Lesson 12

Persistence:
Files & Preferences

Victor Matos
Cleveland State University

Files & Preferences

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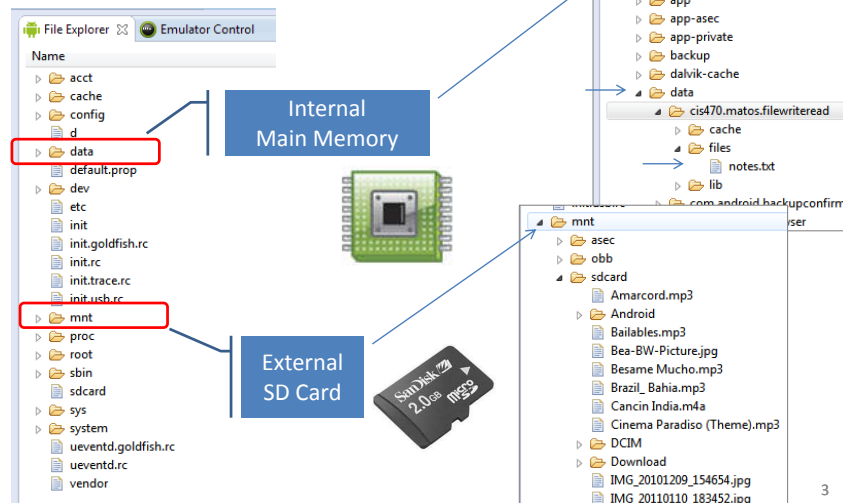
Internal files are called: **Resource Files** or **Embedded Files.**

2

Files & Preferences

Exploring Android's File System

Use the emulator's **File Explorer** to see and manage your device's storage structure.

3

Files & Preferences

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.

4

Files & 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.

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

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

PREFERENCES are typically used to keep state information and shared data among several activities of an application.
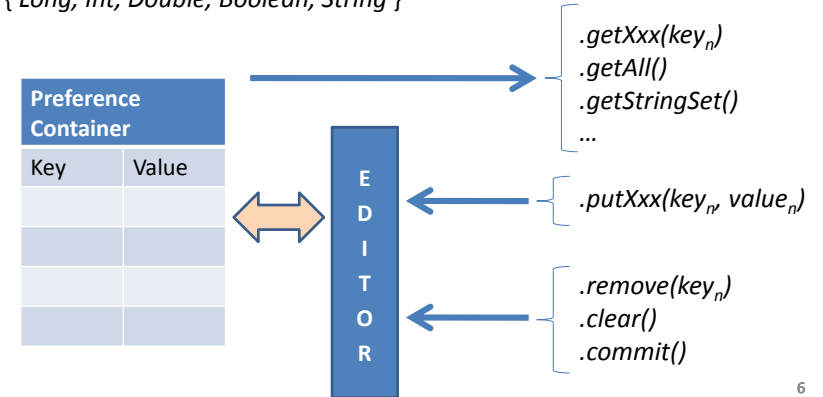
KEY	VALUE

5

Files & Preferences

Shared 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* }

Preference Container	
Key	Value

E D I T O R

.getXxx(key$_n$)
.getAll()
.getStringSet()
...

.putXxx(key$_n$, value$_n$)

.remove(key$_n$)
.clear()
.commit()

6

Files & Preferences

Example. Shared Preferences

In this example the user selects a preferred 'color' and 'number'. Both values are stored in a SharedPreferences file.

Key	Value
chosenColor	RED
chosenNumber	7

```
private void usingPreferences(){
    // Save data in a SharedPreferences container
    // We need an Editor object to make preference changes.

❶  SharedPreferences myPrefs = getSharedPreferences("my_preferred_choices",
                                          Activity.MODE_PRIVATE);
    SharedPreferences.Editor editor = myPrefs.edit();
❷         editor.putString("chosenColor", "RED");
           editor.putInt("chosenNumber", 7 );
    editor.commit();

    // retrieving data from SharedPreferences container (apply default if needed)
❸  String favoriteColor = myPrefs.getString("chosenColor", "BLACK");
    int favoriteNumber = myPrefs.getInt("chosenNumber", 11 );

}
```

7

Files & Preferences

Shared Preferences. Example - Comments

1. The method getSharedPreferences(…) creates (or retrieves) a table called *my_preferred_choices* file, using the default *MODE_PRIVATE* access. Under this access mode only the calling application can operate on the file.

