F O R U M   N O K I A

Flash Lite 1.1: Sound for Nokia S60 and Series 40 Devices

Version 1.0; December 20, 2007

NOKIA

Contents

Change history

December 20, 2007	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 1.1 from Adobe for its devices based on S60 3rd Edition and Series 40 3rd Edition, Feature Pack 1. Flash Lite 1.1 is based on the Flash 4 player for PC and includes features for device interaction and mobile-specific applications.

An important difference between Flash Lite for mobile devices and Flash for the PC environment is that Flash Lite supports two sound playback systems. Flash Lite “native” audio has capabilities similar to the PC player. Flash Lite “device” sound is a new feature that enables Flash Lite to play sound content in formats supported by the device operating system.

Flash Lite 1.1 for S60 devices from Nokia supports both native audio and the MIDI sound format using the device sound implementation. Flash Lite 1.1 for Series 40 devices from Nokia supports only the MIDI sound format using the device sound implementation.

2 Flash Lite 1.1 device sound

The device sound implementation enables Flash Lite to play sound formats that are supported by a device operating system. In the Flash Lite 1.1 device sound implementation, Flash Lite does not play a device sound itself. Instead, it passes a copy of a device sound stored within a currently loaded SWF file to the device operating system for playback. Flash Lite 1.1 for S60 and Series 40 devices from Nokia supports playback of device sounds in the MIDI (musical instrument digital interface) format.

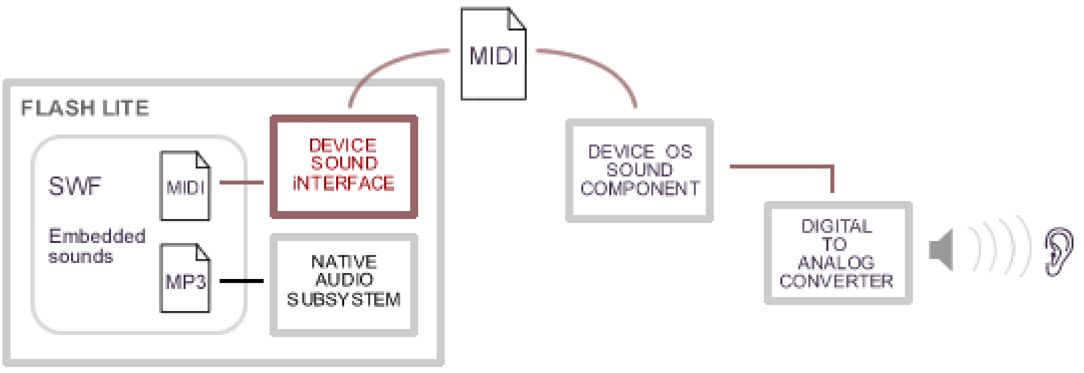


Figure 1: Flash Lite 1.1 device sound implementation

Note that Flash Lite 1.1 cannot load MIDI files directly. It is limited to playing MIDI files that are embedded within SWF files; however, for modularity, Flash Lite can dynamically load external SWF files containing MIDI files.

Tip: When creating Flash Lite 1.1 content, you should consider using the MIDI format for sound because it is the only sound format that all Nokia Flash Lite-enabled devices support.

2.1 Understanding the MIDI format

Unlike digitized audio formats such as WAV or MP3, a MIDI file contains music performance data such as the pitch, duration, time, volume, and timbre of each note in the composition. A device plays a MIDI file through a sequencer that sends MIDI data to a synthesizer to generate sound using its built-in sound banks.

Because MIDI represents sound as data instead of digitized audio, a MIDI file can be very small in file size compared to a MP3 recording, and MIDI playback can be less CPU-intensive than decoding a MP3 file. These two advantages make MIDI an effective sound format for mobile content because devices tend to have limited CPU and RAM resources and cellular networks have limited bandwidth for delivering content over the air.

Most mobile devices on the market including S60 and Series 40 devices from Nokia have GM (General MIDI specification) compliant synthesizers for playing MIDI files. The GM specification requires that a compliant synthesizer have an instrument bank of 128 common instruments and a percussion bank of 47 standard percussion instruments.

Refer to the MIDI And True Tones In Nokia Devices document for a complete listing of the instruments in the GM banks.

2.2 Limitations of MIDI

A limitation of MIDI is that it can sound differently on different devices. The sound of a MIDI file depends on the instrument timbres in the GM sound banks of a given device synthesizer. One device synthesizer may have a different sounding flute timbre than another so that the same flute music in a MIDI file sounds differently on different devices. You should test MIDI files on target devices to make sure that a given MIDI timbre will work effectively in the context of Flash Lite content. There can be noticeable variation in the timbres of sound banks in Nokia devices. You should test your MIDI files on a variety of Nokia devices to ensure that MIDI plays in a way that is acceptable for a Flash Lite game or application.

