

# Flash Lite 2.0: Sound for Nokia S60 and Series 40 Devices

Version 1.0; April 11, 2008

Flash Lite

**NOKIA**

Copyright ©, 2008 Nokia Corporation. All rights reserved.

Nokia and Forum Nokia are registered trademarks of Nokia Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

#### **Disclaimer**

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

#### **License**

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.

## Contents

<b>1</b>	<b>Introduction</b> .....	<b>6</b>
<b>2</b>	<b>Flash Lite 2.0 device sound</b> .....	<b>7</b>
2.1	Common device sound formats.....	7
2.2	Differences between S60 and Series 40.....	7
2.3	Testing MIDI sounds .....	8
2.3.1	S60 devices from Nokia.....	8
2.3.2	Series 40 devices from Nokia .....	8
2.4	Using the Sound class API.....	8
2.4.1	Creating an ActionScript identifier for a bundled MIDI sound .....	9
2.4.2	Playing a bundled MIDI file .....	10
2.4.3	Loading and playing an external device sound.....	10
2.4.4	Displaying a load progress message .....	11
2.4.5	Looping device sounds with onSoundComplete .....	12
2.4.6	Reading MP3 ID3 metadata .....	13
2.5	Detecting sound capabilities with the System.capabilities object .....	14
2.6	Working with bundled 3GP sounds.....	15
2.6.1	Importing 3GP sounds.....	15
2.6.2	Creating a Video object.....	17
2.6.3	Hiding the sound-only video object default image.....	17
2.6.4	Introduction to the Video Class API.....	18
2.7	Synchronizing device sound and animation.....	22
2.7.1	Using the _forceframerate global property.....	22
2.7.2	Working with sound interruption .....	22
<b>3</b>	<b>Flash Lite 2.0 native audio</b> .....	<b>23</b>
3.1	Creating Sound Objects for timeline sounds.....	23
3.2	Creating Sound Objects for library sounds.....	23
3.3	Understanding global sound objects.....	24
3.4	Playing and looping sound objects.....	24
3.5	Pausing sound objects.....	25
3.6	Setting volume and pan position .....	26
3.7	Guidelines for preparing native audio content.....	27
<b>4</b>	<b>Managing Flash Lite 2.0 memory usage</b> .....	<b>28</b>
<b>5</b>	<b>Terms and abbreviations</b> .....	<b>29</b>
<b>6</b>	<b>References</b> .....	<b>30</b>
<b>Appendix A</b>	<b>Sound-related System.capabilities</b> .....	<b>31</b>
<b>Appendix B</b>	<b>Sound Class API</b> .....	<b>33</b>

**About the author ..... 35**  
**Evaluate this resource..... 36**

## Change history

April 11, 2008	Version 1.0	Initial document release

## 1 Introduction

Flash Lite from Adobe is a mobile profile of the Flash runtime intended for mobile and handheld devices. Flash Lite can be an effective mobile platform for casual games, entertainment content, and mobile multimedia applications.

Nokia has licensed Flash Lite 2.0 from Adobe for its devices based on S60 3rd Edition, Feature Pack 1 and Series 40 3rd Edition, Feature Pack 2. Flash Lite 2.0 is based on the Flash 7 player for PC and includes features for device interaction and mobile-specific applications.

Flash Lite 2.0 adds new sound capabilities including a dedicated sound and video ActionScript API, the ability to load sounds stored locally on the device or over the network, and a means of locking the frame rate to synchronize animation with device sound formats.

Like Flash Lite 1.1, Flash Lite 2.0 has both a device sound and a native audio implementation. Flash Lite 2.0 for S60 supports both the device sound and native audio implementations. Flash Lite 2.0 for Series 40 supports only the device sound capabilities.

## 2 Flash Lite 2.0 device sound

The device sound implementation enables Flash Lite to play sound formats that are supported by a device's operating system. S60 and Series 40 mobile devices developed by Nokia support many types of sound formats, but the most common device sound formats that Flash Lite 2.0 supports are MIDI, MP3, AAC, and 3GP.

### 2.1 Common device sound formats

**MIDI** – A Music Instrument Digital Interface (MIDI) file contains music performance data such as the pitch, duration, time, volume, and timbre of each note in the composition. Unlike a recorded format such as WAV or MP3, MIDI contains no sound, only data to reproduce a recording when played through a mobile phone synthesizer. MIDI files can be very small in file size and more CPU efficient to play compared to recorded formats like MP3 and AAC.

**MP3** – MPEG 1 layer III (MP3) is a popular audio encoder for digital music on the Internet. MP3 is part of the MPEG-1 and MPEG-2 standard. MP3 can be an effective format for music, voice, and sound effects.

**AAC** – Advanced Audio Coding (AAC) is standardized in MPEG-2 and MPEG-4. Low Complexity AAC (AAC-LC), standardized in MPEG-2, generally offers higher quality at lower bit rates compared to the MP3 encoder. High Efficiency Advanced Audio Coding (HE-AAC), also known as aacPlus, offers higher quality at lower bit rates than AAC-LC. Nokia devices support both versions of AAC. AAC is an ideal format for music.

**AMR-NB** – AMR Narrow Band (NB) is a voice codec providing “toll” quality speech starting at 7.4 kbps, with near-toll quality at lower bit rates. AMR-NB supports a range of bit rates from 4.75 to 12.2 kbps. AMR is a monophonic format and does not support stereo output. AMR-NB is commonly used for voiceover sound tracks in 3GP video. AMR is well suited for voice content and low-fidelity sound effects.

**3GP** – 3GP is an audio/video format developed by the Third Generation Partnership Project. 3GP is a subset of the MPEG-4 video format and can contain AAC and AMR-NB audio streams.

### 2.2 Differences between S60 and Series 40

There are several differences between the Flash Lite 2.0 device sound implementation for Series 40 and S60. Flash Lite 2.0 for S60 can load and play MIDI, AAC, MP3, and 3GP sounds that are stored locally on the device or over the network. S60 can also play MIDI and 3GP files that are bundled within a SWF. Flash Lite 2.0 for Series 40 is limited to playing MIDI sounds that are stored locally or over the network and playing MIDI and 3GP sounds bundled within a Shock Wave File (SWF).

Use the Sound Class API to play bundled MIDI sounds or to load external sounds in other formats. To play bundled sounds in 3GP format use the Video Class API. Table 1 gives a comparison of the device sound capabilities of Flash Lite 2.0 for S60 and Series 40.

Platform	MIDI	MP3	AAC	3GP (aac/amr-nb)
S60 3rd Ed FP1	Bundled, local, network	Local, network	Local, network	Bundled, local, network
Series 40 3rd Ed FP2	Bundled, local, network	No	No	Bundled only

Table 1: Comparison of device sound formats supported by S60 and Series 40

**Note:** Flash Lite 2.0 for Series 40 and S60 is limited to playing one device sound at a time.

## 2.3 Testing MIDI sounds

Test a device sound directly on a device to verify that it plays as intended before integrating it with Flash Lite. The main concern is that the sound needs to project well through the external speaker of a device. Test at High and Low volume settings, through headphones, and in different locations to determine how it sounds against competing sounds from the environment.

Nokia SDK emulators for both Series 40 and S60 devices contain the MIDI synthesizer and sound banks installed on the corresponding devices. You can audition MIDI sounds by loading these sounds into the emulator. You can also test some formats, but not all recorded audio formats in the SDK. Visit <http://www.forum.nokia.com> for more information about the Nokia SDK device emulators.

### 2.3.1 S60 devices from Nokia

Install a MIDI file in the Simple folder within the Sounds folder. On the device, select the MIDI file from the Sound Clips category under Gallery.

