public class

Environment

extends Object

java.lang.Object
↳ android.os.Environment

Class Overview

Provides access to environment variables.

Summary

Constants		
String	MEDIA_BAD_REMOVAL	`getExternalStorageState()` returns MEDIA_BAD_REMOVAL if the media was removed before it was unmounted.
String	MEDIA_CHECKING	`getExternalStorageState()` returns MEDIA_CHECKING if the media is present and being disk-checked
String	MEDIA_MOUNTED	`getExternalStorageState()` returns MEDIA_MOUNTED if the media is present and mounted at its mount point with read/write access.
String	MEDIA_MOUNTED_READ_ONLY	`getExternalStorageState()` returns MEDIA_MOUNTED_READ_ONLY if the media is present and mounted at its mount point with read only access.
String	MEDIA_NOFS	`getExternalStorageState()` returns MEDIA_NOFS if the media is present but is blank or is using an unsupported filesystem

String	MEDIA_REMOVED	`getExternalStorageState()` returns MEDIA_REMOVED if the media is not present.
String	MEDIA_SHARED	`getExternalStorageState()` returns MEDIA_SHARED if the media is present not mounted, and shared via USB mass storage.
String	MEDIA_UNMOUNTABLE	`getExternalStorageState()` returns MEDIA_UNMOUNTABLE if the media is present but cannot be mounted.
String	MEDIA_UNMOUNTED	`getExternalStorageState()` returns MEDIA_UNMOUNTED if the media is present but not mounted.

Fields		
public static String	DIRECTORY_ALARMS	Standard directory in which to place any audio files that should be in the list of alarms that the user can select (not as regular music).
public static String	DIRECTORY_DCIM	The traditional location for pictures and videos when mounting the device as a camera.
public static String	DIRECTORY_DOWNLOADS	Standard directory in which to place files that have been downloaded by the user.
public static String	DIRECTORY_MOVIES	Standard directory in which to place movies that are available to the user.
public static String	DIRECTORY_MUSIC	Standard directory in which to place any audio files that should be in the regular list of music for the user.
public static	DIRECTORY_NOTIFICATIONS	Standard directory in which to place any audio files that should be in the list of notifications that the

String		user can select (not as regular music).
public static String	DIRECTORY_PICTURES	Standard directory in which to place pictures that are available to the user.
public static String	DIRECTORY_PODCASTS	Standard directory in which to place any audio files that should be in the list of podcasts that the user can select (not as regular music).
public static String	DIRECTORY_RINGTONES	Standard directory in which to place any audio files that should be in the list of ringtones that the user can select (not as regular music).

Public Constructors	
	Environment()

Public Methods	
static File	getDataDirectory() Gets the Android data directory.
static File	getDownloadCacheDirectory() Gets the Android Download/Cache content directory.
static File	getExternalStorageDirectory() Gets the Android external storage directory.
static File	getExternalStoragePublicDirectory(String type) Get a top-level public external storage directory for placing files of a particular type.
static String	getExternalStorageState() Gets the current state of the primary "external" storage device.

static File	getRootDirectory() Gets the Android root directory.
static boolean	isExternalStorageEmulated() Returns whether the device has an external storage device which is emulated.
static boolean	isExternalStorageRemovable() Returns whether the primary "external" storage device is removable.

[Expand]

Inherited Methods

▶From class java.lang.Object

Object	clone() Creates and returns a copy of this `Object`.
boolean	equals(Object o) Compares this instance with the specified object and indicates if they are equal.
void	finalize() Invoked when the garbage collector has detected that this instance is no longer reachable.
final Class<?>	getClass() Returns the unique instance of `Class` that represents this object's class.
int	hashCode() Returns an integer hash code for this object.

final void	notify() Causes a thread which is waiting on this object's monitor (by means of calling one of the `wait()` methods) to be woken up.
final void	notifyAll() Causes all threads which are waiting on this object's monitor (by means of calling one of the `wait()` methods) to be woken up.
String	toString() Returns a string containing a concise, human-readable description of this object.
final void	wait() Causes the calling thread to wait until another thread calls the `notify()` or `notifyAll()` method of this object.
final void	wait(long millis, int nanos) Causes the calling thread to wait until another thread calls the `notify()` or `notifyAll()` method of this object or until the specified timeout expires.
final void	wait(long millis) Causes the calling thread to wait until another thread calls the `notify()` or `notifyAll()` method of this object or until the specified timeout expires.