2. A SharedPreferences editor is needed to make any changes on the file. For instance editor.putString("chosenColor", "RED") creates(or updates) the key "chosenColor" and assigns to it the value "RED". All editing actions must be explicitly committed for the file to be updated.

3. The method **getXXX(…)** is used to extract a value for a given key. If no key exists for the supplied name, the method uses the designated default value. For instance myPrefs.getString("chosenColor", "BLACK") looks into the file *myPrefs* for the key "chosenColor" to returns its value, however if the key is not found it returns the default value "BLACK".
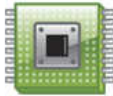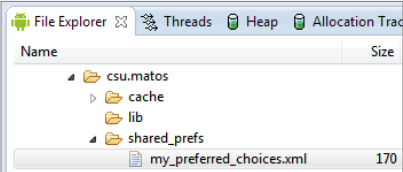
8

Files & Preferences

Shared Preferences. Example - Comments

SharedPreference containers are saved as XML files in the application's internal memory space. The path to a preference files is
/data/data/packageName/shared_prefs/filename.

For instance in this example we have:

If you pull the file from the device, you will see the following

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
- <map>
      <string name="favorite_color">#ff0000ff</string>
      <int name="favorite_number" value="101"/>
  </map>
```
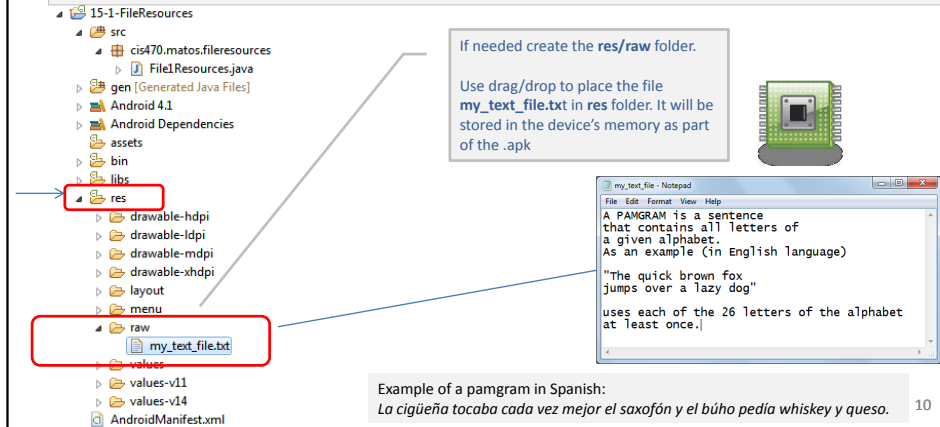

Files & Preferences

Internal Storage. Reading an Internal Resource File

An Android application may include resource elements such as those in:
res/drawable , res/raw, res/menu, res/style, etc.
Resources could be accessed through the **.getResources(…)** method. The method's argument is the ID assigned by Android to the element in the R resource file. For example:

```
InputStream is = this.getResources()
                      .openRawResource(R.raw.my_text_file);
```


If needed create the **res/raw** folder.

Use drag/drop to place the file **my_text_file.txt** in **res** folder. It will be stored in the device's memory as part of the .apk

A PAMGRAM is a sentence
that contains all letters of
a given alphabet.
As an example (in English language)

"The quick brown fox
jumps over a lazy dog"

uses each of the 26 letters of the alphabet
at least once.

Example of a pamgram in Spanish:
La cigüeña tocaba cada vez mejor el saxofón y el búho pedía whiskey y queso.
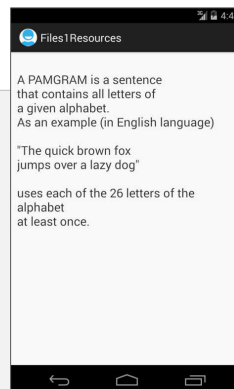

Files & Preferences

Example 1. Reading an Internal Resource File

This app stores a text file in its RESOURCE (**res/raw**) folder.
The embedded raw data (containing a *pamgram*) is read and displayed in a text box (see previous image)