Install recorded formats, such as AAC, MP3, or 3GP in the Digital folder within the Sounds folder. On the device, navigate to Gallery. Nokia S60 devices categorize AMR and WAV format sounds under the "Sound Clips" folder, MP3 and AAC files in the "Tracks" folder, and 3GP in the "Videos" folder.

### 2.3.2 Series 40 devices from Nokia

Transfer device sounds to the Ringing Tones folder within the Tones folder. On the device, play the selected sound from the Ringing Tones folder under the Tones folder of the Gallery.

## 2.4 Using the Sound class API

Flash Lite 2.0 has a dedicated Sound Class API for ActionScript control over sound playback. The Sound Class API supports playing MIDI sounds bundled within a SWF, loading and playing device sounds that are stored locally on the device or stored on the network, and getting information about the sound file.

The device sound implementation is limited to a subset of the full Sound Class API. Table 2 lists the Sound Class API methods relevant to interacting with device sounds.

Constructor	Comments
<code>Sound(target:MovieClip)</code>	Creates a new Sound object. The target argument is not required for device sounds.
Properties	
<code>id3</code>	Array listing values of ID3 1.0, 1.1 and 2.3, 2.4

	ID3 tags in MP3 files.
<b>Events</b>	
onID3	Invoked when ID3 data is available for an MP3 file.
onLoad	Invoked after a sound loads completely. Passes a Boolean value, representing success or failure of load, to an assigned function.
onSoundComplete	Invoked after a sound finishes playing.
<b>Methods</b>	
attachSound (idName:String)	Attaches the sound represented by the ActionScript identifier parameter in the Sound Properties window to a Sound object. MIDI sounds only.
getBytesLoaded ()	Returns number of bytes loaded for a sound.
getBytesTotal ()	Returns total file size in bytes of a sound.
loadSound (url:String, istreaming:Boolean)	Load a device sound. Flash Lite 2.0 does not support progressive load of device sound and ignores the isStreaming argument.
start (secondoffset:Number, loop:Number)	Starts playing the attached or loaded sound. Flash Lite ignores the secondoffset and loop arguments for device sounds.
stop ()	Stops the corresponding sound object.

Table 2: Overview of Sound Class API relevant to the Device sound implementation.

See Appendix B, “Sound Class API,” for a comparison of Sound Object API support for device sound and native audio implementations.

#### 2.4.1 Creating an ActionScript identifier for a bundled MIDI sound

An ActionScript identifier is the ActionScript reference for an asset in the FLA library window. In order to play a bundled MIDI sound using the Sound Class API, give the MIDI sound an ActionScript identifier.

**Note:** Review [Flash Lite 1.1: Sound for Nokia S60 and Series 40 Devices](#) [1] for details on how to bundle a MIDI file within a SWF.

Once the export settings for the proxy sound in the Sound Properties window have been configured, click the “Advanced” button at the bottom of the window to expand the window for more options to create an ActionScript identifier.

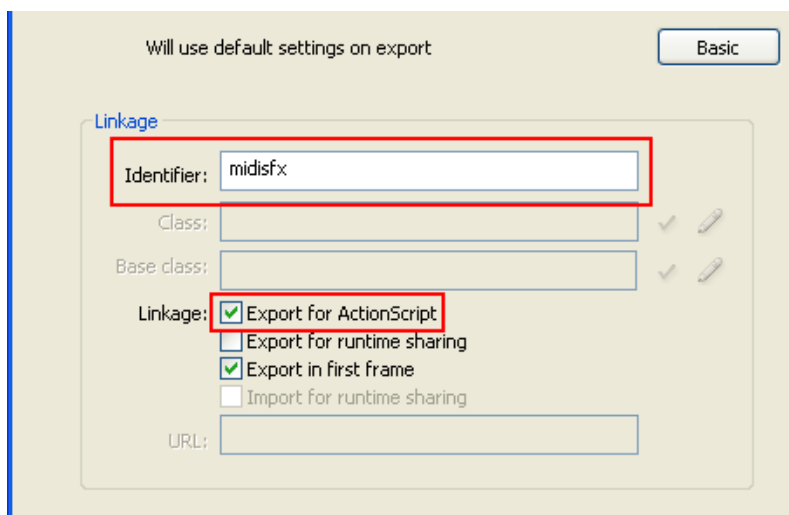


Figure 1: Assign an ActionScript Identifier to a proxy sound

Check the “Linkage: Export for ActionScript” check box and enter a name, with no spaces, in the Identifier box. Click OK. Also make sure the checkbox “Export in first frame” is checked so that the Flash IDE includes the sound in the SWF.

**Note:** Flash Lite 2.0 for both S60 and Series 40 only supports the MIDI format as a bundled device sound with proxy sounds.

#### 2.4.2 Playing a bundled MIDI file

The code in Example 1 demonstrates the process of creating an instance of the Sound Class, assigning a bundled MIDI sound to the instance and playing the MIDI sound. Pass the ActionScript identifier of the MIDI sound to the `attachSound()` method to assign the MIDI file to the Sound object. The `attachSound()` method connects a sound from the library to the sound object. To play the sound, call the `start()` method; use the `stop()` method to stop the sound.

```

/* create an instance of the Sound class */
var mysoundobject:Sound = new Sound();

/*
connect a sound from the library to the Sound Object.
Pass the ActionScript identifier of the proxy sound
to the attachSound() method
*/
mysoundobject.attachSound("midisfx");

/* play the sound object */
mysoundobject.start();
  
```

Example 1: Play a bundled MIDI file

**Note:** You can only attach bundled MIDI files that are in the same SWF as the sound object.

#### 2.4.3 Loading and playing an external device sound

You can also use the Sound Class API to load and play a device sound file that is stored locally on the mobile device or over the network. In this case use the `loadSound()` method to load the sound into the sound object.

Unlike the desktop PC version of Flash, Flash Lite 2.0 for Series 40 and S60 cannot play a device sound file as it loads. In the device sound implementation, Flash Lite 2.0 first loads the sound and then

passes the sound file to the device's OS for playback. Because sounds load asynchronously, use the `onLoad` event to execute code to play a sound after Flash Lite 2.0 completely loads the sound file.

Example 2 demonstrates how to assign a function literal to the `onLoad` event and process the Boolean value, representing success or failure to load, that the event passes to your function to play a sound immediately after it loads.

```
/* Create an instance of the Sound class */
var mysoundobject:Sound = new Sound();

/*
Set up the onLoad event to play a sound immediately after it loads.
The onLoad event passes a Boolean value, representing success or failure
when loading sound, to the assigned function literal.
In this example the variable "success" is either true or false.
If success is true then play the sound.
*/
mysoundobject.onLoad = function(success){if(success){this.start()}};

/*
Load a midi file named test.mid stored locally on the phone and
located in the same directory as the SWF.
Alternatively, you can replace the argument with a valid url to load a
MIDI file over the network.
*/
mysoundobject.loadSound("test.mid");
```

Example 2: Using `loadSound()` method and the `onLoad` event

**Note:** Flash Lite 2.0 for Series 40 only supports loading MIDI files. Flash Lite 2.0 for S60 can load additional formats such as MP3, AAC, and 3GP.

**Note:** Device Central relies upon QuickTime to play device sound formats, and can audition sounds in MIDI, MP3, AAC, and 3GP (both AAC and AMR formats). It does not support direct playback of all formats that a mobile device may support such as AMR-NB or AMR-WB.

#### 2.4.4 Displaying a load progress message

In cases where a file is being loaded over the network, it is useful to provide a load progress message so the user knows the status of the application. A common way to create a load progress message is to use the Flash Lite 2.0 `onEnterFrame` event as a loop that repeatedly calculates the percent of sound loaded as the file downloads. Start the `onEnterFrame` loop by assigning a function name or function literal to the event. Stop the `onEnterFrame` loop by assigning it to a null value.

