| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/DataLine.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/javax/sound/sampled/Control.Type.html)   [**NEXT CLASS**](http://docs.google.com/javax/sound/sampled/DataLine.Info.html) | [**FRAMES**](http://docs.google.com/index.html?javax/sound/sampled/DataLine.html)    [**NO FRAMES**](http://docs.google.com/DataLine.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: [NESTED](#3znysh7) | FIELD | CONSTR | [METHOD](#2et92p0) | DETAIL: FIELD | CONSTR | [METHOD](#3dy6vkm) |

## **javax.sound.sampled**

Interface DataLine

**All Superinterfaces:** [Line](http://docs.google.com/javax/sound/sampled/Line.html) **All Known Subinterfaces:** [Clip](http://docs.google.com/javax/sound/sampled/Clip.html), [SourceDataLine](http://docs.google.com/javax/sound/sampled/SourceDataLine.html), [TargetDataLine](http://docs.google.com/javax/sound/sampled/TargetDataLine.html)

public interface **DataLine**extends [Line](http://docs.google.com/javax/sound/sampled/Line.html)

DataLine adds media-related functionality to its superinterface, [Line](http://docs.google.com/javax/sound/sampled/Line.html). This functionality includes transport-control methods that start, stop, drain, and flush the audio data that passes through the line. A data line can also report the current position, volume, and audio format of the media. Data lines are used for output of audio by means of the subinterfaces [SourceDataLine](http://docs.google.com/javax/sound/sampled/SourceDataLine.html) or [Clip](http://docs.google.com/javax/sound/sampled/Clip.html), which allow an application program to write data. Similarly, audio input is handled by the subinterface [TargetDataLine](http://docs.google.com/javax/sound/sampled/TargetDataLine.html), which allows data to be read.

A data line has an internal buffer in which the incoming or outgoing audio data is queued. The [drain()](http://docs.google.com/javax/sound/sampled/DataLine.html#drain()) method blocks until this internal buffer becomes empty, usually because all queued data has been processed. The [flush()](http://docs.google.com/javax/sound/sampled/DataLine.html#flush()) method discards any available queued data from the internal buffer.

A data line produces [START](http://docs.google.com/javax/sound/sampled/LineEvent.Type.html#START) and [STOP](http://docs.google.com/javax/sound/sampled/LineEvent.Type.html#STOP) events whenever it begins or ceases active presentation or capture of data. These events can be generated in response to specific requests, or as a result of less direct state changes. For example, if [start()](http://docs.google.com/javax/sound/sampled/DataLine.html#start()) is called on an inactive data line, and data is available for capture or playback, a START event will be generated shortly, when data playback or capture actually begins. Or, if the flow of data to an active data line is constricted so that a gap occurs in the presentation of data, a STOP event is generated.

Mixers often support synchronized control of multiple data lines. Synchronization can be established through the Mixer interface's [synchronize](http://docs.google.com/javax/sound/sampled/Mixer.html#synchronize(javax.sound.sampled.Line%5B%5D,%20boolean)) method. See the description of the [Mixer](http://docs.google.com/javax/sound/sampled/Mixer.html) interface for a more complete description.

**Since:** 1.3 **See Also:**[LineEvent](http://docs.google.com/javax/sound/sampled/LineEvent.html)

| **Nested Class Summary** | |
| --- | --- |
| static class | [**DataLine.Info**](http://docs.google.com/javax/sound/sampled/DataLine.Info.html)            Besides the class information inherited from its superclass, DataLine.Info provides additional information specific to data lines. |

| **Method Summary** | |
| --- | --- |
| int | [**available**](http://docs.google.com/javax/sound/sampled/DataLine.html#available())()            Obtains the number of bytes of data currently available to the application for processing in the data line's internal buffer. |
| void | [**drain**](http://docs.google.com/javax/sound/sampled/DataLine.html#drain())()            Drains queued data from the line by continuing data I/O until the data line's internal buffer has been emptied. |
| void | [**flush**](http://docs.google.com/javax/sound/sampled/DataLine.html#flush())()            Flushes queued data from the line. |
| int | [**getBufferSize**](http://docs.google.com/javax/sound/sampled/DataLine.html#getBufferSize())()            Obtains the maximum number of bytes of data that will fit in the data line's internal buffer. |
| [AudioFormat](http://docs.google.com/javax/sound/sampled/AudioFormat.html) | [**getFormat**](http://docs.google.com/javax/sound/sampled/DataLine.html#getFormat())()            Obtains the current format (encoding, sample rate, number of channels, etc.) of the data line's audio data. |
| int | [**getFramePosition**](http://docs.google.com/javax/sound/sampled/DataLine.html#getFramePosition())()            Obtains the current position in the audio data, in sample frames. |
| float | [**getLevel**](http://docs.google.com/javax/sound/sampled/DataLine.html#getLevel())()            Obtains the current volume level for the line. |
| long | [**getLongFramePosition**](http://docs.google.com/javax/sound/sampled/DataLine.html#getLongFramePosition())()            Obtains the current position in the audio data, in sample frames. |
| long | [**getMicrosecondPosition**](http://docs.google.com/javax/sound/sampled/DataLine.html#getMicrosecondPosition())()            Obtains the current position in the audio data, in microseconds. |
| boolean | [**isActive**](http://docs.google.com/javax/sound/sampled/DataLine.html#isActive())()            Indicates whether the line is engaging in active I/O (such as playback or capture). |
| boolean | [**isRunning**](http://docs.google.com/javax/sound/sampled/DataLine.html#isRunning())()            Indicates whether the line is running. |
| void | [**start**](http://docs.google.com/javax/sound/sampled/DataLine.html#start())()            Allows a line to engage in data I/O. |
| void | [**stop**](http://docs.google.com/javax/sound/sampled/DataLine.html#stop())()            Stops the line. |

