Android writing files to the external storage device

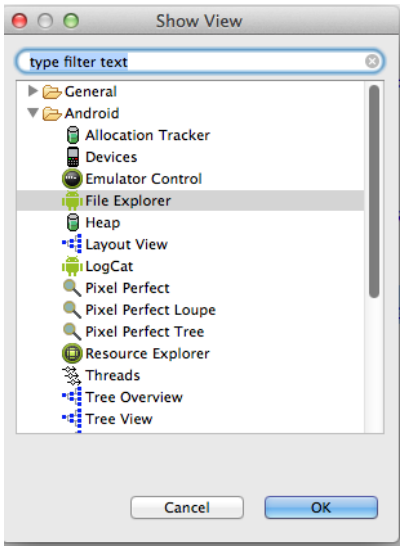
The external storage area is what Android knows as the SD card. There is a virtual SD card within the Android file system although this may be of size 0. This is not the same location as the SD card that may be added through a reader slot.

If an application writes to the SD card other applications and the user can potentially access files at that location.

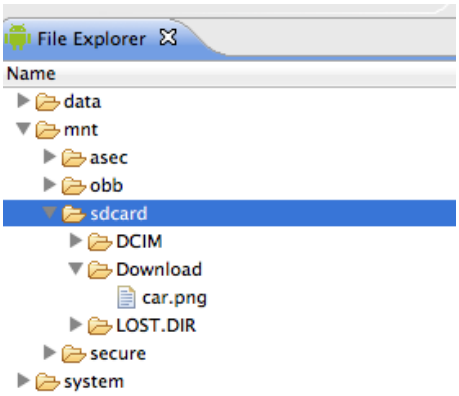
To write to the SD location permission must be set in the AndroidManifest.xml file.

This xml code has been added towards the end of the file:

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



The files can be browsed within the AVD from the File Explore view



Here the directory Download within mnt/sdcard has been created using the + button on the view. The file car.png will be transferred over by the Android program.

Note that the directory is named Download but the code refers to it as `.DIRECTORY_DOWNLOADS`.

The following code is from the developer.android.com site. The necessary imports have been fixed and the file car.png placed in res/drawable-hdpi. There are no controls on the GUI. The code will run as the activity loads.

