public class

Summary: Constants | Fields | Ctors | Methods | Inherited Methods | [Expand All]

Added in API level 1

Environment

extends Object

java.lang.Object
↳ android.os.Environment

Class Overview

Provides access to environment variables.

Summary

Constants	
String MEDIA_BAD_REMOVAL	Storage state if the media was removed before it was unmounted.
String MEDIA_CHECKING	Storage state if the media is present and being disk-checked.
String MEDIA_MOUNTED	Storage state if the media is present and mounted at its mount point with read/write access.
String MEDIA_MOUNTED_READ_ONLY	Storage state if the media is present and mounted at its mount point with read-only access.
String MEDIA_NOFS	Storage state if the media is present but is blank or is using an unsupported filesystem.
String MEDIA_REMOVED	Storage state if the media is not present.
String MEDIA_SHARED	Storage state if the media is present not mounted, and shared via USB mass storage.
String MEDIA_UNKNOWN	Unknown storage state, such as when a path isn't backed by known storage media.

String MEDIA_UNMOUNTABLE	Storage state if the media is present but cannot be mounted.
String MEDIA_UNMOUNTED	Storage state 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_DOCUMENTS	Standard directory in which to place documents that have been created by the user.
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 String DIRECTORY_NOTIFICATIONS	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).
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 () Return the user data directory.
static File	getDownloadCacheDirectory () Return the download/cache content directory.
static File	getExternalStorageDirectory () Return the primary 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 () Returns the current state of the primary "external" storage device.
static File	getRootDirectory () Gets the Android root directory.
static String	getStorageState (File path) Returns the current state of the storage device that provides the given path.
static boolean	isExternalStorageEmulated () Returns whether the device has an external storage device which is emulated.

	isExternalStorageRemovable()
static boolean	Returns whether the primary "external" storage device is removable.

Inherited Methods [Expand]

- From class java.lang.Object

Constants

public static final String **MEDIA_BAD_REMOVAL** Added in API level 1

Storage state if the media was removed before it was unmounted.

See Also

`getStorageState(File)`

Constant Value: "bad_removal"

public static final String **MEDIA_CHECKING** Added in API level 3

Storage state if the media is present and being disk-checked.

See Also

`getStorageState(File)`

Constant Value: "checking"

public static final String **MEDIA_MOUNTED** Added in API level 1

Storage state if the media is present and mounted at its mount point with read/write access.

See Also

`getStorageState(File)`

Constant Value: "mounted"

public static final String **MEDIA_MOUNTED_READ_ONLY** Added in API level 1

Storage state if the media is present and mounted at its mount point with read-only access.

See Also

`getStorageState(File)`

Constant Value: "mounted_ro"

public static final String **MEDIA_NOFS** Added in API level 3

Storage state if the media is present but is blank or is using an unsupported filesystem.

See Also

`getStorageState(File)`

Constant Value: "nofs"

public static final String **MEDIA_REMOVED** Added in API level 1

Storage state if the media is not present.

See Also

`getStorageState(File)`

Constant Value: "removed"

public static final String **MEDIA_SHARED** Added in API level 1

Storage state if the media is present not mounted, and shared via USB mass storage.

See Also

`getStorageState(File)`

Constant Value: "shared"

public static final String **MEDIA_UNKNOWN** Added in API level 19

Unknown storage state, such as when a path isn't backed by known storage media.

See Also

`getStorageState(File)`

Constant Value: "unknown"

public static final String **MEDIA_UNMOUNTABLE** Added in API level 1

Storage state if the media is present but cannot be mounted. Typically this happens if the file system on the media is corrupted.

See Also

`getStorageState(File)`

Constant Value: "unmountable"

public static final String **MEDIA_UNMOUNTED**

Storage state if the media is present but not mounted.