Example 3 has three functions to enable the load progress message. The `loadMySound` function passes a URL to the `loadSound()` method to start loading a sound over the network. It also assigns the `onEnterFrame` loop event to a function named `displayPercentLoaded`. The `displayPercentLoaded` function uses the Sound Class API `getBytesLoaded()` and `getBytesTotal()` methods to calculate the percent of the sound that has loaded and print this value to a predefined text box on the screen. The `onLoad` event has a function literal that plays the sound after it finishes loading and code that terminates the `onEnterFrame` loop.

```
/* create a new sound object */
var mysoundobject:Sound = new Sound();
```

```

/*
Set up onLoad event to play a sound after it has loaded,
and terminate the onEnterFrame loop, because it is no longer needed.
*/
mysoundobject.onLoad = function(success){
    if(success){ // play sound on successful load
        mysoundobject.start();
    } else { // print failure message to text box
        textbox.text = "Failed to load sound.";
    }
    this.onEnterFrame = null; // terminate the loop
}

/*
Calculate the percent of sound that has loaded.
Print this to pre-defined text box
*/
function displayPercentLoaded(){
    // calculate percent of sound loaded, convert decimal to percent
    var loaded:Number = mysoundobject.getBytesLoaded();
    var total:Number = mysoundobject.getBytesTotal();
    var percent:Number = Math.floor(loaded/total * 100);

    // loaded or total are 0 or undefined until sound begins to load
    // set percent to 0 during this time
    if(isNaN(percent) == true) { percent = 0; }

    // print message to a pre defined text box
    textbox.text = "Loading: " + percent + "%";
}

/*
This function passes the url of a sound to the loadSound method,
and starts the onEnterFrame loop to create the load progress message.
*/
function loadMySound(url:String){
    this.onEnterFrame = null; // stop any previous loop
    mysoundobject.loadSound(url); // start loading a sound

    // assign function object name to the onEnterFrame loop
    this.onEnterFrame = displayPercentLoaded;
}

/* start loading a sound and display load progress message. */
loadMySound("valid url to a supported device sound");

```

Example 3: Displaying a load progress message

#### 2.4.5 Looping device sounds with onSoundComplete

The Flash Lite 2.0 Sound class API does not have a built-in means to seamlessly loop device sounds. However, it is possible to repeatedly play a device sound with the `onSoundComplete` event. Some mobile phones loop device sounds with a noticeable gap between the end and restart of a sound. This technique for sound looping is most effective for ambient tracks that do not need rhythmically precise looping.

The `onSoundComplete` event can be used to restart a sound after it stops. Example 4 contains two functions that demonstrate how to set up a repeating device sound. The `startLoop` function assigns the `start()` method to the `onSoundComplete` event so the sound will restart after it finishes playing. The `stopLoop` function stops the sound by calling the `stop()` method and terminates the `onSoundComplete` event by setting it to null.

```

/* create a sound object */
var mysoundobject:Sound = new Sound();

```

```

/* restart the sound after it finishes */
function startLoop(){
    mysoundobject.start(); // start playing the sound
    // set up event to restart sound at end
    mysoundobject.onSoundComplete = function(){this.start()}
}

/* stop the sound and the onSoundComplete loop */
function stopLoop(){
    mysoundobject.stop(); // stop the sound
    mysoundobject.onSoundComplete = null; // stop the event
}

```

Example 4: Looping device sounds with onSoundComplete

## 2.4.6 Reading MP3 ID3 metadata

The Flash Lite 2.0 player for S60 supports the `id3` array and the `onID3` event for reading ID3 v1.0 and v2.0 metadata in MP3 files. Flash Lite 2.0 executes the `onID3` event when it has loaded enough of the sound file to access the ID3 data field. Flash Lite 2.0 then parses the ID3 data from the MP3 and stores it as properties of the `id3` associative array.

Flash Lite 2.0 creates a property in the `id3` array named for each ID3 tag name from the ID3 data field, and stores the value of the ID3 tag as the value corresponding property. The data in the `id3` array can be accessed anytime after Flash Lite 2.0 executes the `onID3` event.

Table 3 is an example of the types of ID3 tags an MP3 may contain and how Flash Lite 2.0 represents the ID3 tags as properties of the `id3` array.

ID3 Tag Name	ID3 Tag Value	Sound Class id3 property
TALB:	A Jazz Album	<code>mysoundobject.id3.TALB</code>
TPE1:	Jazz Recording Artist	<code>mysoundobject.id3.TPE1</code>
TRCK:	01	<code>mysoundobject.id3.TRCK</code>
TCON:	General Jazz	<code>mysoundobject.id3.TCON</code>
TIT2:	Well You Needn't	<code>mysoundobject.id3.TIT2</code>
genre:	General Jazz	<code>mysoundobject.id3.genre</code>
track	01	<code>mysoundobject.id3.track</code>
album	A Jazz Album	<code>mysoundobject.id3.album</code>
artist	Jazz Recording Artist	<code>mysoundobject.id3.artist</code>
songname	Well You Needn't	<code>mysoundobject.id3.songname</code>

Table 3: MP3 ID3 tags represented as a property of the Sound Class id3 array

Example 5 demonstrates how to access `id3` metadata using the `onID3` event and the `id3` array.

```

/*
print ID3 data to a predefined text box
*/
function printID3(){
    // loop through associative array, print ID3 value to a textfield
    for (var prop in mysoundobject.id3) {
        textbox.text += prop + ": " + mysoundobject.id3[prop] + "\n";
    }
}

```

```

    }
}

/* create a sound object */
mysoundobject:Sound = new Sound();

/* load a MP3 */
mysoundobject.loadSound("urlto.mp3");

/*
set the id3ready variable to true after the onID3 event.
call the printID3 function from the onID3 event,
so that Flash Lite 2.0 prints the data as soon as it is ready.
*/
var id3ready:Boolean = false;
mysoundobject.onID3 = function() { id3ready = true; printID3(); }

/*
get the id3 data at a later time from a button press or other event
*/
if(id3ready == true) {printID3();}

/* read a specific ID3 tag as a property of the array */
textbox.text = mysoundobject.id3.songname;

```

Example 5: Reading MP3 ID3 metadata

## 2.5 Detecting sound capabilities with the System.capabilities object

Flash Lite 2.0 extends the `System.capabilities` class to return information about the capabilities of a device. Included in the Flash Lite 2.0 `System.capabilities` properties are sound-specific capabilities. One of the properties is the `audioMIMETypes` array, which can be used to get a list of device sound formats that Flash Lite 2.0 should support.

Loop through the `audioMIMETypes` array to view a complete list of the supported device sound formats for Flash Lite 2.0 on a given mobile device.

```

/*
loop through the audioMIMETypes array and print each mime type to a text
box
*/
for(var i:Number=0; i<System.capabilities.audioMIMETypes.length; i++){
    textbox.text += System.capabilities.audioMIMETypes[i] + "\n";
}

```

Example 6: Display all supported sound format mimetypes

Ideally, the `audioMIMETypes` array can also be used to determine whether Flash Lite 2.0 for a given mobile phone supports a device sound format before attempting to load and play the sound.

```

/*
Pass a mime type to this function
Return true or false depending whether
a given mime type is in the audioMIMETypes array.
Change the audioMIMETypes to MIMETYPE to check support
for any external image, sound or video format.
*/
function hasAudioMIMETYPE(mimeType:String) {
    // convert the audioMIMETypes array to a string
    var mimeList:String = String(System.capabilities.audioMIMETypes);

    // return false if the position of the mimeType substring
    // is not present.
    return (mimeList.indexOf(mimeType) > -1);
}

```

```

/*create a sound object */
var mysoundobject:Sound = new Sound();

/* define the onLoad event */
mysoundobject.onLoad = function(success) {if (success) {this.start()}}

/*
verify that Flash Lite 2.0 supports MP3 sound before attempting to load
the sound.
If it does not support MP3 sounds, then print a message to a predefined
text box.
*/
if (hasAudioMIMEType("audio/mp3")) {
    mysoundobject.loadSound("locallystored.mp3");
} else {
    textbox.text = "This phone does not support the MP3 format";
}

```

Example 7: Detect support for a given format

See Appendix A for a complete list of `Sound.capabilities` properties.