Another issue is that the MIDI specification does not define a standard means for seamless looping. For Flash Lite 1.1 content, it is possible to restart a MIDI file with a timeline loop to create a repeating sound, though this inserts a pause between repeats. This technique is most appropriate for ambient backing tracks that do not require rhythmically precise loops.

MIDI is a data only format, and cannot contain recorded audio or vocal sounds. It is not possible to convert WAV and MP3 recordings directly into the MIDI format, nor is it possible to include custom WAV samples in a GM sound bank. MIDI produces sound and music based on the 175 timbres available in the built-in instrument and percussion banks of a device synthesizer.

A “MIDI browser 2” application for auditioning each of the 175 sounds of a Flash Lite-enabled Nokia device synthesizer is included in the same package as this document.

Note: Because MIDI is a data only format, it is not possible to convert recorded audio formats like WAV or MP3 into MIDI, nor is it possible to edit MIDI in typical audio editing software. You should use a MIDI sequencer to edit MIDI files.

2.3 Using MIDI for music

In many cases you can use precomposed MIDI files or work with a musician to create MIDI music. It is important to make MIDI files that sound clearly and project loudly through the external speaker of a device. Another important issue is to create MIDI files that are compatible with a device synthesizer and optimized for CPU performance.

2.3.1 Guidelines for MIDI music

Normalize the volume of a MIDI file to maximize sound output. Normalizing MIDI requires changing the velocity value (a MIDI parameter for a note’s volume) for all the notes in the MIDI file. Many MIDI creation software have a convenient means to normalize MIDI that involves an algorithm to find the loudest note, shift its velocity to full volume, and then shift the remaining note velocities accordingly.

Select ranges for instrument sounds that project clearly. Some instruments project more loudly than others, and may project more loudly within different ranges of the instrument. If a MIDI file does not project loudly after normalizing, you may need to transpose the key of the music or shift the primary instrumental part to a different octave for better projection through the speaker.

Avoid using low frequency bass sounds. Low frequencies and low-pitched notes may not project well through the small external speakers of a mobile device.

2.3.2 Guidelines for MIDI compatibility

Use MIDI files with 65-note polyphony or less. Polyphony is the number of notes a synthesizer can play simultaneously. S60 3rd Edition and Series 40 3rd Edition devices from Nokia support 65-note polyphony at the maximum. A Nokia device will perform the MIDI file incompletely if the file contains more than 65-note simultaneous polyphony. In most cases, 65-note polyphony is more than sufficient for MIDI music. Consider using MIDI music with fewer simultaneous notes, which can be more CPU-efficient for a synthesizer to play.

Use MIDI files with SP-MIDI channel priority messages. The SP-MIDI (scalable polyphony MIDI) specification defines a set of rules known as channel priority messages. A composer adds these rules to a MIDI file to prioritize musical parts so that the device synthesizer plays the parts most essential to the composition if there are not enough CPU resources to manage all parts. This is an issue if the MIDI file exceeds the polyphony of a device, a device has a low battery charge, or is executing CPU-demanding animation and sound content. All Nokia devices are SP-MIDI-compliant.

Use MIDI format 0. Format 0 is a simpler, more efficient format for the device synthesizer to parse than format 1. Format 1 is useful for preserving MIDI track organization for editing purposes. Converting from format 1 to format 0 does not change the sound of a MIDI file.

2.4 Using MIDI for sound effects

Single-note MIDI files that use instruments from the percussion bank can be effective for simple user interface sound effects like button press sounds. For more advanced effects, you can design new sounds based on the 175 built-in timbres of the device synthesizer and manipulate these using MIDI sequencing techniques and MIDI controllers. MIDI controllers are parameters for altering the sound of a note or a group of notes.

The device synthesizer for S60 and Series 40 devices from Nokia supports the following common MIDI controllers:

- *Volume* – Sets the volume for a MIDI channel.
- *Pan* – Sets the pan for a MIDI channel, for devices supporting stereo output.
- *Pitch Bend* – Changes the pitch of a note or group of notes, defaulting to a range of 2 semitones above and below a note.
- *RPN* – Parameter for altering the range of pitch bend.
- *Modulation* – A rapid change in pitch also known as vibrato.
- *Master volume* – Controls the volume of all tracks.

Note: Nokia devices do not support pitch bend controller messages for sounds in the GM percussion bank.