```java
//reading an embedded RAW data file
public class File1Resources extends Activity {
   TextView txtMsg;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      txtMsg = (TextView) findViewById(R.id.textView1);
      try {
         PlayWithRawFiles();

      } catch (IOException e) {
         txtMsg.setText( "Problems: " + e.getMessage() );
      }
   }// onCreate
```

A PAMGRAM is a sentence
that contains all letters of
a given alphabet.
As an example (in English language)

"The quick brown fox
jumps over a lazy dog"

uses each of the 26 letters of the
alphabet
at least once.

Files & Preferences

Example 1. Reading an Internal Resource File

Reading an embedded file containing lines of text.

```java
public void PlayWithRawFiles() throws IOException {
   String str="";
   StringBuffer buf = new StringBuffer();

① int fileResourceId = R.raw.my_text_file;
   InputStream is = this.getResources().openRawResource(fileResourceId);
② BufferedReader reader = new BufferedReader(new
                              InputStreamReader(is) );

   if (is!=null) {
      while ((str = reader.readLine()) != null) {
③       buf.append(str + "\n" );
         }
   }
   reader.close();
   is.close();
   txtMsg.setText( buf.toString() );

 }// PlayWithRawFiles
} // File1Resources
```

Files & Preferences

Example1 - Comments

1. A **raw file** is an arbitrary dataset stored in its original raw format (such as .docx, pdf, gif, jpeg, etc). Raw files can be accessed through an *InputStream* acting on a *R.raw.filename* resource entity.
   CAUTION: *Android requires resource file names to be in lowercase form.*

2. The expression `getResources().openRawResource(fileResourceId)` creates an InputStream object that sends the bytes from the selected resource file to an input buffer. If the resource file is not found it raises a *NotFoundException* condition.

3. A *BufferedReader* object is responsible for extracting lines from the input buffer and assembling a string which finally will be shown to the user in a textbox. Protocol expects that conventional IO housekeeping operations should be issued to close the reader and stream objects.
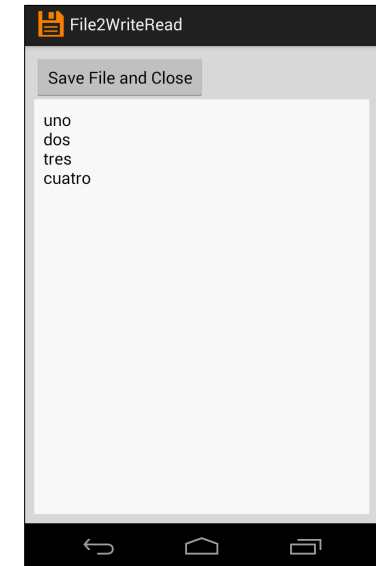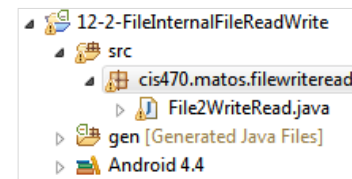

Files & Preferences

Example 2. Reading /Writing an Internal Resource File

In this example an application exposes a GUI on which the user enters a few lines of data. The app collects the input lines and **writes** them to a persistent **internal data file.**

Next time the application is executed the *Resource File* will be **read** and its data will be shown on the UI.

Files & Preferences

Example 2. Reading /Writing an Internal Resource File

The *internal resource file* (notes.txt) is private and cannot be seen by other apps residing in main memory.

In our example the files **notes.txt** is stored in the phone's internal memory under the name:
/data/data/cis470.matos.fileresources/files/notes.txt
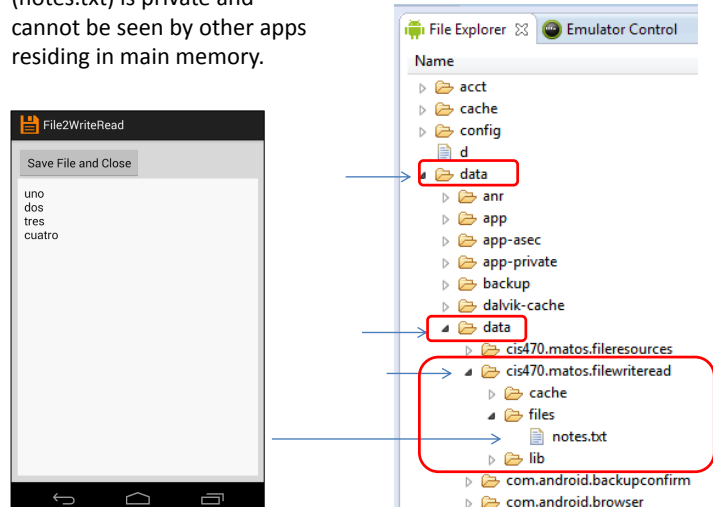
Files & Preferences