Constants

public static final String **MEDIA_BAD_REMOVAL**

Since: API Level 1

`getExternalStorageState()` returns MEDIA_BAD_REMOVAL if the media was removed before it was unmounted.

Constant Value: "bad_removal"

public static final String **MEDIA_CHECKING**

Since: API Level 3

`getExternalStorageState()` returns MEDIA_CHECKING if the media is present and being disk-checked

Constant Value: "checking"

public static final String **MEDIA_MOUNTED**

Since: API Level 1

`getExternalStorageState()` returns MEDIA_MOUNTED if the media is present and mounted at its mount point with read/write access.

Constant Value: "mounted"

public static final String **MEDIA_MOUNTED_READ_ONLY**

Since: API Level 1

`getExternalStorageState()` returns MEDIA_MOUNTED_READ_ONLY if the media is present and mounted at its mount point with read only access.

Constant Value: "mounted_ro"

public static final String **MEDIA_NOFS**

Since: API Level 3

`getExternalStorageState()` returns MEDIA_NOFS if the media is present but is blank or is using an unsupported filesystem

Constant Value: "nofs"

public static final String **MEDIA_REMOVED**

Since: API Level 1

`getExternalStorageState()` returns MEDIA_REMOVED if the media is not present.

Constant Value: "removed"

public static final String **MEDIA_SHARED**

Since: API Level 1

`getExternalStorageState()` returns MEDIA_SHARED if the media is present not mounted, and shared via USB mass storage.

Constant Value: "shared"

public static final String **MEDIA_UNMOUNTABLE**

Since: API Level 1

`getExternalStorageState()` returns MEDIA_UNMOUNTABLE if the media is present but cannot be mounted. Typically this happens if the file system on the media is corrupted.

Constant Value: "unmountable"

public static final String **MEDIA_UNMOUNTED**

Since: API Level 1

`getExternalStorageState()` returns MEDIA_UNMOUNTED if the media is present but not mounted.

Constant Value: "unmounted"

Fields

public static String **DIRECTORY_ALARMS**

Since: API Level 8

Standard directory in which to place any audio files that should be in the list of alarms that the user can select (not as regular music). This may be combined with `DIRECTORY_MUSIC`, `DIRECTORY_PODCASTS`, `DIRECTORY_NOTIFICATIONS`, and `DIRECTORY_RINGTONES` as a series of directories to categories a particular audio file as more than one type.

public static String **DIRECTORY_DCIM**

Since: API Level 8

The traditional location for pictures and videos when mounting the device as a camera. Note that this is primarily a convention for the top-level public directory, as this convention makes no sense elsewhere.

public static String **DIRECTORY_DOWNLOADS**

Since: API Level 8

Standard directory in which to place files that have been downloaded by the user. Note that this is primarily a convention for the top-level public directory, you are free to download files anywhere in your own private directories. Also note that though the constant here is named DIRECTORY_DOWNLOADS (plural), the actual file name is non-plural for backwards compatibility reasons.

public static String **DIRECTORY_MOVIES**

Since: API Level 8

Standard directory in which to place movies that are available to the user. Note that this is primarily a convention for the top-level public directory, as the media scanner will find and collect movies in any directory.

public static String **DIRECTORY_MUSIC**

Since: API Level 8

Standard directory in which to place any audio files that should be in the regular list of music for the user. This may be combined with `DIRECTORY_PODCASTS`, `DIRECTORY_NOTIFICATIONS`, `DIRECTORY_ALARMS`, and `DIRECTORY_RINGTONES` as a series of directories to categories a particular audio file as more than one type.