**Caution:** Flash Lite 2.0 for some Nokia devices does not support all of the sound formats listed in the `audioMIMETypes` array. For example, Flash Lite 2.0 for the Nokia 6290 device lists AMR-WB and AMR-NB in the `audioMIMEType` array, but will not load sounds in these formats and displays the player error message “Flash content error: 7” for corrupt or unsupported sound format.

## 2.6 Working with bundled 3GP sounds

In addition to supporting bundled MIDI sounds, Flash Lite 2.0 for Series 40 and S60 can also play bundled 3GP files that contain sound compressed with AAC or AMR-NB. 3GP is a useful option for adding voice recordings, real music recordings, and sound effects to a Flash application or game. Flash Lite 2.0 requires the Video Class API to play a bundled 3GP sound.

There are two situations where you may consider using bundled 3GP sound assets. The first case is to support recorded sound content for the Series 40 Flash Lite 2.0 player. Flash Lite 2.0 for Series 40 cannot load external sounds other than MIDI and cannot load external 3GP through the Video Class API. If you intend to include voice recordings or real music recordings for Series 40 Flash Lite content, then you must use bundled 3GP sounds and interact with them using the Video Class API.

**Note:** Flash Lite 2.0 for Series 40 and S60 can load external SWF files containing 3GP.

The second case is to enable Pause, Resume, Fast Forward, and Rewind features for 3GP sound playback in the S60 Flash Lite 2.0 player. These features are supported in the Video Class API but not the Sound Class API for device sounds. Note that Flash Lite 2.0 for Series 40 does not support the Video Class API `seek()` method required for fast forward and rewind features.

### 2.6.1 Importing 3GP sounds

Importing and bundling sound-only 3GP files is a different process compared to importing and bundling MIDI files.

1. Select New Video from the Library menu.

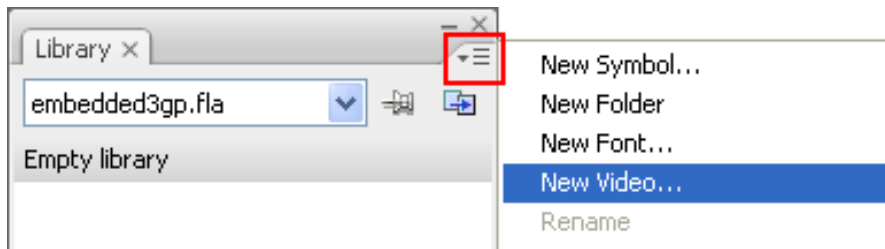


Figure 2: New video option from Library menu

2. In the Video Properties Dialog, check the bundle for SWF check box.

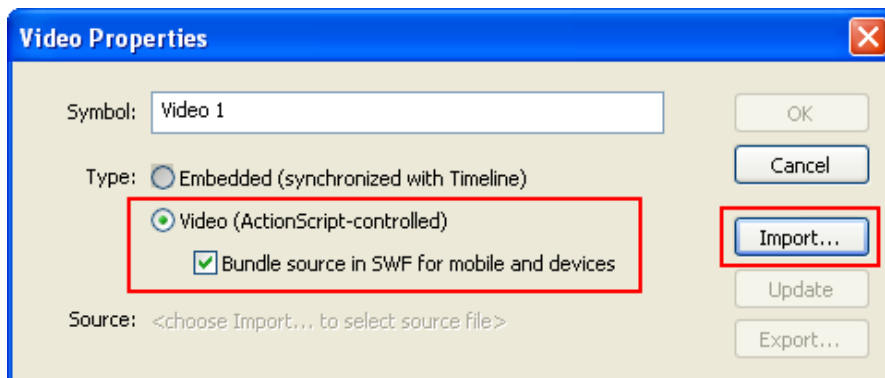


Figure 3: Video Properties dialog box

3. Click the Import button.
4. Navigate to the 3GP video file you intend to use.
5. Click OK in the file browser dialog.
6. Give the asset a name in the Symbol box.
7. Click OK to close the Video Properties Dialog box.

The video should now appear as an asset in the library window.

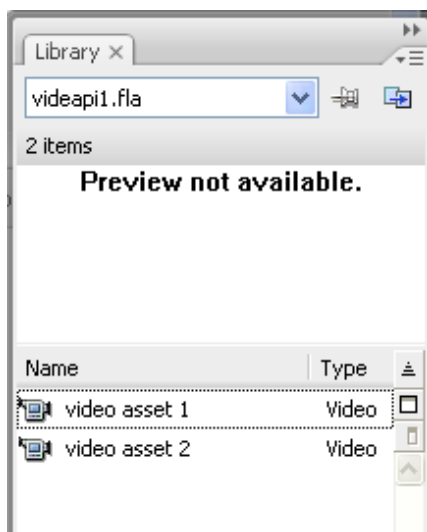


Figure 4: Video asset icons display in the Library window

## 2.6.2 Creating a Video object

Unlike the Sound Class API, you do not use ActionScript to create an instance of the Video class. To create a video object, you must manually place a video asset from the library on the stage.

1. Drag the video asset onto the stage. The video object displays as a letterbox placeholder on the stage.

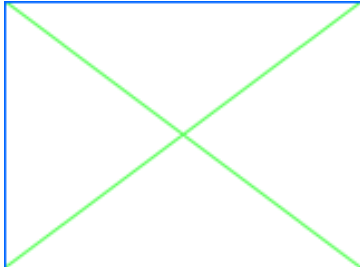


Figure 5: Video object letter box image

2. Select the video object on the stage and give the video object an ActionScript instance name in the inspector panel.



Figure 6: Video object instance name in properties inspector

Refer to this instance name when using the Video Class API to control the video.

## 2.6.3 Hiding the sound-only video object default image

Some Nokia mobile devices, like the Nokia 5300 XpressMusic device, will not play 3GP files without a video channel. For compatibility with all Nokia Flash Lite 2.0 devices, make sure to include a video channel when authoring a sound-only 3GP.

Nokia devices will display a sound-only 3GP video object containing a video channel with no image, as a black rectangle with a red text message: "Input file has no video stream." For sound-only 3GP content, you will most likely want to minimize or hide this default image.

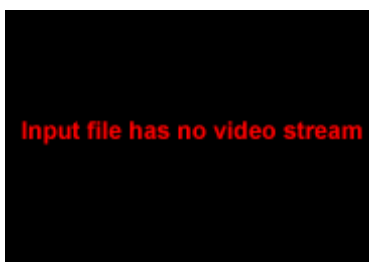


Figure 7: Sound only 3GP displays with default image

The normal means of hiding Flash Lite elements do not apply to video objects. The device video player displays 3GP video in a layer on top of all other Flash content. It is not possible to obscure the 3GP video with graphics in a SWF, or change its visibility with the `_visible` ActionScript property. Also, the video object must be in the visible portion of the stage to play. Positioning the video off screen is not an option, because some devices will not play the 3GP file if it is located off screen.