Example 2. Reading /Writing an Internal Resource File

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#ffdddddd"
    android:padding="10dp"
    android:orientation="vertical" >

    <Button android:id="@+id/btnFinish"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
         android:padding="10dp"
        android:text=" Save File and Close " />

    <EditText
        android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="10dp"
        android:background="#ffffffff"
        android:gravity="top"
        android:hint="Enter some lines of data here..."  />

</LinearLayout>
```
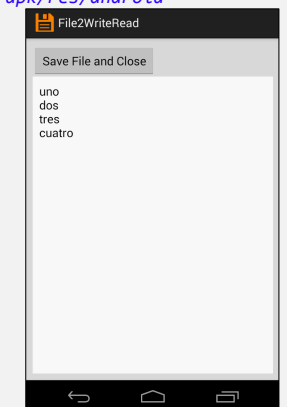
Files & Preferences

```java
public class File2WriteRead extends Activity {

   private final static String FILE_NAME = "notes.txt";
   private EditText txtMsg;

   @Override
   public void onCreate(Bundle icicle) {
      super.onCreate(icicle);
      setContentView(R.layout.main);
      txtMsg = (EditText) findViewById(R.id.txtMsg);

      // deleteFile();  //keep for debugging

      Button btnFinish = (Button) findViewById(R.id.btnFinish);
      btnFinish.setOnClickListener(new Button.OnClickListener() {
         public void onClick(View v) {
            finish();
         }
      });

   }// onCreate
```

Files & Preferences

```java
public void onStart() {
   super.onStart();

   try {
❶    InputStream inputStream = openFileInput(FILE_NAME);

      if (inputStream != null) {

         BufferedReader reader = new BufferedReader(new
                                 InputStreamReader(inputStream));
❷       String str = "";
         StringBuffer stringBuffer = new StringBuffer();

         while ((str = reader.readLine()) != null) {
            stringBuffer.append(str + "\n");
         }

         inputStream.close();
         txtMsg.setText(stringBuffer.toString());
      }
   }
   catch ( Exception ex ) {
      Toast.makeText(CONTEXT, ex.getMessage() , 1).show();
   }
}// onStart
```

Files & Preferences

```java
public void onPause() {
   super.onPause();
   try {
❸    OutputStreamWriter out = new OutputStreamWriter(
                              openFileOutput(FILE_NAME, 0));
      out.write(txtMsg.getText().toString());
      out.close();
   } catch (Throwable t) {
      txtMsg.setText( t.getMessage() );
   }
}// onPause
```

```java
private void deleteFile() {
   String path = "/data/data/cis470.matos.filewriteread/files/" + FILE_NAME;
   File f1 = new File(path);
   Toast.makeText(getApplicationContext(), "Exists?" + f1.exists() , 1).show();
❹  boolean success = f1.delete();
   if (!success){
      Toast.makeText(getApplicationContext(), "Delete op. failed.", 1).show();
   }else{
      Toast.makeText(getApplicationContext(), "File deleted.", 1).show();
   }
}
```

Files & Preferences

Example2 - Comments

1. The expression **openFileInput(**FILE_NAME**)** opens a private file linked to this *Context's* application package for reading. This is an alternative to the method getResources().openRawResource(fileResourceId) discussed in the previous example.