public static String **DIRECTORY_NOTIFICATIONS**

Since: API Level 8

Standard directory in which to place any audio files that should be in the list of notifications that the user can select (not as regular music). This may be combined with `DIRECTORY_MUSIC`, `DIRECTORY_PODCASTS`, `DIRECTORY_ALARMS`, and `DIRECTORY_RINGTONES` as a series of directories to categories a particular audio file as more than one type.

public static String **DIRECTORY_PICTURES**

Since: API Level 8

Standard directory in which to place pictures that are available to the user. Note that this is primarily a convention for the top-level public directory, as the media scanner will find and collect pictures in any directory.

public static String **DIRECTORY_PODCASTS**

Since: API Level 8

Standard directory in which to place any audio files that should be in the list of podcasts that the user can select (not as regular music). This may be combined with `DIRECTORY_MUSIC`, `DIRECTORY_NOTIFICATIONS`, `DIRECTORY_ALARMS`, and `DIRECTORY_RINGTONES` as a series of directories to categories a particular audio file as more than one type.

public static String **DIRECTORY_RINGTONES**

Since: API Level 8

Standard directory in which to place any audio files that should be in the list of ringtones that the user can select (not as regular music). This may be combined with `DIRECTORY_MUSIC`, `DIRECTORY_PODCASTS`, `DIRECTORY_NOTIFICATIONS`, and `DIRECTORY_ALARMS` as a series of directories to categories a particular audio file as more than one type.

Public Constructors

public **Environment**()

Since: API Level 1

Public Methods

public static File**getDataDirectory**()

Since: API Level 1

Gets the Android data directory.

public static File**getDownloadCacheDirectory**()

Since: API Level 1

Gets the Android Download/Cache content directory.

public static File**getExternalStorageDirectory**()

Since: API Level 1

Gets the Android external storage directory. This directory may not currently be accessible if it has been mounted by the user on their computer, has been removed from the device, or some other problem has happened. You can determine its current state with `getExternalStorageState()`.

Note: don't be confused by the word "external" here. This directory can better be thought as media/shared storage. It is a filesystem that can hold a relatively large amount of data and that is shared across all applications (does not enforce permissions). Traditionally this is an SD card, but it may also be implemented as built-in storage in a device that is distinct from the protected internal storage and can be mounted as a filesystem on a computer.

In devices with multiple "external" storage directories (such as both secure app storage and mountable shared storage), this directory represents the "primary" external storage that the user will interact with.

Applications should not directly use this top-level directory, in order to avoid polluting the user's root namespace. Any files that are private to the application should be placed in a directory returned by `Context.getExternalFilesDir`, which the system will take care of deleting if the application is uninstalled. Other shared files should be placed in one of the directories returned by `getExternalStoragePublicDirectory(String)`.

Here is an example of typical code to monitor the state of external storage:

```
BroadcastReceiver
mExternalStorageReceiver;
boolean
mExternalStorageAvailable =
false;
boolean
mExternalStorageWriteable =
false;

void
updateExternalStorageState(
) {
   String state =
Environment.getExternalStor
ageState();
   if
(Environment.MEDIA_MOUNTED.
equals(state)) {

mExternalStorageAvailable =
mExternalStorageWriteable =
true;
   } else if
(Environment.MEDIA_MOUNTED_
READ_ONLY.equals(state)) {

mExternalStorageAvailable =
true;

mExternalStorageWriteable =
false;
   } else {

mExternalStorageAvailable =
mExternalStorageWriteable =
false;
   }

handleExternalStorageState(
mExternalStorageAvailable,

mExternalStorageWriteable);
}
```

```
void
startWatchingExternalStorag
e() {
   mExternalStorageReceiver
= new BroadcastReceiver() {
      @Override
      public void
onReceive(Context context,
Intent intent) {
          Log.i("test",
"Storage: " +
intent.getData());

updateExternalStorageState(
);
      }
   };
   IntentFilter filter = new
IntentFilter();

filter.addAction(Intent.ACT
ION_MEDIA_MOUNTED);

filter.addAction(Intent.ACT
ION_MEDIA_REMOVED);

registerReceiver(mExternalS
torageReceiver, filter);

updateExternalStorageState(
);
}

void
stopWatchingExternalStorage
() {

unregisterReceiver(mExterna
lStorageReceiver);
}
```

See Also

- `getExternalStorageState()`
- `isExternalStorageRemovable()`

public static File**getExternalStoragePublicDirectory**(String type)

Since: API Level 8

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, so you should be careful about what you put here to ensure you don't erase their files or get in the way of their own organization.