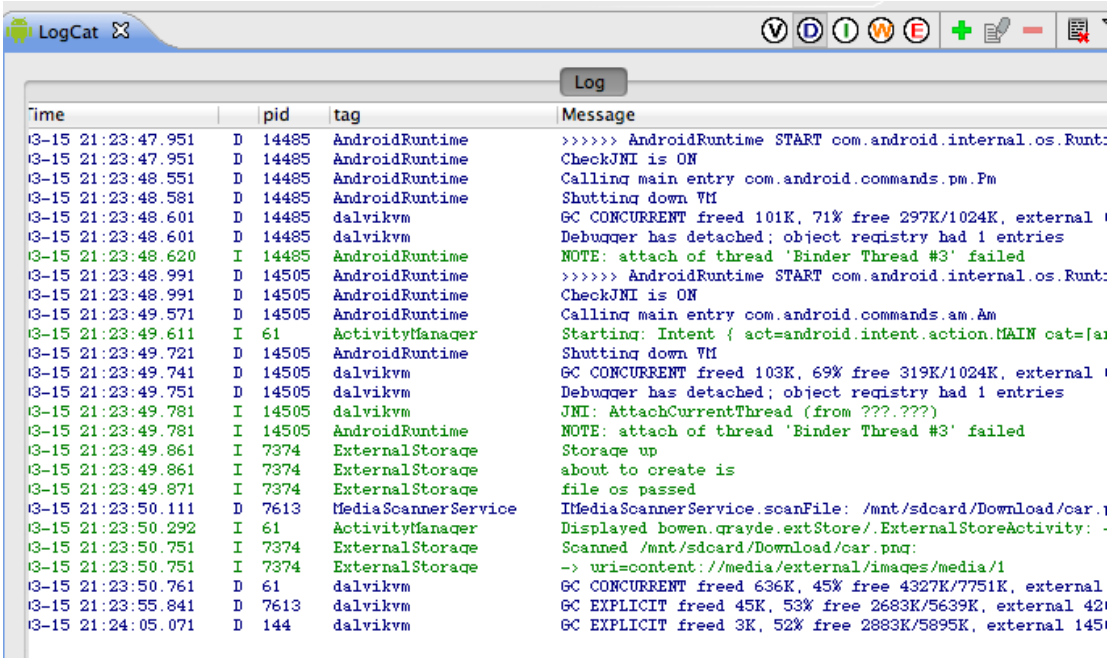
For the code to run correctly there must be a file named car.png within one of the res/drawable folders of the internal SD card.

```
public class ExternalStoreActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
       //storage permission also set in AndroidManifest.xml

      if(Environment.getExternalStorageState().equals(Environme
      nt.MEDIA_MOUNTED))
            Log.i("ExternalStorage", "Storage up");
           //if this runs storage is up
       else
            Log.i("ExternalStorage", "Storage down");
        createExternalStoragePublicPicture();
}
// end of onCreate
void createExternalStoragePublicPicture() {
// Create a path where we will place our picture in the user's
// public pictures directory.
      File path =
      Environment.getExternalStoragePublicDirectory(
                Environment.DIRECTORY_DOWNLOADS);
     File file = new File(path, "car.png");
     try {
            // Make sure the Pictures directory exists.
            path.mkdirs();
           // .  Note that this code does no error checking,
            Log.i("ExternalStorage", "about to create is");
            InputStream is =
           getResources().openRawResource(R.drawable.car);
            OutputStream os = new FileOutputStream(file);
            byte[] data = new byte[is.available()];
            is.read(data);
            os.write(data);
            is.close();
            os.close();
            Log.i("ExternalStorage", "file os passed");
           // Tell the media scanner about the new file so
           //that it is immediately available to the user.
            MediaScannerConnection.scanFile(this,
                    new String[] { file.toString() }, null,
                    new
MediaScannerConnection.OnScanCompletedListener() {
                @Override
                      public void onScanCompleted(String path,
                      Uri uri) {
                    Log.i("ExternalStorage", "Scanned " + path
                    + ":");
                    Log.i("ExternalStorage", "-> uri=" + uri);

                      }
            });
        } catch (IOException e) {
           // Unable to create file, likely because external
           //storage is not currently mounted.
            Log.w("ExternalStorage", "Error writing " + file,
            e);
        }
    }
```

The Log files write output to LogCat a useful view that shows how far the file writing has got and give some idea of where it might have gone wrong. Here the 7374 output is of interest. If something is going wrong the output will probably be in orange or red.



To get the file back or load any file from the SD card we need the absolute path of the file. There is no error handling here and the file name is hard coded but the path is picked up and the image loaded.

```
String getExternalStoragePublicPicture() {
      File path =
      Environment.getExternalStoragePublicDirectory(
                  Environment.DIRECTORY_DOWNLOADS);
        File file = new File(path, "car.png");
         String fullPath = file.getAbsolutePath();
         return fullPath;
    }
```

The above code will only be of any use if the file car.png exists in the Download directory of the external SD card.

The method is called and the String containing the absolute file path assigned to an ImageView (`ivDisplay`).

```
String fileName = getExternalStoragePublicPicture() ;
Log.i("filename is " , fileName);
Bitmap myBitmap = BitmapFactory.decodeFile(fileName);
ivDisplay.setImageBitmap(myBitmap);
```

File browsing.