2. A *BufferedReader* object moves data line by line from the input file to a textbox. After the buffer is emptied the data sources are closed.

3. An *OutputStreamWriter* takes the data entered by the user and send this stream to an internal file. The method **openFileOutput()** opens a private file for writing and creates the file if it doesn't already exist. The file's path is: **/data/data/packageName/FileName**

4. You may delete an existing resource file using conventional .delete() method.

Slide 21

Files & Preferences

Reading /Writing External SD Files

SD cards offer the advantage of a *much larger capacity* as well as *portability.*

Many devices allow SD cards to be easily removed and reused in another device.

SD cards are ideal for keeping your collection of music, picture, ebooks, and video files.
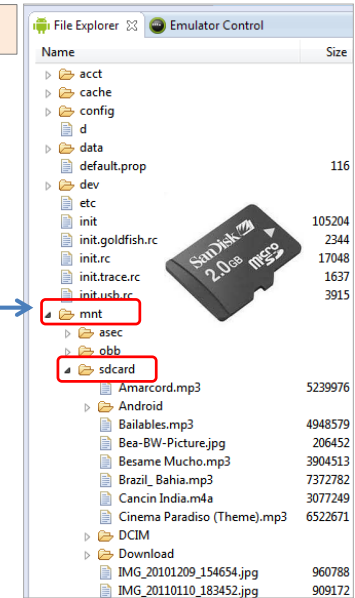

Slide 22

Files & Preferences

Reading /Writing External SD Files

Use the **File Explorer** tool to locate files in your device (or emulator).

Look into the folder: **mnt/sdcard/**
there you typically keep music, pictures, videos, etc.

Slide 23

Files & Preferences

Reading /Writing External SD Files

Although you may use the specific path to an SD file, such as:

mnt/sdcard/mysdfile.txt

it is a better practice to determine the SD location as suggested below

```
String sdPath = Environment.getExternalStorageDirectory().getAbsolutePath() ;
```

WARNING
When you deal with external files you need to request permission to read and write to the SD card.   Add the following clauses to your AndroidManifest.xml

```
<uses-permission  android:name="android.permission.READ_EXTERNAL_STORAGE"/>

<uses-permission  android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```
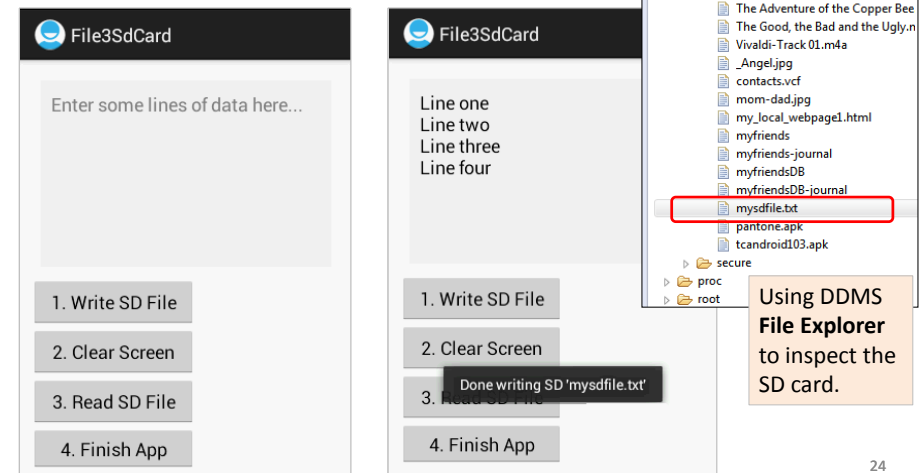

Slide 24

Files & Preferences

Example 3. Reading /Writing External SD Files

This app accepts a few lines of user input and writes it to the external SD card. User clicks on buttons to either have the data read and brought back, or terminate the app.

Using DDMS **File Explorer** to inspect the SD card.

Files & Preferences

Example 3. Reading /Writing External SD Files

Layout

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/widget28"
    android:padding="10dp"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <EditText
        android:id="@+id/txtData"
        android:layout_width="match_parent"
        android:layout_height="180dp"
        android:layout_margin="10dp"
        android:background="#55dddddd"
        android:padding="10dp"
        android:gravity="top"
        android:hint=
        "Enter some lines of data here..."
        android:textSize="18sp" />
    <Button
        android:id="@+id/btnWriteSDFile"
        android:layout_width="160dp"
        android:layout_height="wrap_content"
        android:text="1. Write SD File" />

    <Button
        android:id="@+id/btnClearScreen"
        android:layout_width="160dp"
        android:layout_height="wrap_content"
        android:text="2. Clear Screen" />
    <Button
        android:id="@+id/btnReadSDFile"
        android:layout_width="160dp"
        android:layout_height="wrap_content"
        android:text="3. Read SD File" />
    <Button
        android:id="@+id/btnFinish"
        android:layout_width="160dp"
        android:layout_height="wrap_content"
        android:text="4. Finish App" />

</LinearLayout>
```
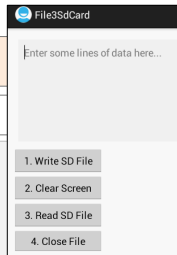
File3SdCard