2.4.1 Common MIDI sequencing techniques

Create an echo effect by copying music from one MIDI track into another, offsetting its time position and lowering its volume.

Create a phaser/chorus like effect by copying music from one MIDI track into another, slightly offsetting its time, and detune the copy from the original by applying pitch bend and modulation. Use the volume controller to control the “wet/dry” mix between the effect track and its original track.

Use the volume controller as a volume envelope editor to hide the attack of a built-in sound to create a new sound. The human ear tends to identify an instrument by its attack. Hiding the attack of an instrument can significantly change its character. For example, it is possible to create a “whoosh” like effect by using the volume controller to hide the attack of and fade in a cymbal sound.

2.5 Looping MIDI files

The MIDI specification does not define a standard means for looping, nor does Flash Lite 1.1 have a means to seamlessly loop MIDI files. It is possible to create a repeating MIDI sound using a timeline loop; however, this introduces a noticeable delay between the end of the loop and its beginning.

To use a timeline loop to repeatedly play a MIDI file, you can place the MIDI file in the first frame of a movie clip and add frames to the movie clip timeline so that the duration of the movie clip is one frame longer than the MIDI file. When Flash Lite reaches the end of the movie clip, it will automatically restart at the first frame and replay the MIDI sound.

It is important that the movie clip duration be longer than the MIDI file so that Flash Lite does not prematurely stop the MIDI file before restarting again. Flash Lite will continuously repeat the timeline loop and replay the MIDI sound until explicitly stopped by ActionScript. To stop the sound in this case, you must stop the playback of both the movie clip timeline loop, using the `stop()` command, and the MIDI sound, using the `stopAllSounds()` command.

This technique is most effective for ambient tracks that do not need rhythmically precise looping.

A “demo MIDI” application that demonstrates various MIDI sound effects and a repeating MIDI file is included in the same package as this document. The package also includes 175 one-note MIDI files in the MIDI browse folder.

2.6 Testing MIDI sounds

You should test MIDI sounds directly on a device to verify that it plays as intended before integrating it with Flash Lite. The main concern is that MIDI timbres and music need to project well through the external speaker of a device. You should test at loud and soft volumes, 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 synthesizer and sound banks installed on the corresponding devices. You can audition MIDI sounds by loading these sounds into the emulator.

Nokia also provides the Nokia Audio Suite 2.0 VST plug-in for Steinberg Nuendo and Steinberg Cubase 3.0+ MIDI sequencers. This plug-in contains the sound banks of many types of Nokia devices.

You can audition MIDI files directly in the desktop environment using Nokia SDK device emulators or the Nokia Audio Suite to emulate the sound of Nokia devices. Visit www.forum.nokia.com for more information about the Nokia SDK device emulators and the Audio Suite VST plug-in.

2.6.1 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**.

2.6.2 Series 40 devices from Nokia

Transfer the MIDI file to the **Ringing Tones** folder inside the **Tones** folder. On the device, play the MIDI file from the **Ringing Tones** folder within the **Tones** folder of the **Gallery**.

2.7 Integrating MIDI with Flash Lite content

You can use MIDI sounds as backing music and sound effects in a linear animation, or as interactive sounds that play in response to user interaction with a game or application. Flash Lite 1.1 can only play MIDI files that are embedded within a SWF file. You can use ActionScript to control playback of an embedded MIDI file and can also synchronize MIDI sound and animation.

2.7.1 Embedding MIDI within SWF

The Flash IDE does not support MIDI sounds directly. Instead, you first need to import a WAV, MP3, or AIFF sound into a FLA to act as a placeholder or “proxy” sound that the IDE replaces with a MIDI file during the SWF testing or publishing process. Each MIDI file must have a corresponding proxy sound in the FLA library.

Embedding a MIDI file in a Flash Lite 1.1 SWF is a four-step process:

1. Import a proxy sound file in WAV, AIFF, or MP3 format into the FLA library. Name the sound asset according to its corresponding MIDI file.