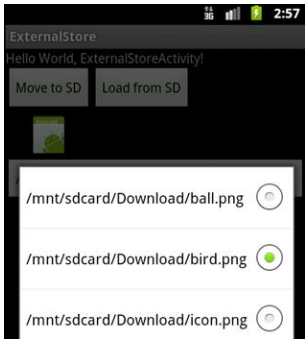
There is no built-in file browsing widget but we can find all the files in a stated path and put them in an array.

```
void findFilesinPath()
{
    File path = Environment.getExternalStoragePublicDirectory(
                Environment.DIRECTORY_DOWNLOADS);
    String sPath = path.getAbsolutePath();
    File f = new File(sPath);
    File[] files = f.listFiles();
    for(int x=0;x<files.length;x++)
           Log.i("Path files", files[x].getAbsolutePath());

    }
```

The log output shows that 2 files (only 2 were present) have been found and their paths identified.

03-19 19:22:38.204: INFO/Path files(5020): /mnt/sdcard/Download/hang0.png
03-19 19:22:38.204: INFO/Path files(5020): /mnt/sdcard/Download/car.png



A straightforward, if ugly, approach to picking files is to map the array into the rows of a Spinner object.

The user can then select an item from the dropdown on the spinner.

The following code is based on that used in Linking Andriod and XML. The data from the spinner (`spList`) is output to a Toast for checking.

This section is added at the end of `findFilesInPath()`

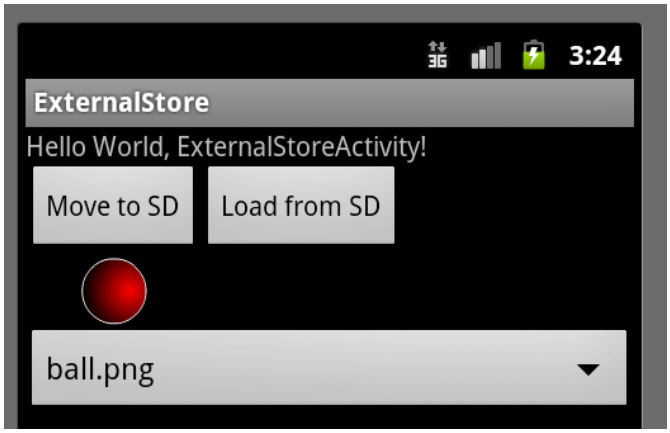
```
String []fileNames = new String[files.length];
for(int x=0;x<files.length;x++)
           fileNames[x]=files[x].getAbsolutePath();
if(files.length >0)
{//MyOnItemSelectedListener() is declared later
           spList.setOnItemSelectedListener(new
           MyOnItemSelectedListener());
           ArrayAdapter<String> aa=new
           ArrayAdapter<String>(this,
           android.R.layout.simple_spinner_item,
           fileNames);
           aa.setDropDownViewResource(
           android.R.layout.simple_spinner_dropdown_item);
             spList.setAdapter(aa);
 } }//method ends
```

This is the code for the inner class:

```
public class MyOnItemSelectedListener implements
OnItemSelectedListener {
          public void onItemSelected(AdapterView<?>
          parent,View view, int pos, long id) {
                    Toast.makeText(parent.getContext(),
          "The file is " +
          parent.getItemAtPosition(pos).toString(),
          Toast.LENGTH_LONG).show();
          }
          public void onNothingSelected(AdapterView
          parent) {
                    // Do nothing
            }
}
```

We need to take the name of the file and place it in the path that is used by the internal SD card. This can be achieved by overloading the existing method

```
String getExternalStoragePublicPicture(String fileName) {
        // new parameter added for the filename
     File path =
     Environment.getExternalStoragePublicDirectory(
               Environment.DIRECTORY_DOWNLOADS);
     File file = new File(path, fileName);
     //filename not car.png
     String fullPath ;
     fullPath = file.getAbsolutePath();
     return fullPath;
}
```



As it is the `onItemSelected` method of `MyOnItemSelectedListener` that chooses an item from the list. The same method can be used to move that image to the ImageView. The load from SD Button is now redundant.

```
public void onItemSelected(AdapterView<?> parent,View
view, int pos, long id)
{
      String fileName =
      getExternalStoragePublicPicture(parent.getItemAtPosi
      tion(pos).toString()) ;
      Bitmap myBitmap =
      BitmapFactory.decodeFile(fileName);
      ivDisplay.setImageBitmap(myBitmap);
}
```