Enter some lines of data here...

1. Write SD File
2. Clear Screen
3. Read SD File
4. Close File

Files & Preferences

Example 3. Reading /Writing External SD Files

```java
public class File3SdCard extends Activity {
    // GUI controls
    private EditText txtData;
    private Button btnWriteSDFile;
    private Button btnReadSDFile;
    private Button btnClearScreen;
    private Button btnClose;
    private String mySdPath;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // find SD card absolute location
❶   mySdPath = Environment.getExternalStorageDirectory().getAbsolutePath();

        // bind GUI elements to local controls
        txtData = (EditText) findViewById(R.id.txtData);
        txtData.setHint("Enter some lines of data here...");
```

Files & Preferences

Example 3. Reading /Writing External SD Files

```java
    btnWriteSDFile = (Button) findViewById(R.id.btnWriteSDFile);
    btnWriteSDFile.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            // WRITE on SD card file data taken from the text box
            try {
❷             File myFile = new File(mySdPath + "/mysdfile.txt");

              OutputStreamWriter myOutWriter = new OutputStreamWriter(
                                    new FileOutputStream(myFile));

              myOutWriter.append(txtData.getText());
              myOutWriter.close();

              Toast.makeText(getBaseContext(),
                  "Done writing SD 'mysdfile.txt'",
                    Toast.LENGTH_SHORT).show();
            } catch (Exception e) {
              Toast.makeText(getBaseContext(), e.getMessage(),
                    Toast.LENGTH_SHORT).show();
            }
        }// onClick
    }); // btnWriteSDFile
```

Files & Preferences

Example 3. Reading /Writing External SD Files

```java
    btnReadSDFile = (Button) findViewById(R.id.btnReadSDFile);
    btnReadSDFile.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            // READ data from SD card show it in the text box
            try {
              BufferedReader myReader = new BufferedReader(
                                new InputStreamReader(
                                new FileInputStream(
❸                               new File(mySdPath + "/mysdfile.txt"))));
              String aDataRow = "";
              String aBuffer = "";
              while ((aDataRow = myReader.readLine()) != null) {
                aBuffer += aDataRow + "\n";
              }
              txtData.setText(aBuffer);
              myReader.close();
              Toast.makeText(getApplicationContext(),
                  "Done reading SD 'mysdfile.txt'", Toast.LENGTH_SHORT).show();
            } catch (Exception e) {
              Toast.makeText(getApplicationContext(), e.getMessage(),
                    Toast.LENGTH_SHORT).show();
            }
        }// onClick
    }); // btnReadSDFile
```