| **Methods inherited from interface javax.sound.sampled.**[**Line**](http://docs.google.com/javax/sound/sampled/Line.html) |
| --- |
| [addLineListener](http://docs.google.com/javax/sound/sampled/Line.html#addLineListener(javax.sound.sampled.LineListener)), [close](http://docs.google.com/javax/sound/sampled/Line.html#close()), [getControl](http://docs.google.com/javax/sound/sampled/Line.html#getControl(javax.sound.sampled.Control.Type)), [getControls](http://docs.google.com/javax/sound/sampled/Line.html#getControls()), [getLineInfo](http://docs.google.com/javax/sound/sampled/Line.html#getLineInfo()), [isControlSupported](http://docs.google.com/javax/sound/sampled/Line.html#isControlSupported(javax.sound.sampled.Control.Type)), [isOpen](http://docs.google.com/javax/sound/sampled/Line.html#isOpen()), [open](http://docs.google.com/javax/sound/sampled/Line.html#open()), [removeLineListener](http://docs.google.com/javax/sound/sampled/Line.html#removeLineListener(javax.sound.sampled.LineListener)) |

| **Method Detail** |
| --- |

### drain

void **drain**()

Drains queued data from the line by continuing data I/O until the data line's internal buffer has been emptied. This method blocks until the draining is complete. Because this is a blocking method, it should be used with care. If drain() is invoked on a stopped line that has data in its queue, the method will block until the line is running and the data queue becomes empty. If drain() is invoked by one thread, and another continues to fill the data queue, the operation will not complete. This method always returns when the data line is closed.

**See Also:**[flush()](http://docs.google.com/javax/sound/sampled/DataLine.html#flush())

### flush

void **flush**()

Flushes queued data from the line. The flushed data is discarded. In some cases, not all queued data can be discarded. For example, a mixer can flush data from the buffer for a specific input line, but any unplayed data already in the output buffer (the result of the mix) will still be played. You can invoke this method after pausing a line (the normal case) if you want to skip the "stale" data when you restart playback or capture. (It is legal to flush a line that is not stopped, but doing so on an active line is likely to cause a discontinuity in the data, resulting in a perceptible click.)

**See Also:**[stop()](http://docs.google.com/javax/sound/sampled/DataLine.html#stop()), [drain()](http://docs.google.com/javax/sound/sampled/DataLine.html#drain())

### start

void **start**()

Allows a line to engage in data I/O. If invoked on a line that is already running, this method does nothing. Unless the data in the buffer has been flushed, the line resumes I/O starting with the first frame that was unprocessed at the time the line was stopped. When audio capture or playback starts, a [START](http://docs.google.com/javax/sound/sampled/LineEvent.Type.html#START) event is generated.

**See Also:**[stop()](http://docs.google.com/javax/sound/sampled/DataLine.html#stop()), [isRunning()](http://docs.google.com/javax/sound/sampled/DataLine.html#isRunning()), [LineEvent](http://docs.google.com/javax/sound/sampled/LineEvent.html)

### stop

void **stop**()

Stops the line. A stopped line should cease I/O activity. If the line is open and running, however, it should retain the resources required to resume activity. A stopped line should retain any audio data in its buffer instead of discarding it, so that upon resumption the I/O can continue where it left off, if possible. (This doesn't guarantee that there will never be discontinuities beyond the current buffer, of course; if the stopped condition continues for too long, input or output samples might be dropped.) If desired, the retained data can be discarded by invoking the flush method. When audio capture or playback stops, a [STOP](http://docs.google.com/javax/sound/sampled/LineEvent.Type.html#STOP) event is generated.

**See Also:**[start()](http://docs.google.com/javax/sound/sampled/DataLine.html#start()), [isRunning()](http://docs.google.com/javax/sound/sampled/DataLine.html#isRunning()), [flush()](http://docs.google.com/javax/sound/sampled/DataLine.html#flush()), [LineEvent](http://docs.google.com/javax/sound/sampled/LineEvent.html)

### isRunning

boolean **isRunning**()

Indicates whether the line is running. The default is false. An open line begins running when the first data is presented in response to an invocation of the start method, and continues until presentation ceases in response to a call to stop or because playback completes.