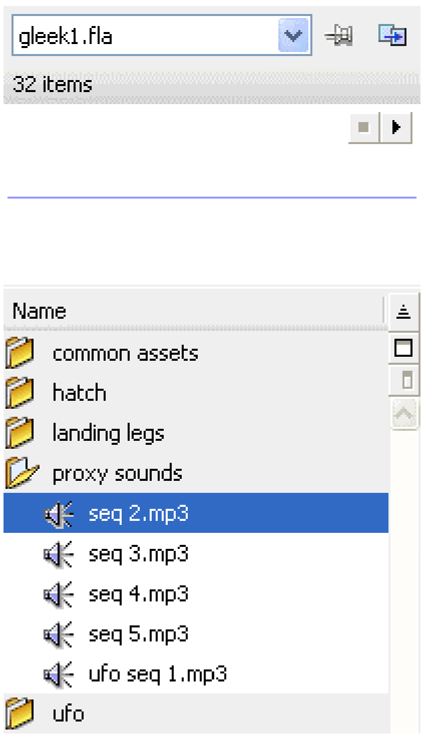
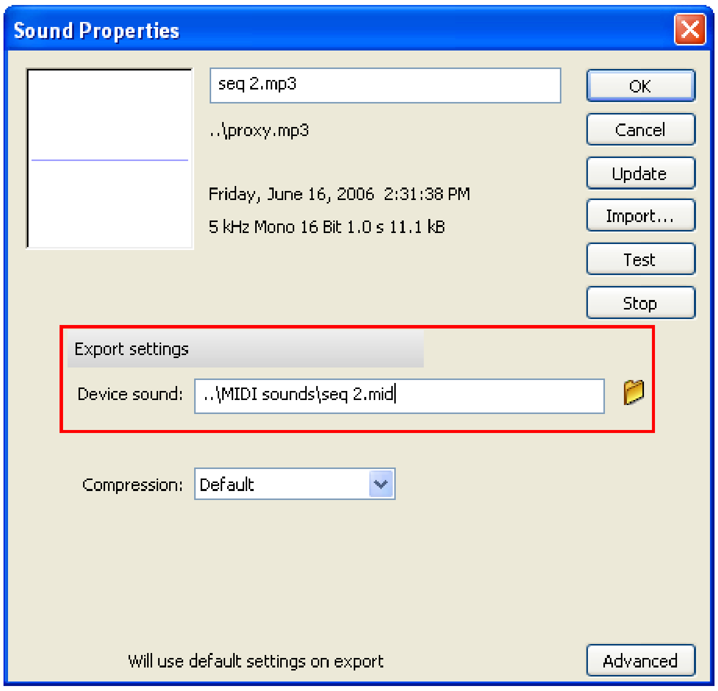


Figure 2: FLA library

2. From the Library window, open the sound properties window for a proxy sound asset and browse to the location of the corresponding MIDI sound. This step assigns the selected MIDI file to the proxy sound.



Sound Properties

seq 2.mp3

..\proxy.mp3

Friday, June 16, 2006 2:31:38 PM

5 kHz Mono 16 Bit 1.0 s 11.1 kB

Export settings

Device sound: ..\MIDI sounds\seq 2.mid

Compression: Default

Will use default settings on export

Figure 3: Sound properties window

3. Place the proxy sound asset from the library in the frame where the MIDI sound should play. The Flash IDE displays wave forms for proxy sounds in green and native audio sounds in blue.

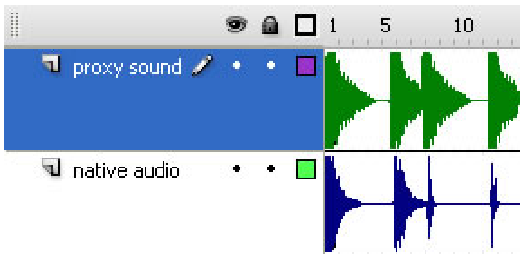


Figure 4: Proxy sounds and native audio sounds

4. Test or publish the Flash Lite application. The IDE replaces all instances of the proxy sound with the MIDI sound, embedding it within the resulting SWF.

After making changes to a MIDI file, you can retest or republish the FLA to update MIDI sounds in a SWF.

None of the sound settings in the Flash IDE — such as the Sound Inspector, Sound Edit window, or Sound Properties window — apply to device sounds.

The Flash Lite emulator for Adobe Device Central and Flash 8 IDE plays MIDI using the QuickTime synthesizer. The instrument timbres of the QuickTime synthesizer sound differently than those in Nokia devices and some aspects of MIDI may not play correctly through the QuickTime synthesizer.

You should test the SWF containing MIDI sounds in the Nokia SDK device emulator, which plays MIDI using the same or similar MIDI synthesizer timbres as those installed on Nokia devices. Ultimately, it is best to test Flash Lite SWF containing MIDI sounds directly on the target devices.

Tip: For projects involving MIDI only, use a short, silent MP3 proxy sound. This sound can be reimported and renamed in the library without the need to create a unique sound for each MIDI file. A short, silent MP3 8 kbps compressed sound also reduces the size of the FLA.

2.7.2 Controlling MIDI playback with ActionScript

Flash Lite 1.1 for Nokia devices can start and stop MIDI sounds, though only one MIDI sound may play at a time. Flash Lite 1.1 does not have a dedicated command for starting sound. Instead, ActionScript is used to move the Flash Lite movie clip playhead to a frame containing a sound to start the sound, and the `stopAllSounds()` command is called to stop the sound.

