public class

StatFs

extends Object

java.lang.Object

↳ android.os.StatFs

Class Overview

Retrieve overall information about the space on a filesystem. This is a Wrapper for Unix statfs().

Summary

Public Constructors	
	StatFs(String path) Construct a new StatFs for looking at the stats of the filesystem at *path*.
Public Methods	
int	getAvailableBlocks() The number of blocks that are free on the file system and available to applications.
int	getBlockCount() The total number of blocks on the file system.
int	getBlockSize() The size, in bytes, of a block on the file system.
int	getFreeBlocks() The total number of blocks that are free on the file system, including reserved blocks (that

	are not available to normal applications).
void	restat(String path) Perform a restat of the file system referenced by this object.

Protected Methods

void	finalize() Invoked when the garbage collector has detected that this instance is no longer reachable.

[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.

Public Constructors

public **StatFs**(String path)

Since: API Level 1

Construct a new StatFs for looking at the stats of the filesystem at *path*. Upon construction, the stat of the file system will be performed, and the values retrieved available from the methods on this class.

Parameters

path A path in the desired file system to state.

Public Methods

public int **getAvailableBlocks**()

Since: API Level 1

The number of blocks that are free on the file system and available to applications. This corresponds to the Unix statfs.f_bavail field.

public int **getBlockCount**()

Since: API Level 1

The total number of blocks on the file system. This corresponds to the Unix statfs.f_blocks field.

public int **getBlockSize**()

Since: API Level 1

The size, in bytes, of a block on the file system. This corresponds to the Unix statfs.f_bsize field.

public int **getFreeBlocks**()

Since: API Level 1

The total number of blocks that are free on the file system, including reserved blocks (that are not available to normal applications). This corresponds to the Unix statfs.f_bfree field. Most applications will want to use `getAvailableBlocks()` instead.

public void **restat**(String path)

Since: API Level 1

Perform a restat of the file system referenced by this object. This is the same as re-constructing the object with the same file system path, and the new stat values are available upon return.

Protected Methods

protected void **finalize**()

Since: API Level 1

Invoked when the garbage collector has detected that this instance is no longer reachable. The default implementation does nothing, but this method can be overridden to free resources.

Note that objects that override `finalize` are significantly more expensive than objects that don't. Finalizers may be run a long time after the object is no longer reachable, depending on memory pressure, so it's a bad idea to rely on them for cleanup. Note also that finalizers are run on a single VM-wide finalizer thread, so doing blocking work in a finalizer is a bad idea. A finalizer is usually only necessary for a class that has a native peer and needs to call a native method to destroy that peer. Even then, it's better to provide an explicit `close` method (and implement `Closeable`), and insist that callers manually dispose of instances. This works well for something like files, but less well for something like a `BigInteger` where typical calling code would have to deal with lots of temporaries. Unfortunately, code that creates lots of temporaries is the worst kind of code from the point of view of the single finalizer thread.

If you *must* use finalizers, consider at least providing your own `ReferenceQueue` and having your own thread process that queue.

Unlike constructors, finalizers are not automatically chained. You are responsible for calling `super.finalize()` yourself.

Uncaught exceptions thrown by finalizers are ignored and do not terminate the finalizer thread. See *Effective Java* Item 7, "Avoid finalizers" for more.