To minimize this default image, resize the video object to a 1 x 1 pixel. You can also create a 1-frame image for the video that has a color matching a graphic element in the SWF. To hide the video object, position the video in the Flash content on top of the SWF element so that the video object blends into the color of the element.

## 2.6.4 Introduction to the Video Class API

The Video Class API has several methods for controlling playback of sound and offers several features that are different than the Sound Class API. Table 4 gives an overview of the Video Class API and the features that Flash Lite 2.0 Series 40 and S60 support.

Method	Description	Series 40	S60
<code>close()</code>	Stops playback of the 3GP sound, frees the memory associated with this Video object, and clears the video area onscreen.	Yes	Yes
<code>loop(loopState:Boolean)</code>	True enables repeating playback of 3GP file. False disables repeating playback.	Yes	Yes
<code>mute(muteState:Boolean)</code>	Toggle sound volume on and off. Passing true turns on sound, false mutes sound.	Yes	Yes
<code>pause()</code>	Pauses playback of the video at its current position.	Yes	Yes
<code>play(source:String)</code>	Opens a video source and begins playback of a video. Source can be a <code>symbol://id</code> to play a bundled video, file path to a locally stored 3GP, or network 3GP with <code>http://</code> or <code>rtsp://</code> protocols.	Support for bundled 3GP only.	Support for bundled, locally stored, network or RTSP streamed 3GP.
<code>resume()</code>	Resumes playback of the video. If <code>pause()</code> was previously called, playback begins from the current position. If <code>stop()</code> was previously called, playback begins from the first frame.	Yes	Yes
<code>seek(seconds:Number)</code>	Start playback at seconds position passed. Seconds argument is a whole number.	No	Yes

<code>stop()</code>	Stops playback of the video. A subsequent call to <code>resume()</code> resumes playback from the beginning of the video.	Yes	Yes
<b>Event</b>			
<code>onStatus</code>	Returns an object with a property named <code>level</code> , indicating status of playback. A value of 0 for <code>level</code> indicates there is an error. Use the code property to determine the error message.	No	Yes  <code>level = 1</code> corresponds to video playback complete.

Table 4: Overview of Flash Lite 2.0 Video Class API

**Tip:** Use the `MovieClipLoader` class methods to generate a load progress message when loading external SWF containing bundled 3GP.

#### 2.6.4.1 Using the play method

Refer to the Video Object instance name that was entered in the property inspector.

Include the following ActionScript in the same frame or a frame after the Video Object on the stage.

```
/*
automatically play a bundled 3GP sound
when the playhead reaches this frame
*/
video_object.play();
```

In addition to creating a video object on the stage, you can also play each 3GP sound in the library from a single video object. This second option requires that each 3GP video asset be given an ActionScript identifier from the Properties option in the Library menu.

1. Select the video asset in the Library window.
2. Select Properties from the Library window menu.
3. In the Video properties dialog box, check "Export for ActionScript."

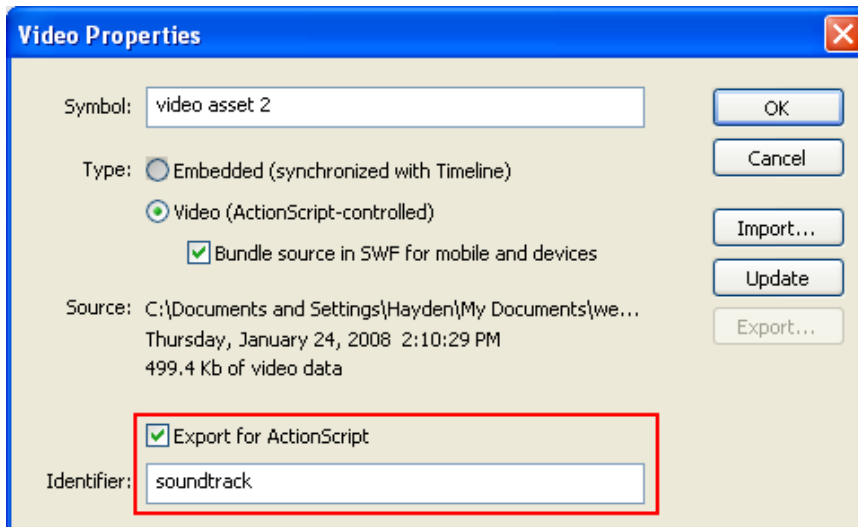


Figure 8: Assign an ActionScript identifier to 3GP video library asset

4. In the identifier box, type the ActionScript identifier you want to use for this asset. The identifier name should not include spaces.

To play a bundled 3GP from its identifier, pass the ActionScript identifier to the `play()` method using the `symbol://` protocol. This protocol indicates that the source is a bundled 3GP sound.

```
/* select and play a bundled 3GP sound from its identifier */
video_object.play("symbol://soundtrack");
```

**Tip:** To conserve memory usage, create the initial video object using a very short duration silent bundled 3GP sound or video. Then load other 3GP content as needed.

Flash Lite 2.0 for Series 40 gives focus to the current video object. To play another video object or attach a new 3GP sound to the current video object, you must first close the current video object with the `close()` method, then attempt to play another video object or new 3GP sound. This is not necessary for the S60 Flash Lite 2.0 player.

```
/*
play a different video object for the series 40 Flash Lite 2.0
*/
video_object1.close(); // first close the current video object
video_object2.play(); // play a different video object

/*
play a different bundled 3GP sound for the series 40 Flash Lite 2.0
*/
video_object1.close(); // clear the current 3GP sound
video_object1.play("symbol://soundtrack2"); // play a new 3GP sound
```

Example 8: Close video object before attempting to play a different video

For the Series 40 Flash Lite 2.0 player, you must also close a Video object before attempting to play a Sound object.

**Note:** You can only access ActionScript identifiers for 3GP that are bundled in the same SWF as the video object.

Flash Lite 2.0 for S60 can also load external 3GP files that are stored locally on the device, or on the network through the `http://` protocol or `rtsp://` streaming media protocol. Note that Flash Lite 2.0 for Series 40 only supports playing bundled 3GP, not loading of external 3GP.

```

/*
play a locally stored 3GP sound on a S60 device
in the same directory as the SWF
*/
video_object.play("sound.3gp");

```

#### 2.6.4.2 Using the loop method

Both Series 40 and S60 Flash Lite 2.0 players support the loop method, which enables Flash Lite 2.0 to repeatedly play a 3GP sound. To enable looping playback, pass a value of `true` to the loop method. To disable looping, pass a value of `false` to the method.

```

video_object.loop(true); // enable looping.
video_object.loop(false); // disable looping

```

You can change the loop state while the 3GP is stopped or paused, or during playback. Note that the `onStatus` event does not execute while the `loop()` method has a value of `true`.

On some Nokia devices, 3GP looped playback is not seamless, and there is a noticeable gap during repeat. For this reason, use sound tracks that do not require rhythmically accurate loops.

#### 2.6.4.3 Using the seek method

The `seek()` method enables Flash Lite 2.0 for S60 to start a 3GP sound at a specified position measured in seconds. You can use the `seek()` method to support fast forward, rewind, or navigate through chapters of content within a 3GP sound. The argument for this method is a whole Number representing seconds. Note that Flash Lite 2.0 for Series 40 does not support the `seek()` method.

```

/* start playing the 3GP sound 10 seconds from the beginning */
video_object.seek(10);

```

You can use the `seek()` method to change to a new position during playback. You can also set the position of the 3GP sound, while it is stopped or paused, and call the `resume()` method to start playing at the new position.

```

/*
set start position at 10 seconds into the sound, then play the sound
*/
video_object.pause(); // pause the 3GP sound
video_object.seek(10); // set the sound position to 10 seconds
video_object.resume(); // resume playback at the 10 second position

```

#### 2.6.4.4 Using the onStatus event

For the S60 Flash Lite 2.0 player, use the `onStatus` event to determine when the video player finishes playing a 3GP file, much like the Sound Class API `onSoundComplete` event.