**Returns:**true if the line is running, otherwise false**See Also:**[start()](http://docs.google.com/javax/sound/sampled/DataLine.html#start()), [stop()](http://docs.google.com/javax/sound/sampled/DataLine.html#stop())

### isActive

boolean **isActive**()

Indicates whether the line is engaging in active I/O (such as playback or capture). When an inactive line becomes active, it sends a [START](http://docs.google.com/javax/sound/sampled/LineEvent.Type.html#START) event to its listeners. Similarly, when an active line becomes inactive, it sends a [STOP](http://docs.google.com/javax/sound/sampled/LineEvent.Type.html#STOP) event.

**Returns:**true if the line is actively capturing or rendering sound, otherwise false**See Also:**[Line.isOpen()](http://docs.google.com/javax/sound/sampled/Line.html#isOpen()), [Line.addLineListener(javax.sound.sampled.LineListener)](http://docs.google.com/javax/sound/sampled/Line.html#addLineListener(javax.sound.sampled.LineListener)), [Line.removeLineListener(javax.sound.sampled.LineListener)](http://docs.google.com/javax/sound/sampled/Line.html#removeLineListener(javax.sound.sampled.LineListener)), [LineEvent](http://docs.google.com/javax/sound/sampled/LineEvent.html), [LineListener](http://docs.google.com/javax/sound/sampled/LineListener.html)

### getFormat

[AudioFormat](http://docs.google.com/javax/sound/sampled/AudioFormat.html) **getFormat**()

Obtains the current format (encoding, sample rate, number of channels, etc.) of the data line's audio data.

If the line is not open and has never been opened, it returns the default format. The default format is an implementation specific audio format, or, if the DataLine.Info object, which was used to retrieve this DataLine, specifies at least one fully qualified audio format, the last one will be used as the default format. Opening the line with a specific audio format (e.g. [SourceDataLine.open(AudioFormat)](http://docs.google.com/javax/sound/sampled/SourceDataLine.html#open(javax.sound.sampled.AudioFormat))) will override the default format.

**Returns:**current audio data format**See Also:**[AudioFormat](http://docs.google.com/javax/sound/sampled/AudioFormat.html)

### getBufferSize

int **getBufferSize**()

Obtains the maximum number of bytes of data that will fit in the data line's internal buffer. For a source data line, this is the size of the buffer to which data can be written. For a target data line, it is the size of the buffer from which data can be read. Note that the units used are bytes, but will always correspond to an integral number of sample frames of audio data.

**Returns:**the size of the buffer in bytes

### available

int **available**()

Obtains the number of bytes of data currently available to the application for processing in the data line's internal buffer. For a source data line, this is the amount of data that can be written to the buffer without blocking. For a target data line, this is the amount of data available to be read by the application. For a clip, this value is always 0 because the audio data is loaded into the buffer when the clip is opened, and persists without modification until the clip is closed.

Note that the units used are bytes, but will always correspond to an integral number of sample frames of audio data.

An application is guaranteed that a read or write operation of up to the number of bytes returned from available() will not block; however, there is no guarantee that attempts to read or write more data will block.

**Returns:**the amount of data available, in bytes

### getFramePosition

int **getFramePosition**()

Obtains the current position in the audio data, in sample frames. The frame position measures the number of sample frames captured by, or rendered from, the line since it was opened. This return value will wrap around after 2^31 frames. It is recommended to use getLongFramePosition instead.

**Returns:**the number of frames already processed since the line was opened**See Also:**[getLongFramePosition()](http://docs.google.com/javax/sound/sampled/DataLine.html#getLongFramePosition())

### getLongFramePosition

long **getLongFramePosition**()

Obtains the current position in the audio data, in sample frames. The frame position measures the number of sample frames captured by, or rendered from, the line since it was opened.

**Returns:**the number of frames already processed since the line was opened**Since:** 1.5

### getMicrosecondPosition

long **getMicrosecondPosition**()

Obtains the current position in the audio data, in microseconds. The microsecond position measures the time corresponding to the number of sample frames captured by, or rendered from, the line since it was opened. The level of precision is not guaranteed. For example, an implementation might calculate the microsecond position from the current frame position and the audio sample frame rate. The precision in microseconds would then be limited to the number of microseconds per sample frame.

**Returns:**the number of microseconds of data processed since the line was opened

### getLevel

float **getLevel**()

Obtains the current volume level for the line. This level is a measure of the signal's current amplitude, and should not be confused with the current setting of a gain control. The range is from 0.0 (silence) to 1.0 (maximum possible amplitude for the sound waveform). The units measure linear amplitude, not decibels.

**Returns:**the current amplitude of the signal in this line, or [AudioSystem.NOT\_SPECIFIED](http://docs.google.com/javax/sound/sampled/AudioSystem.html#NOT_SPECIFIED)

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/DataLine.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/javax/sound/sampled/Control.Type.html)   [**NEXT CLASS**](http://docs.google.com/javax/sound/sampled/DataLine.Info.html) | [**FRAMES**](http://docs.google.com/index.html?javax/sound/sampled/DataLine.html)    [**NO FRAMES**](http://docs.google.com/DataLine.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: [NESTED](#3znysh7) | FIELD | CONSTR | [METHOD](#2et92p0) | DETAIL: FIELD | CONSTR | [METHOD](#3dy6vkm) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).