See Also
getStorageState(File)

Constant Value: "unmounted"

Fields

public static String **DIRECTORY_ALARMS** Added in 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 (/reference/android/os/Environment.html#DIRECTORY_MUSIC), DIRECTORY_PODCASTS (/reference/android/os/Environment.html#DIRECTORY_PODCASTS), DIRECTORY_NOTIFICATIONS (/reference/android/os/Environment.html#DIRECTORY_NOTIFICATIONS), and DIRECTORY_RINGTONES (/reference/android/os/Environment.html#DIRECTORY_RINGTONES) as a series of directories to categories a particular audio file as more than one type.

public static String **DIRECTORY_DCIM** Added in 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_DOCUMENTS** Added in API level 19

Standard directory in which to place documents that have been created by the user.

public static String **DIRECTORY_DOWNLOADS** Added in 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** Added in 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** Added in 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 (/reference/android/os/Environment.html#DIRECTORY_PODCASTS), DIRECTORY_NOTIFICATIONS (/reference/android/os/Environment.html#DIRECTORY_NOTIFICATIONS), DIRECTORY_ALARMS (/reference/android/os/Environment.html#DIRECTORY_ALARMS), and DIRECTORY_RINGTONES (/reference/android/os/Environment.html#DIRECTORY_RINGTONES) as a series of directories to categories a particular audio file as more than one type.

public static String **DIRECTORY_NOTIFICATIONS** Added in 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 (/reference/android/os/Environment.html#DIRECTORY_MUSIC), DIRECTORY_PODCASTS (/reference/android/os/Environment.html#DIRECTORY_PODCASTS), DIRECTORY_ALARMS (/reference/android/os/Environment.html#DIRECTORY_ALARMS), and DIRECTORY_RINGTONES (/reference/android/os/Environment.html#DIRECTORY_RINGTONES) as a series of directories to categories a particular audio file as more than one type.

public static String **DIRECTORY_PICTURES** Added in 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** Added in 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 (/reference/android/os/Environment.html#DIRECTORY_MUSIC), DIRECTORY_NOTIFICATIONS (/reference/android

/os/Environment.html#DIRECTORY_NOTIFICATIONS), DIRECTORY_ALARMS (/reference/android/os/Environment.html#DIRECTORY_ALARMS), and DIRECTORY_RINGTONES (/reference/android/os/Environment.html#DIRECTORY_RINGTONES) as a series of directories to categories a particular audio file as more than one type.

public static String **DIRECTORY_RINGTONES** Added in 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 (/reference/android/os/Environment.html#DIRECTORY_MUSIC), DIRECTORY_PODCASTS (/reference/android/os/Environment.html#DIRECTORY_PODCASTS), DIRECTORY_NOTIFICATIONS (/reference/android/os/Environment.html#DIRECTORY_NOTIFICATIONS), and DIRECTORY_ALARMS (/reference/android/os/Environment.html#DIRECTORY_ALARMS) as a series of directories to categories a particular audio file as more than one type.

Public Constructors

public **Environment** () Added in API level 1

Public Methods

public static File **getDataDirectory** () Added in API level 1

Return the user data directory.

public static File **getDownloadCacheDirectory** () Added in API level 1

Return the download/cache content directory.

public static File **getExternalStorageDirectory** () Added in API level 1

Return the primary 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() (/reference/android

/os/Environment.html#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.

On devices with multiple users (as described by UserManager (/reference/android/os/UserManager.html)), each user has their own isolated external storage. Applications only have access to the external storage for the user they're running as.

In devices with multiple "external" storage directories, this directory represents the "primary" external storage that the user will interact with. Access to secondary storage is available through

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 (/reference/android/content/Context.html#getExternalFilesDir(java.lang.String)), 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) (/reference/android/os/Environment.html#getExternalStoragePublicDirectory(java.lang.String)).

Writing to this path requires the WRITE_EXTERNAL_STORAGE (/reference/android/Manifest.permission.html#WRITE_EXTERNAL_STORAGE) permission, and starting in read access requires the READ_EXTERNAL_STORAGE (/reference/android/Manifest.permission.html#READ_EXTERNAL_STORAGE) permission, which is automatically granted if you hold the write permission.

Starting in KITKAT (/reference/android/os/Build.VERSION_CODES.html#KITKAT), if your application only needs to store internal data, consider using getExternalFilesDir(String) (/reference/android/content/Context.html#getExternalFilesDir(java.lang.String)) or getExternalCacheDir() (/reference/android/content/Context.html#getExternalCacheDir()), which require no permissions to read or write.

This path may change between platform versions, so applications should only persist relative paths.