The `onStatus` event returns an object with a property named `level` and a property named `code`. The S60 Flash Lite 2.0 player returns a value of 1 for the `level` property when the video player reaches the end of the 3GP. A `level` value of 0 or less indicates an error. The `code` property of the object contains the error message.

```

/*
execute ActionScript when video player finishes playing 3GP
*/
video_object.onStatus = function(video_status:Object){

```

```

if(video_status.level > 0){ // sound is complete
    textbox.text = "3GP sound complete";
} else { // there is an error
    textbox.text = video_status.code;
}
}

```

## 2.7 Synchronizing device sound and animation

Flash Lite 2.0 introduces a new global property called `_forceframerate` to aid in synchronizing animation and device sounds. This property gives you the option of having Flash Lite 2.0 render graphics in sync with device sounds, in much the same manner as Flash Lite renders graphics in sync with stream sync sounds in the native audio implementation.

The normal graphic rendering behavior of Flash is to give priority to drawing all graphic elements on the stage before moving to the next frame. If the graphics for a given frame are more complex than others, Flash will take longer to draw this frame, consequently slowing down the frame rate. This frame drawing behavior can lead to loss of synchronization because sound always plays at the same rate, while the animation frame rate can vary.

### 2.7.1 Using the `_forceframerate` global property

If you set the `_forceframerate` to `true`, Flash Lite will give priority to playing frames at or very close to the specified frame rate and not draw graphics that it cannot render in time. This helps to keep animation at the same speed and synchronize device sound.

**Note:** Forcing the frame rate may lead to less smooth animation for complex graphics because Flash Lite is not drawing all frames in the animation.

Setting `_forceframerate` to `false`, restores Flash Lite graphic rendering to its default behavior of varying frame rates to draw each frame regardless of frame rate.

```

_forceframerate = true; // establish a consistent frame rate
_forceframerate = false; // allow Flash Lite to render all frames

```

### 2.7.2 Working with sound interruption

Playing a device sound tends to pause Flash Lite 2.0 ActionScript and animation, which can make animation less smooth or interrupt the flow of game play. The amount of time that Flash Lite pauses animation and ActionScript when starting a sound, varies between mobile devices and also by the type of sound format.

For example, Flash Lite 2.0 for the Nokia 5300 XpressMusic device tends to introduce a 100 ms or less pause when starting a bundled MIDI or bundled 3GP sounds. Flash Lite 2.0 for the Nokia 6290 mobile device has a longer pause time, up to 500 ms, when starting bundled 3GP sounds, but similar pause as the Nokia 5300 XpressMusic device for bundled MIDI or external MP3 loaded into a sound object. It is important to evaluate the impact that sound interruption has on a given presentation for all target devices.

Refer to the Forum Nokia document [Flash Lite 1.1: Sound for Nokia S60 and Series 40 Devices](#) [1] for more information about managing pauses in ActionScript and animation that is introduced by playing device sounds.

### 3 Flash Lite 2.0 native audio

In the native audio implementation, Flash Lite 2.0 for S60 plays sound through its built-in audio subsystem. The features of native audio are similar to the desktop PC Flash player with some important differences. The Flash Lite 2.0 Sound Class for native audio does not support the `loadSound()` method, `onLoad` event, `onID3` event, or `id3` array. Flash Lite 2.0 does not support the “speech” codec option listed in the compression menu of the sound properties window.

Unlike the Flash Lite 1.1 player, the Flash Lite 2.0 native audio implementation does support progressive download of sounds set to stream sync and supports the `_soundbuftime` global property for setting the buffer size of progressively loaded sounds.

Use the Sound Class API with the native audio implementation to play 4 to 8 sounds simultaneously and control each sound’s volume and pan position independently.

There are two ways to connect Flash Lite 2.0 native audio assets to Sound Objects, and each approach offers a different set of features. One approach uses sounds placed in a frame of a movie clip, sometimes referred to as a “timeline” sound(s). The other approach uses sounds stored in the library rather than in a frame of a movie clip.

**Note:** Flash Lite 2.0 for S60 supports the native audio implementation but Series 40 does not.

#### 3.1 Creating Sound Objects for timeline sounds

To create a Sound Object that controls one or more sounds in frames of a movie clip, pass the movie clip object reference of the movie clip containing the sound(s) that you want to control, to the Sound constructor function. Note that the target path is an object not a string.

```
/*
create a sound object to control a sound in a frame
of the soundmc movie clip object
*/
var mysoundobject:Sound = new Sound(_root.soundmc);
```

The Flash Lite 2.0 Sound Class API is limited to executing `get/setVolume()`, `get/setPan()`, `get/setTransform()`, and the `stop()` methods for timeline sounds.

Review the document [Flash Lite 1.1: Sound for Nokia S60 and Series 40 Devices](#) [1] for more information about the capabilities of timeline sounds.

**Caution:** A bug in Flash Lite 2.0 prevents the player from correctly interpreting Sound Edit window volume point positions or start and stop positions for timeline sounds.

#### 3.2 Creating Sound Objects for library sounds

To create a sound object to control a sound that is stored in the library, pass the ActionScript identifier of the sound to the `attachSound()` method. For more information about ActionScript identifiers and the `attachSound()` method see Section 2.4.1, “Creating an ActionScript identifier for a bundled MIDI sound.”

If you intend for a sound object to control a unique sound, you should always pass a unique movie clip object reference to the Sound constructor function. In the case of sounds stored in the library, you can associate the Sound object with an empty movie clip on the stage. The following example dynamically creates an empty movie clip called `sndmc1`, using the `createEmptyMovieClip()` method, and associates it with the Sound object, then attaches the ActionScript identifier of a sound

asset in the library. Read the Flash IDE Help documentation for more information about the `createEmptyMovieClip()` method.

```
/*
Create a sound object and dynamically create an empty movie clip
associated with the sound object.
*/
var sndlmc:MovieClip =
this.createEmptyMovieClip("sndlmc",this.getNextHighestDepth());
var mysoundobject:Sound = new Sound(sndlmc);
mysoundobject.attachSound("soundtrack");
```

Example 9: Dynamically create an empty movie clip and associate it with a Sound Object

Unlike sounds in a frame on the timeline, you cannot manipulate a library sound using the sound edit window or the sound inspector panel.

### 3.3 Understanding global sound objects

If you do not pass a movie clip object reference to the `Sound` constructor function, then the `Sound` object will control all sounds in the SWF. Developers often refer to this type of sound object as a “global” sound object. Global sound objects are useful for controlling the volume of all native audio assets.

```
/* create global sound object to act as a master volume control */
var allSounds:Sound = new Sound(); // no associated movie clip
allSounds.setVolume(0); // mute all native audio sounds
```

### 3.4 Playing and looping sound objects

Unlike the device sound implementation, the native audio implementation for the S60 Flash Lite 2.0 player supports the `secondoffset` and `loop` arguments of the `start()` method. To play a sound one time with no loop or offset, call the `start()` method with no arguments. You can also set the number of loops and the starting position of the sound by passing values to the corresponding arguments.

```
/* play a sound one time from the beginning */
mysoundobject.start();

/* play a sound starting at the 1.5 second position */
mysoundobject.start(1.5);

/* play a sound that loops 3 times, from the beginning */
mysoundobject.start(0,3);
```

To stop a sound object, call the `stop()` method.

```
/* stop a sound object */
mysoundobject.stop();
```

You can also stop a sound attached to another sound object by passing the ActionScript identifier of the sound to the `stop()` method.

```
/* stop a sound attached to another sound object */
mysoundobject.stop("soundtrack2");
```

Library sounds attached to a sound object always play with the event sync behavior, which means that the sound can overlap itself. To prevent unintended sound overlap you should explicitly stop a sound before restarting it.

```
/* stop before restarting */
```

```
mysoundobject.stop(); // stop previous instance of this sound
mysoundobject.start(0,3); // restart the sound with no overlap
```

**Tip:** Always import uncompressed WAV or AIFF files for loops and use ADPCM or RAW compression to ensure that the sounds loop seamlessly.