Here is an example of typical code to manipulate a picture on the public external storage:

```
void
createExternalStoragePublic
Picture() {
    // Create a path where we
will place our picture in the
user's
    // public pictures
directory.  Note that you
should be careful about
    // what you place here,
since the user often manages
these files.  For
    // pictures and other
media owned by the
application, consider
    //
Context.getExternalMediaDir
().
    File path =
Environment.getExternalStor
agePublicDirectory(

Environment.DIRECTORY_PICTU
RES);
    File file = new File(path,
"DemoPicture.jpg");

    try {
        // Make sure the
Pictures directory exists.
```

```
        path.mkdirs();

        // Very simple code to
copy a picture from the
application's
        // resource into the
external file.  Note that this
code does
        // no error checking,
and assumes the picture is
small (does not
        // try to copy it in
chunks).  Note that if
external storage is
        // not currently
mounted this will silently
fail.
        InputStream is =
getResources().openRawResou
rce(R.drawable.balloons);
        OutputStream os = new
FileOutputStream(file);
        byte[] data = new
byte[is.available()];
        is.read(data);
        os.write(data);
        is.close();
        os.close();

        // Tell the media
scanner about the new file so
that it is
        // immediately
available to the user.

MediaScannerConnection.scan
File(this,
                new String[]
{ file.toString() }, null,
                new
MediaScannerConnection.OnSc
anCompletedListener() {
            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);
   }
}

void
deleteExternalStoragePublic
Picture() {
   // Create a path where we
will place our picture in the
user's
   // public pictures
directory and delete the file.
If external
   // storage is not
currently mounted this will
fail.
   File path =
Environment.getExternalStor
agePublicDirectory(

Environment.DIRECTORY_PICTU
RES);
   File file = new File(path,
"DemoPicture.jpg");
   file.delete();
}
```

```
boolean
hasExternalStoragePublicPic
ture() {
    // Create a path where we
will place our picture in the
user's
    // public pictures
directory and check if the
file exists.  If
    // external storage is not
currently mounted this will
think the
    // picture doesn't exist.
    File path =
Environment.getExternalStor
agePublicDirectory(

Environment.DIRECTORY_PICTU
RES);
    File file = new File(path,
"DemoPicture.jpg");
    return file.exists();
}
```

Parameters

type	The type of storage directory to return. Should be one of `DIRECTORY_MUSIC`, `DIRECTORY_PODCASTS`, `DIRECTORY_RINGTONES`, `DIRECTORY_ALARMS`, `DIRECTORY_NOTIFICATIONS`, `DIRECTORY_PICTURES`, `DIRECTORY_MOVIES`, `DIRECTORY_DOWNLOADS`, or `DIRECTORY_DCIM`. May not be null.

Returns

- Returns the File path for the directory. Note that this directory may not yet exist, so you must make sure it exists before using it such as with `File.mkdirs()`.

public static String**getExternalStorageState**()

Since: API Level 1

Gets the current state of the primary "external" storage device.

See `getExternalStorageDirectory()` for more information.

public static File**getRootDirectory**()

Since: API Level 1

Gets the Android root directory.

public static boolean **isExternalStorageEmulated**()

Since: API Level 11

Returns whether the device has an external storage device which is emulated. If true, the device does not have real external storage, and the directory returned by `getExternalStorageDirectory()` will be allocated using a portion of the internal storage system.

Certain system services, such as the package manager, use this to determine where to install an application.

Emulated external storage may also be encrypted - see `setStorageEncryption(android.content.ComponentName, boolean)` for additional details.

public static boolean **isExternalStorageRemovable**()

Since: API Level 9

Returns whether the primary "external" storage device is removable. If true is returned, this device is for example an SD card that the user can remove. If false is returned, the storage is built into the device and can not be physically removed.

See `getExternalStorageDirectory()` for more information.