Files & Preferences

Example 3. Reading /Writing External SD Files 4 of 4

```java
        btnClearScreen = (Button) findViewById(R.id.btnClearScreen);
        btnClearScreen.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // clear text box
                txtData.setText("");
            }
        }); // btnClearScreen

        btnClose = (Button) findViewById(R.id.btnFinish);
        btnClose.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // terminate app
                Toast.makeText(getApplicationContext(),
                        "Adios...", Toast.LENGTH_SHORT).show();
                finish();
            }
        }); // btnClose

    }// onCreate

}// File3SdCard
```
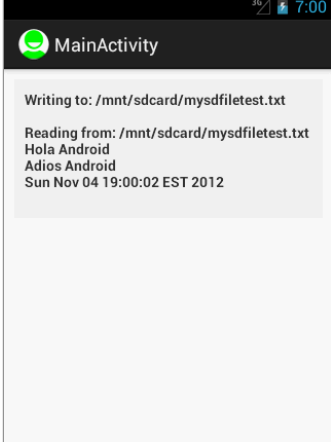

Files & Preferences

Example 4. Using Scanner/PrintWriter on External SD Files 1 of 3

In this example we use the Scanner and PrintWriter classes. Scanners are useful for dissecting formatted input into simple **tokens**. *Whitespace* markers separate the tokens, which could be translated according to their data type.


```
Writing to: /mnt/sdcard/mysdfiletest.txt

Reading from: /mnt/sdcard/mysdfiletest.txt
Hola Android
Adios Android
Sun Nov 04 19:00:02 EST 2012
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_margin="10dp"
    >

<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:padding="10dp"
    android:id="@+id/txtMsg"
    android:textStyle="bold"
    android:background="#77eeeeee"
    />
</LinearLayout>
```

Files & Preferences

Example 4. Using Scanner/PrintWriter on External SD Files 2 of 3

```java
public class File4Scanner extends Activity  {
TextView txtMsg;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        txtMsg = (TextView) findViewById(R.id.txtMsg);
        testScannedFile();
    }//onCreate

    private void testScannedFile(){
        try {
            String SDcardPath = Environment.getExternalStorageDirectory().getPath();
            String mySDFileName = SDcardPath + "/" + "mysdfiletest.txt";

            txtMsg.setText("Writing to: " + mySDFileName);
            // write to SD, needs "android.permission.WRITE_EXTERNAL_STORAGE"
            PrintWriter outfile= new PrintWriter( new FileWriter(mySDFileName) );

            outfile.println("Hola Android");
            outfile.println("Adios Android");
            outfile.println(new Date().toString());

            outfile.close();
```

❶ ❷

Files & Preferences

Example 4. Using Scanner/PrintWriter on External SD Files 3 of 3

```java
            // read SD-file, show records.
            // needs permission "android.permission.READ_EXTERNAL_STORAGE"

            Scanner infile= new Scanner(new FileReader(mySDFileName));
            String inString= "\n\nReading from: " + mySDFileName + "\n";

            while(infile.hasNextLine()) {
                inString += infile.nextLine() + "\n";
            }

            txtMsg.append(inString);
            infile.close();

        } catch (FileNotFoundException e) {
            txtMsg.setText( "Error: " + e.getMessage());
        } catch (IOException e) {
            txtMsg.setText( "Error: " + e.getMessage());
        }

    }//testScannerFiles

}//class
```
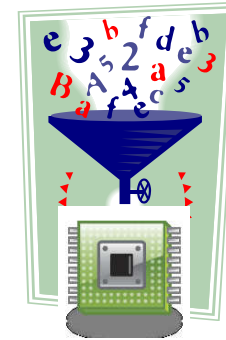
❸

Files & Preferences

Example 4. Comments

1. You want to use the method
   Environment.*getExternalStorageDirectory().getPath()*
   to detemine the path to the external SD card.

2. A PrintWriter object is used to send data tokens to disk using any of the following
   methods: `print()`, `println()`, `printf()`.

3. A Scanner accepts whitespace separated tokens and converts then to their
   corresponding types using methods: `next()`, `nextInt()`, `nextDouble()`,
   etc.

Files & Preferences

Questions ?