### 3.5 Pausing sound objects

There is no built-in method for pausing a sound object, but it is possible to create this feature by using the Sound Class API `position` property and the offset argument of the `start()` method.

For a pause behavior, use the `stop()` method to stop playback; the `position` property retains the millisecond position where Flash Lite 2.0 stopped the sound.

The `playSound` function below defaults to resuming playback at the value of the `position` property, which is the position of the sound when Flash Lite executes the `stop()` method, or at the millisecond position passed to the `mssndpos` argument.

An awkward problem arises when a user resumes playback very near the end of a song, and only hears a brief sound or silence. To address this aesthetic issue, the `playSound` function restarts the sound at the beginning if the current position of the sound is within 500 milliseconds of the end.

Note that the `secondsoffset` argument of the `start` method is a value in seconds and that the `position` property is a value in milliseconds. The `playSound` function converts the `mssndpos` variable, which is a value in milliseconds, to seconds before passing it to the `start()` method.

```
/*
mssndpos is the sound position in milliseconds
default to the value of the position property, if no position passed
*/
function playSound(mssndpos:Number) {
    // default to position property if no value passed to function
    if(mssndpos == undefined){pos = mysoundobject.position;}

    mysoundobject.stop(); // prevent sound overlap

    // avoid starting near end
    if(mssndpos > mysoundobject.duration - 500){
        mysoundobject.start(0); // start at beginning
    } else {
        // convert milliseconds to seconds
        // resume at last paused position
        mysoundobject.start(mssndpos/1000);
    }
}
```

Example 10: Function to resume playback at last stopped position

The `playSound` function makes use of the Sound Class API `duration` property. Flash Lite 2.0 does not establish a value for the `duration` property until after the first time it plays a sound, which may be a problem if you need to know the value before playing a sound. To work around this problem, silently play and stop the sound object to establish a value for the sound `duration` property, and then access the value of the `duration` property.

```
/*
duration is 0 until Flash Lite 2.0 plays the sound object
start and stop sound silently to set duration property
*/
mysoundobject.setVolume(0);
mysoundobject.start();
mysoundobject.stop();
```

```
mysoundobject.setVolume(100);
textbox.text = mysoundobject.duration; // print actual duration
```

Example 11: Play a sound silently to establish a value for the duration property

### 3.6 Setting volume and pan position

Flash Lite 2.0 for S60 supports the `setVolume()` and `setPan()` methods to change sound volume and pan, and `getVolume()` and `getPan()` to return the current volume and pan position for native audio sound assets in a sound object. These methods cannot be used to alter volume or pan position of device sounds.

The `setVolume()` method takes a range of values from 0 (sound muted) to 100 (full volume). In practice, the `setVolume()` method can accept values larger than 100, which can over-amplify sounds, and will treat values less than 0 as absolute values, which also increases volume. To ensure proper sound quality and to avoid speaker problems, make sure the input values are in a range of 0 to 100.

```
/* completely silence a sound */
mysoundobject.setVolume(0);

/* restore full volume */
mysoundobject.setVolume(100);

/* limit the volume to a maximum of 100 and minimum of 0 */
if(mysoundobject.getVolume() > 100){mysoundobject.setVolume(100);}
if(mysoundobject.getVolume() < 0){mysoundobject.setVolume(0);}
```

Use the `setPan()` method to reposition a monophonic sound in the left or right speaker, or alter the left/right speaker balance of a stereophonic sound. The `setPan()` limits input to a range of -100 (left speaker only) to 100 (right speaker only), with 0 as the center position for monophonic sounds and equal left/right balance for stereo sound.

```
/* pan a monophonic sound to the left speaker */
mysoundobject.setPan(-100);

/* pan a monophonic sound to the right speaker */
mysoundobject.setPan(100);
```

The `getPan()` method returns the current pan position, which is a value between -100 and 100.

**Note:** Only change monophonic sound panning and stereo channel balance for content intended for headphone listening because not all S60 devices from Nokia support stereo separation and panning through the external speaker.

**Note:** Flash Lite 2.0 for some S60 devices from Nokia will introduce a noticeable delay between the time ActionScript executes the `setVolume()` or `setPan()` method, and the audible change in the sound volume or pan position.

Flash Lite 2.0 for S60 also supports the `getTransform()` and `setTransform()` methods. Use these methods to manipulate the volume and pan position of each stereo channel independently, as if the stereo sound were two monophonic channels. Read the Flash IDE Help documentation for more detailed information about these two Sound Class API methods.

### 3.7 Guidelines for preparing native audio content

Always test native audio sounds through the Flash Lite player. The audio output for Flash Lite yields a different sound quality than the built-in sound player of a device.

Use a graphic equalizer to alter audio to achieve optimal projection through external speakers. Deemphasize bass and high-end frequencies and emphasize mid-range frequencies.

Avoid using sounds with deep bass or high-end frequencies. These frequencies may not project well through the small external speakers of a mobile device.

Use RAW or ADPCM-compressed sounds for looping instead of MP3. On some devices, Flash Lite may not have enough CPU resources to seamlessly loop MP3-compressed sounds, and instead plays the loop with gap.

**Caution:** Do not use the 5 kHz sample rate. Flash Lite 2.0 plays 5 kHz sounds incorrectly at a slower speed and lower pitch.

## 4 Managing Flash Lite 2.0 memory usage

The total assets that Flash Lite can have loaded at a given time must not exceed the total memory allocated to Flash Lite. This includes device sound assets and the SWF application loading device sound. If you attempt to load assets that exceed the player memory limit, Flash Lite will display a player error message, "Not enough memory to open file." It is important to carefully plan how your application will use large sound file assets to ensure that users have the best possible experience and avoid unwanted player errors.

Use the following `fscommand2` functions to get information about available memory and total memory.

```
/*
get the amount of free memory in kilobytes from the device
*/
var freememory:Number = fscommand2("GetFreePlayerMemory");

/*
get the amount of total memory in kilobytes allocated by the device
*/
var totalmemory:Number = fscommand2("GetTotalPlayerMemory");
```

In practice, the memory available for loading sound is less than the value returned by the `GetFreePlayerMemory` function. For example Flash Lite 2.0 for a Nokia 6290 mobile device may indicate that it has 2016 kb of memory available after loading a simple SWF application, however it is limited to loading additional device sound files of 600 kb or less without generating a memory error.

Flash Lite 2.0 tends to allow for larger sound assets bundled within the `_level0` SWF compared to loading external sounds or sounds stored in external SWF. In some cases it may be more memory efficient to sequentially load SWF files containing large audio assets into `_level0` to avoid out-of-memory errors.