To facilitate control over individual sounds, sound-only movie clips are often created as containers for sounds. Typically, a sound-only movie clip has a `stop()` command in the first frame to prevent the Flash Lite player from automatically playing the sound at run time, a MIDI sound in frame 2 with a label such as “sound”, and a `gotoAndStop(1)` command in frame 3 to reset the movie clip to frame 1, after playing sound. Resetting the movie clip to frame 1 makes the timeline ready for subsequent sound playback.

In the code example below, pressing the 1 key causes Flash Lite to play a sound by moving the playhead to the frame labeled “sound” in a sound-only movie clip named “soundmc”. Pressing the 2 key calls the `stopAllSounds()` command, which causes Flash Lite to stop the MIDI sound.

```
// play the frame labeled “sound”, containing a MIDI sound
on(keyPress "1") {
   tellTarget("soundmc"){gotoAndPlay("sound");}
}

// stop MIDI sounds
on(keyPress "2") {
   stopAllSounds();
}
```

Refer to the “Control Sound” SWF and FLA (included in the same package as this document) for an example FLA demonstrating interactive MIDI sound control. Press the 1 key to start or restart sound and the 2 key to stop sound.

2.7.3 Synchronizing MIDI and animation

Flash Lite 1.1 has no built-in means to resynchronize MIDI sound and animation, but it is possible to achieve a form of synchronization by starting both animation and MIDI sound at the same time.

Flash Lite 1.1 briefly pauses animation and ActionScript when starting a MIDI sound. This may have an impact on the flow of animation or gameplay. A second issue is that, without a lock-to-frame capability, MIDI sound and animation playing at the same time can drift out of synch. MIDI always plays at a constant rate; however, the Flash Lite frame rate can vary depending on the time it takes to render a graphic or execute ActionScript for a given frame.

Achieving effective synchronization between MIDI and Flash Lite 1.1 animation requires that Flash Lite 1.1 play sound frequently enough to resynchronize sound and animation, but not too frequently to disrupt the flow of animation with many pauses.

You can use the “Sound Interrupt” SWF, which is included in the same package as this document, to determine the duration of the pause that Flash Lite 1.1 introduces when playing a MIDI sound. This application uses a looping movie clip to play a sound every 20 frames and displays elapsed time between frames including the pause after playing MIDI sound or a native audio sound.

2.7.4 Guidelines for synchronizing MIDI and animation

Verify that the duration and speed at which Flash Lite 1.1 renders an animation is the same or similar on all target devices. Significant variation in the time for Flash Lite 1.1 to render animation on different device models may prevent reliable synchronization between sound and animation across devices.

Include sequences of sounds or music within a single MIDI file instead of distributing individual sounds as separate MIDI files throughout a timeline. This technique reduces the number of MIDI files Flash Lite plays, consequently reducing the number of pauses that Flash Lite may introduce during animation.

Use short sound and music sequences of 10 seconds or less. Shorter sequences reduce the possibility that animation and sound will loose synch over time.

Divide the animation into sections based on natural pause points. Start the MIDI sound at the beginning of a section to minimize interruption in the animation. The delay caused by starting the MIDI sound will be less noticeable during natural pauses in the animation.

Avoid starting MIDI sounds in the same frame as an arrival point in an animation because the pause introduced by starting MIDI may interrupt the flow of animation at an important moment. To achieve this type of synchronization, start a MIDI file containing the desired arrival point sound effect at a prior natural pause point in the animation. Pad the beginning of the MIDI file with a duration of silence equal to the number of frames between the natural pause point and the arrival point of the animation.

Test sound and animation synchronization in the Nokia SDK device emulator. The emulator provides a close rendition of both the MIDI sound timbres on Nokia devices and the synchronization behavior between Flash animation and MIDI sound as it would play on a Nokia device.

For an example of synchronizing MIDI with a long-form animation, see the “Gleek animation” SWF for Flash Lite 1.1 that is included in the same package as this document.

Note: To maintain backlighting during the animation when playing through Flash Lite 1.1, periodically press any key, except 1 or “enter”.

3 Flash Lite 1.1 native audio

In addition to supporting MIDI device sound, Flash Lite 1.1 for S60 devices from Nokia also includes native audio capabilities. In the native audio implementation, Flash Lite 1.1 plays sound through its built-in audio subsystem, which is comparable to the audio features of the PC Flash 4 player.