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.getExternalStorageSt
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        mExternalStorageAvailable = mExternalStorag
    } else if (Environment.MEDIA_MOUNTED_READ_ONLY.
        mExternalStorageAvailable = true;
        mExternalStorageWriteable = false;
    } else {
        mExternalStorageAvailable = mExternalStorag
    }
    handleExternalStorageState(mExternalStorageAvai
            mExternalStorageWriteable);
}

void startWatchingExternalStorage() {
    mExternalStorageReceiver = new BroadcastReceive
        @Override
        public void onReceive(Context context, Inte
            Log.i("test", "Storage: " + intent.getD
            updateExternalStorageState();
        }
    };
    IntentFilter filter = new IntentFilter();
    filter.addAction(Intent.ACTION_MEDIA_MOUNTED);
    filter.addAction(Intent.ACTION_MEDIA_REMOVED);
    registerReceiver(mExternalStorageReceiver, filt
    updateExternalStorageState();
}

void stopWatchingExternalStorage() {
    unregisterReceiver(mExternalStorageReceiver);
}
```

See Also

getExternalStorageState()
isExternalStorageRemovable()

public static File
getExternalStoragePublicDirectory (String type) Added in 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.

On devices with multiple users (as described by `UserManager` (/reference/android/os/UserManager.html)), each user has their own isolated external storage. Applications only have access to the external storage for the user they're running as.

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

```
void createExternalStoragePublicPicture() {
    // Create a path where we will place our pictur
    // public pictures directory.  Note that you sh
    // what you place here, since the user often ma
    // pictures and other media owned by the applic
    // Context.getExternalMediaDir().
    File path = Environment.getExternalStoragePubli
            Environment.DIRECTORY_PICTURES);
    File file = new File(path, "DemoPicture.jpg");

    try {
        // Make sure the Pictures directory exists.
        path.mkdirs();

        // Very simple code to copy a picture from
        // resource into the external file.  Note t
        // no error checking, and assumes the pictu
        // try to copy it in chunks).  Note that if
        // not currently mounted this will silently
        InputStream is = getResources().openRawReso
        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 fil
        // immediately available to the user.
        MediaScannerConnection.scanFile(this,
```

```
                new String[] { file.toString() }, n
                new MediaScannerConnection.OnScanCo
            public void onScanCompleted(String path
                Log.i("ExternalStorage", "Scanned "
                Log.i("ExternalStorage", "-> uri="
            }
        });
    } catch (IOException e) {
        // Unable to create file, likely because ex
        // not currently mounted.
        Log.w("ExternalStorage", "Error writing " +
    }
}

void deleteExternalStoragePublicPicture() {
    // Create a path where we will place our pictur
    // public pictures directory and delete the fil
    // storage is not currently mounted this will f
    File path = Environment.getExternalStoragePubli
            Environment.DIRECTORY_PICTURES);
    File file = new File(path, "DemoPicture.jpg");
    file.delete();
}

boolean hasExternalStoragePublicPicture() {
    // Create a path where we will place our pictur
    // public pictures directory and check if the f
    // external storage is not currently mounted th
    // picture doesn't exist.
    File path = Environment.getExternalStoragePubli
            Environment.DIRECTORY_PICTURES);
    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** ()

Added in API level 1

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

Returns

one of MEDIA_UNKNOWN, MEDIA_REMOVED, MEDIA_UNMOUNTED, MEDIA_CHECKING, MEDIA_NOFS, MEDIA_MOUNTED, MEDIA_MOUNTED_READ_ONLY, MEDIA_SHARED, MEDIA_BAD_REMOVAL, or MEDIA_UNMOUNTABLE.

See Also

`getExternalStorageDirectory()`

public static File **getRootDirectory** ()

Added in API level 1

Gets the Android root directory.

public static String **getStorageState** (File path)

Added in API level 19

Returns the current state of the storage device that provides the given path.

Returns

one of MEDIA_UNKNOWN, MEDIA_REMOVED, MEDIA_UNMOUNTED, MEDIA_CHECKING, MEDIA_NOFS, MEDIA_MOUNTED, MEDIA_MOUNTED_READ_ONLY, MEDIA_SHARED, MEDIA_BAD_REMOVAL, or MEDIA_UNMOUNTABLE.

public static boolean **isExternalStorageEmulated** ()

Added in 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()` (/reference/android/os/Environment.html#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)` (/reference/android/app/admin/DevicePolicyManager.html#setStorageEncryption(android.content.ComponentName, boolean)) for additional details.

public static boolean **isExternalStorageRemovable**
()

Added in 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()` (/reference/android/os/Environment.html#getExternalStorageDirectory()) for more information.