## 5 Terms and abbreviations

Term or abbreviation	Meaning
3GP	3GP is an audio/video format developed by the Third Generation Partnership Project that contains either AAC or AMR-NB audio streams.
AAC	Advanced Audio Coding format standardized in the MPEG2 and 4 specifications.
ADPCM	Adaptive Differential Pulse Code Modulation, a CPU-efficient form of audio encoding generally reducing file size to 25% of original or 4:1 compression.
AIFF	Uncompressed Audio Interchange format common to the Apple Computer platform.
AMR-NB	AMR Narrow Band (NB) is a monophonic voice codec supporting a range of bit rates from 4.75 to 12.2 kbps.
AMR-WB	AMR Wide Band (WB) is a monophonic voice codec based upon ACELP with range of bit rates from 6.6 to 24 kbps.
CMIDI	Compact MIDI is a subset of the MIDI specification developed by DoCoMo and Faith Inc. as a CPU-efficient MIDI format for mobile applications.
Compound sound	A format developed by Adobe for the Flash IDE to support sound for different mobile phone platforms by bundling of two or more versions of the same sound, each in a different format in a single SWF. Intended for the Flash Lite 1.0 player in the Japanese market.
FLA	Format of the Flash IDE software, not applicable to the player. Executing the test or publish command within the IDE converts FLA into SWF.
MFi	Melody for i-mode, a MIDI-based sound format developed by Faith Inc. for NTT DoCoMo i-mode service.
MIDI	Musical Digital Instrument Interface, industry standard protocol for sending messages between electronic musical instruments.
MP3	Common abbreviation for MPEG 1 layer III encoding.
OTA	Over the Air delivery of content through a cellular phone network.
SMAF	Synthetic Music Application Format, a MIDI-based ring-tone format developed for mobile devices by Yamaha.
SMF	Standard MIDI file format.
SP-MIDI	Scalable Polyphony specification for MIDI files.
SWF	Shock Wave File, compiled format for Flash Lite player.
WAV	Uncompressed Audio Interchange format common to the Microsoft Windows Platform.

## 6 References

- [1] [Flash Lite 1.1: Sound for Nokia S60 and Series 40 Devices](http://www.forum.nokia.com) available at <http://www.forum.nokia.com>

## Appendix A Sound-related System.capabilities

Flash Lite 2.0 has a set of sound properties from the `System.capabilities` class that provide information about Flash Lite 2.0 sound capabilities. Table 5 explains each sound property and the value Flash Lite 2.0 should return depending on the platform. The FL1.1 property column shows the deprecated equivalent of the `_cap` table properties for sound.

System.capability	FL1.1 property	Description	S60	Series 40
audioMIMETypes	n/a	Returns an array of audio mime types supported by the device.		
hasAudio	n/a	Returns <code>true</code> if the system has audio capabilities.	True	True
hasCMIDI	n/a	Returns <code>true</code> if the mobile device can play sound data in the CMIDI (compact MIDI) audio format.	False	False
hasCompoundSound	<code>_capCompoundSound</code>	Returns <code>true</code> if the Flash Lite player can process compound sound data.	True	True
hasEmbeddedVideo	n/a	Returns <code>true</code> if the mobile device supports video bundled with SWF.	True	True
hasMIDI	<code>_capMIDI</code>	Returns <code>true</code> if the mobile device is capable of playing sound data in the MIDI sound format.	True	True
hasMFI	<code>_capMFi</code>	Returns <code>true</code> if the mobile device is capable of playing sound data in the MFI sound format.	False	False
hasMP3	<code>_capMP3</code>	Returns <code>true</code> if Flash Lite is capable of playing sound data in the MP3 sound format through the device sound implementation.	True	False
hasSMAF	<code>_capSMAF</code>	Returns <code>true</code> if the mobile device is capable of playing sound data in the SMAF sound format.	False	False
hasStreamingAudio	<code>_capStreamSound</code>	Returns <code>true</code> if the device supports stream synch sounds in the native audio implementation.	False <code>_capStreamSound</code> returns 1 (true)	False

Table 5: Sound-related System.capabilities

**Note:** Not all devices report accurate values for the capability properties listed in Table 5. For example, Flash Lite 2.0 for the Nokia 5300 mobile device (Series 40) incorrectly reports a value of true for the `System.capability.hasMP3`, even though it cannot load MP3 sounds.

**Caution:** Flash Lite 2.0 for S60 incorrectly reports false for the `hasStreamingAudio` property. Instead, use the `_capStreamSound` property to determine support for stream sync sounds.

## Appendix B Sound Class API

Table 6 summarizes the differences between the Sound Class API features supported by the native audio implementation and the device sound implementation.

Constructor	Summary	Native Audio	Device Sound
<code>Sound(target:MovieClip)</code>	Creates a new Sound object.	Yes	Yes
<b>Properties</b>			
<code>duration</code>	Duration of a sound measured in milliseconds.	Yes  Not accessible until after initial start.	No
<code>id3</code>	Array listing values of ID3 1.0,1.1 and 2.3, 2.4 ID3 tags in MP3 files.	No	Yes
<code>position</code>	Current position of sound measured in milliseconds	Yes	No
<b>Global Properties</b>			
<code>_soundbuftime</code>	Establishes the percentage of sound represented as a number of seconds of streaming sound to buffer. The default value is 5 seconds.	Yes	No  Flash Lite 2.0 does not support progressive loading of device sound.
<b>Events</b>			
<code>onID3</code>	Invoked each time new ID3 data is available for an MP3 file	No	Yes
<code>onLoad</code>	Invoked after a sound loads completely. Passes Boolean value, to assigned function, representing success or failure of load.	No	Yes
<code>onSoundComplete</code>	Invoked after a sound finishes playing.	Yes	Yes
<b>Methods</b>			
<code>attachSound(idName:String)</code>	Attaches the sound represented by the linkage ID parameter in the Sound Properties window to a Sound object.	Yes	Yes
<code>getBytesLoaded()</code>	Returns number of bytes loaded.	No	Yes
<code>getBytesTotal()</code>	Returns total file size in bytes of sound	No	Yes

<code>getPan()</code>	Returns the pan level set in the last <code>setPan()</code> call as an integer from -100 (left) to +100 (right).	Yes	No
<code>getTransform()</code>	Returns the sound transform information for the specified Sound object set with the last <code>setTransform()</code> call.	Yes	No
<code>getVolume()</code>	Returns the sound object volume level as an integer from 0 to 100, where 0 is off and 100 is full volume.	Yes	No
<code>loadSound(url:String, isStreaming:Boolean)</code>	Loads a device sound.	No	Yes  Flash Lite does not support progressive load of device sound and ignores the <code>isStreaming</code> argument.
<code>setPan(value:Number)</code>	Determines how the sound is played in the left and right channels (speakers). Value is an integer from -100 (left) to +100 (right).	Yes	No
<code>setTransform()</code>	Sets the sound transform (or balance) information for a Sound object.	Yes	No
<code>setVolume()</code>	Sets the volume for the Sound object.	Yes	No
<code>start(secondOffset:Number, loop:Number)</code>	Starts playing the last attached or loaded sound from the beginning if no parameter is specified, or starting at the point in the sound specified by the <code>secondOffset</code> parameter, and loops the sound the specified number of times.	Yes	Yes  Flash Lite ignores the <code>secondOffset</code> and <code>loop</code> arguments for device sounds.
<code>stop(idName:String)</code>	Stops the corresponding sound object if no parameter is specified, or the sound specified in the <code>idName</code> parameter.	Yes	Yes

Table 6: Comparison of Sound Class API for Native audio and Device Sound

## About the author

Hayden Porter, the author of this document, is a Flash Lite developer with a special interest in integrating sound and music with mobile devices. He has written extensively on the subject of interactive sound including white papers for leading mobile device manufacturers and articles for publications such as *Electronic Musician Magazine*, *Music Education Technology Magazine*, and DevX.com. For more information, see <http://www.aviarts.com>.

## Evaluate this resource

Please spare a moment to help us improve documentation quality and recognize the resources you find most valuable, by [rating this resource](#).