Android Shared Preferences Overview

Shared Preferences allows activities and applications to keep preferences, in the form of key-value pairs similar to a Map that will persist even when the user closes the application. Android stores Shared Preferences settings as XML file in shared_prefs folder under DATA/data/{application package} directory. The DATA folder can be obtained by calling Environment.getDataDirectory(). SharedPreferences is application specific, i.e. the data is lost on performing one of the following options:

- on uninstalling the application
- on clearing the application data (through Settings)

As the name suggests, the primary purpose is to store user-specified configuration details, such as user specific settings, keeping the user logged into the application. To get access to the preferences, we have three APIs to choose from:

- getPreferences() : used from within your Activity, to access activity-specific preferences
- getSharedPreferences() : used from within your Activity (or other application Context), to access application-level preferences
- getDefaultSharedPreferences() : used on the PreferenceManager, to get the shared preferences that work in concert with Android's overall preference framework

In this tutorial we'll go with getSharedPreferences(). The method is defined as follows: getSharedPreferences (String PREFS_NAME, int mode) PREFS_NAME is the name of the file. mode is the operating mode. Following are the operating modes applicable:

- MODE_PRIVATE: the default mode, where the created file can only be accessed by the calling application
- MODE_WORLD_READABLE: Creating world-readable files is very dangerous, and likely to cause security holes in applications
- MODE_WORLD_WRITEABLE: Creating world-writable files is very dangerous, and likely to cause security holes in applications
- MODE_MULTI_PROCESS: This method will check for modification of preferences even if the Shared Preference instance has already been loaded
- 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

Initialization

We need an editor to edit and save the changes in shared preferences. The following code can be used to get the shared preferences.

```
SharedPreferences pref =
getApplicationContext().getSharedPreferences("MyPref", 0); // 0 - for private
mode
Editor editor = pref.edit();
```

Storing Data

editor.commit() is used in order to save changes to shared preferences.

```
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

Data can be retrieved from saved preferences by calling getString() as follows:

```
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

remove(“key_name”) is used to delete that particular value. clear() is used to remove all 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
```