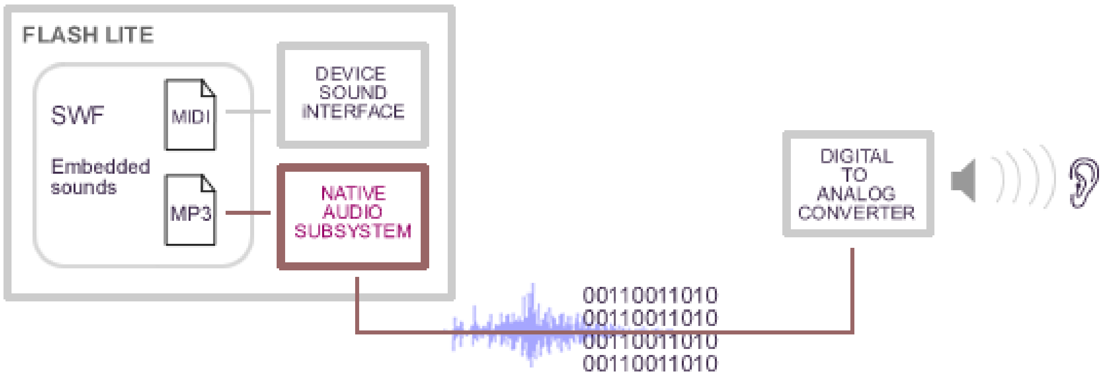


Figure 5: Flash Lite 1.1 native audio implementation

Flash Lite 1.1 for S60 devices from Nokia cannot play MIDI device sound and native audio simultaneously, but it can alternate between the two sound systems.

Flash Lite 1.1 native audio capabilities include a software MP3 decoder, seamless sound looping, built-in sound and animation resynchronization, and layering of 4 to 8 sounds depending on the processing power of a device.

Note: Unlike the PC Flash player, Flash Lite 1.1 for S60 devices from Nokia does not progressively load sounds. It must load a sound completely before playing the sound.

The Flash Lite 1.1 native audio implementation supports digital audio formats such as WAV, AIFF, and MP3 that are stored within SWF files. You should be cautious with digital audio content because large file size audio assets within a SWF may lead to memory management issues and exceed OTA (over the air) file size limitations. Also, using the MP3 decoder or layering sounds in combination with executing animation and ActionScript can be demanding on CPU and RAM usage, leading to unwanted player error messages.

Flash Lite 1.1 cannot load audio files directly. Instead, you need to embed WAV, AIFF, or MP3 sounds within a SWF. For modularity, Flash Lite 1.1 can load and unload SWF files containing audio assets at run time to help manage RAM usage.

3.1 Implementing native audio with the Flash IDE

Flash Lite 1.1 for S60 devices from Nokia employs a timeline approach to playing sound. Using the Flash IDE, you can import WAV, AIFF, or MP3 sounds into a FLA library and then insert them into a frame of a timeline in the FLA. The Flash Lite player plays a sound when its playhead enters a frame containing sound. A timeline can contain a single sound, layers of sound, or sounds in a sequence.

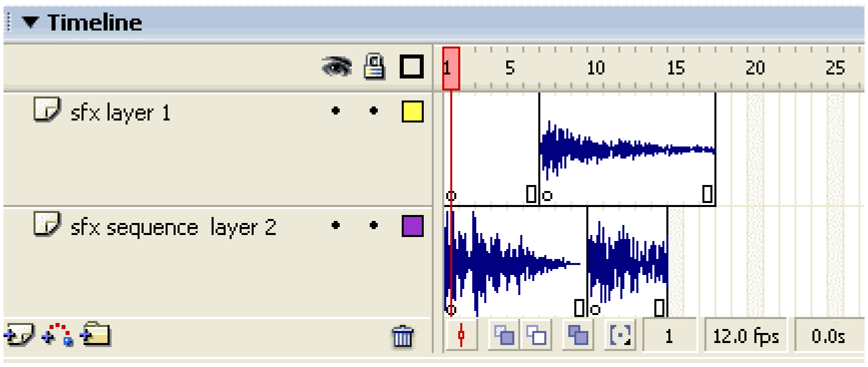
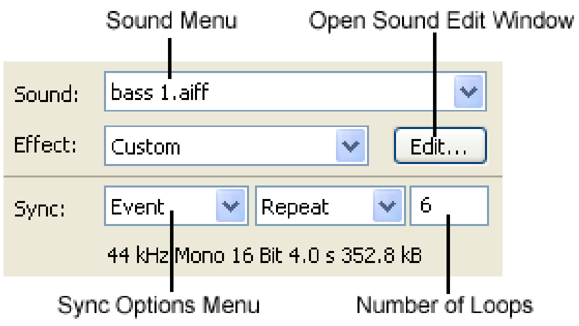


Figure 6: Flash Lite timeline

You can configure a sound using the Flash IDE sound inspector, usually appearing at the bottom of the IDE workspace. From the inspector, you can insert a sound into a frame, choose a sync option, choose the number of loops, or modify the sound using the “Edit” button. The sound inspector consists of a Sound menu, Effect menu, Edit button, Sync menu, and Loop menu.



