

```
{
  "title":"JSONParserTutorial",
  "array":[
    {
    "company":"Google"
    },
    {
      "company":"Facebook"
    },
    {
    "company":"LinkedIn"
    },
    {
      "company" : "Microsoft"
    },
    {
      "company": "Apple"
    }
    ],
    "nested":{
    "flag": true,
    "random_number":1
    }
}
```

- We’ve create a random JSON data string from https://www.jsoneditoronline.org/ page.
- It’s handy for editing JSON data.

JSON data

- JSON data consists of 4 major components that are listed below:
  1. **Array**: A **JSONArray** is enclosed in square brackets ([). It contains a set of objects
  2. **Object**: Data enclosed in curly brackets ({) is a single JSONObject. Nested JSONObjects are possible and are very commonly used
  3. **Keys**: Every JSONObject has a key string that's contains certain value
  4. **Value**: Every key has a single value that can be of any type string, double, integer, boolean etc

Android JSONObject

- Create a JSONObject from the static JSON data string given above and display the JSONArray in a ListView.

JSON - Parsing

- For parsing a JSON object, we will create an object of class JSONObject and specify a string containing JSON data to it. Its syntax is

```
var in1: String = """
val reader = JSONObject(in1)
```

```
{
   "sys":
   {
      "country":"GB",
      "sunrise":1381107633,
      "sunset":1381149604
   },
   "weather":[
      {
         "id":711,
         "main":"Smoke",
         "description":"smoke",
         "icon":"50n"
      }
   ],

  "main":
   {
      "temp":304.15,
      "pressure":1009,
   }
}
```

JSON-Parsing

- A JSON file consist of different object with different key/value pair e.t.c. So JSONObject has a separate function for parsing each of the component of JSON file. Its syntax is given below –

JSON-Parsing

```
val sys: JSONObject = reader.getJSONObject("sys")
country = sys.getString("country")
val main: JSONObject = reader.getJSONObject("main")
temperature = main.getString("temp")
```

JSON-Parsing

- The method **getJSONObject** returns the JSON object.
- The method **getString** returns the string value of the specified key.

JSON-Parsing

- **get(String name)**
- This method just Returns the value but in the form of Object type
- **getBoolean(String name)**
- This method returns the boolean value specified by the key
- **getDouble(String name)**
- This method returns the double value specified by the key

JSON-Parsing

- **getInt(String name)**
- This method returns the integer value specified by the key
- **getLong(String name)**
- This method returns the long value specified by the key

JSON-Parsing

- **length()**
- This method returns the number of name/value mappings in this object..
- **names()**
- This method returns an array containing the string names in this object.

```
{
    "contacts": [
        {
                "id": "c200",
                "name": "Ravi Tamada",
                "email": "ravi@gmail.com",
                "address": "xx-xx-xxxx,x - street, x - country",
                "gender" : "male",
                "phone": {
                    "mobile": "+91 0000000000",
                    "home": "00 000000",
                    "office": "00 000000"
                }
        },
        {
                "id": "c201",
                "name": "Johnny Depp",
                "email": "johnny_depp@gmail.com",
                "address": "xx-xx-xxxx,x - street, x - country",
                "gender" : "male",
                "phone": {
                    "mobile": "+91 0000000000",
                    "home": "00 000000",
                    "office": "00 000000"
                }
        },
```

Questions