Sound: bass 1.aiff
Effect: Custom
Sync: Event Repeat 6
44 kHz Mono 16 Bit 4.0 s 352.8 kB

Figure 7: Sound inspector

3.1.1 Sound menu

The sound menu lists of all the sounds in the library. You can add (or change) a sound to a frame by first selecting the keyframe to contain a sound, and then selecting a sound from this menu. The Flash IDE displays the waveform of the selected sound in the keyframe. You can add frames to the keyframe to see the entire sound waveform if necessary. To remove a sound from a frame, select the frame and then select “none” from the sound menu.

3.1.2 Sync options menu

A sync option determines how a sound synchronizes with graphics and the means to start and stop a sound. There are four sync behaviors.

3.1.2.1 Stream sync

Stream sync sounds are represented as frame size segments of audio synced with graphics in the corresponding timeline. The Flash Lite 1.1 player will render graphic frames in sync with sounds set to stream sync. If it is unable to render the graphics for a frame in time to keep sync with sound, Flash Lite will skip the frame and begin rendering the next to ensure that sound and graphics remain synchronized. Stream sync is effective for synchronizing long-form sounds with animation, such as voice over content or music tracks.

Note: Note that Flash Lite 1.1 does not support progressive loading of stream sync sounds.

3.1.2.2 Stream sync guidelines

Add enough frames to the timeline to see the entire waveform of the sound or the desired portion of a stream sync sound to be played. Flash Lite will only play the portion of sound represented on the timeline.

Optimize graphics that are synchronized with sounds set to stream sync. Flash Lite will skip the frames that it cannot render in time to keep up with sound, which results in a less smooth animation.

Avoid looping stream sync sounds. Setting a number of loops in the inspector for a stream sync sound causes the Flash IDE to duplicate the sound file by the desired number of loops, which increases the resulting SWF file size.

Set compression for stream sync sounds in the Flash Tab of the publish settings window, not the sound properties window.

3.1.2.3 Event sounds

Event is a general term for sounds set to event, start, and stop sync. Event sounds play independently of a timeline. You can reuse event sounds throughout a FLA like a symbol without any increase in file size. Event sounds are effective for music loops, short sound effects that are synched to a specific frame, game sounds, and sounds associated with button presses.

- *Event sync* – Sounds set to event sync can overlap. Playing an event sync sound, then playing it again before the original finishes, causes Flash Lite to play two versions of the sound: the original and its echo. You should avoid accidentally overlapping event sync sounds because this can increase processing demands. Event sync is appropriate for sounds that should overlap, such as reverberant gun shot sound effects in a game.
- *Start sync* – Sounds set to start sync do not overlap and will not restart unless explicitly stopped. You should use start sync for sounds that should not overlap, such as a backing music loop. If the playhead re-enters a frame containing a music loop that is already playing, start sync prevents a second copy of the loop from starting, avoiding cacophony.
- *Stop sync* – This option is a non-ActionScript sound command to stop the specified event type sound. To stop sounds set to event or start sync, Flash Lite must play a frame containing the corresponding sound set to stop sync. The Flash IDE represents stop sync sounds as a black rectangle appearing in a frame.

3.1.3 Loop setting box

Sound looping is configured in the loop setting box of the sound inspector. You can define the number of loops or select repeat for an infinite loop.

To ensure seamless looping, you should import uncompressed WAV or AIFF sounds, instead of importing sounds precompressed with MP3. MP3 compression tends to add silence to the beginning or end of a sound waveform. The Flash IDE places loop points at the beginning and end of the sound waveform. During playback, Flash Lite will include this silence in the MP3 loop, causing a noticeable gap in the loop during sound playback.

The Flash IDE offsets loop points to ensure that WAV and AIFF sounds compressed within the Flash authoring environment using the built-in MP3 encoder will loop correctly.

Note: Not all S60 devices from Nokia will seamlessly loop MP3-compressed sounds. For widest compatibility, use RAW or ADPCM compression for sounds intended for seamless looping.

3.1.4 Effect menu

In this menu, you can select to fade a sound's volume in or out over time or change stereo left and right pan position over time. In most cases, the sound edit window is used to define more precise changes to the sound volume or pan, instead of the preset volume and pan effects available in this menu.

3.1.5 Sound edit window

Selecting the Edit button in the sound inspector opens the sound edit window. In the sound edit window, you can apply non-destructive edits to a sound in a frame, such as shifting its start and stop time, or applying a custom volume and pan envelope. These edits are specific to the instance and do not affect the original source library item. Consequently you can use multiple variations of a sound throughout a SWF, without increasing file size.

Common uses of the sound edit window:

- Attenuate the volume of a sound that distorts because it is too loud.
- Shorten the length of a sound for looping or sound effect design purposes.
- Fade a sound's volume in or out over time, such as fading out a backing music loop.
- Change a sound's pan over time (only supported on devices with stereo output).
- Create a mix of multiple sounds.

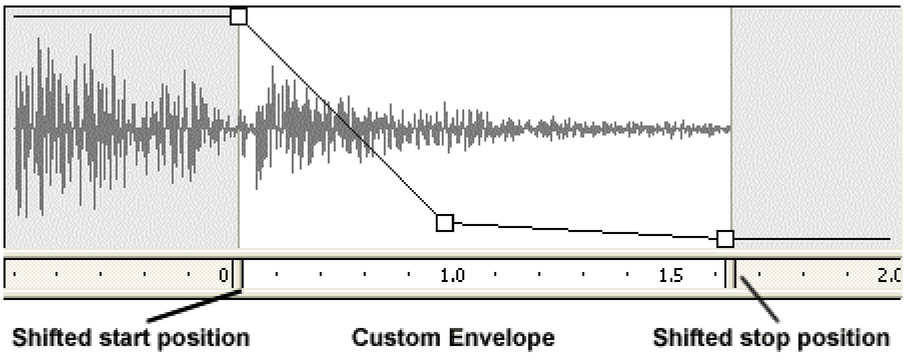


Figure 8: Example of custom edits in the edit window

Review the `soundvariations.zip` file (included in the same package as this document) for an example of using the sound edit window to create 6 sound variations from 2 sound sources.

Note: There is a bug in Flash Lite 2.0 that prevents the player from correctly interpreting Sound Edit window volume point positions or start and stop positions.

3.2 Controlling native audio playback with ActionScript

Flash Lite 1.1 ActionScript does not have a dedicated API for interacting with native audio sounds. Instead, you can use timeline navigation commands such as `gotoAndPlay()`, `play()`, or `stop()` to control playback of a sound by moving the movie clip playhead to a frame containing sound.

The means to control playback of native audio sounds depends on the sync setting of the sound. You should be aware of the difference in sound playback behavior of stream sync sounds compared to event type sounds, and choose sync settings carefully.

Note: Flash Lite tends to introduce some latency between the time when ActionScript is called to start or stop native audio sounds and when the sound actually starts or stops playing.

3.2.1.1 Controlling stream sync sounds

Stream sync sounds are tied closely to the timeline, and can be controlled using the timeline navigation commands.

- `play()` – Starts playing a stream sync sound at the current frame position. You can use the `play()` command to resume playback of a stream sync sound paused with the `stop()` command.
- `stop()` – Stops movement of the playhead and pauses sound at the current frame position.
- `gotoAndStop(frame number or label name)` – Moves playhead to a new frame position but does not resume playback. You can use this command to rewind a stream sync sound.
- `gotoAndPlay(frame number or label name)` – Moves playhead to a new frame position and begins playback at the new frame position. You can use this command to rewind or fast-forward the stream sync sound to a new frame position and restart playback at the new position.

3.2.1.2 Controlling event type sounds

Event type sounds are controlled in a different manner than stream sync sounds. Flash Lite 1.1 plays an event type sound when the playhead enters a frame containing the sound. Event type sounds continue to play independently of the graphical timeline playhead. Stopping the playhead will not stop an event type sound. Also, unlike stream sync sounds, there is no means to pause and resume playback of an event type sound at a position within the sound.

You can only stop event type sounds and start the sound from its beginning only. To stop an event type sound, you must explicitly move the playhead to a frame containing the corresponding sound set to stop sync.

The default behaviors of event sync and start sync do not support a standard stop before restart behavior. To achieve a stop before restart behavior, you must stop a sound by moving the playhead to a frame containing the corresponding sound set to stop sync, before moving the playhead to the frame containing the sound itself.

The following movie clip structure demonstrates a technique of playing a stop sync sound in a frame immediately before the actual sound. For event sync sounds, this automatically stops any previous copy of the sound before restarting the sound. For start sync sounds, this stops the existing sound enabling Flash Lite to restart it again.

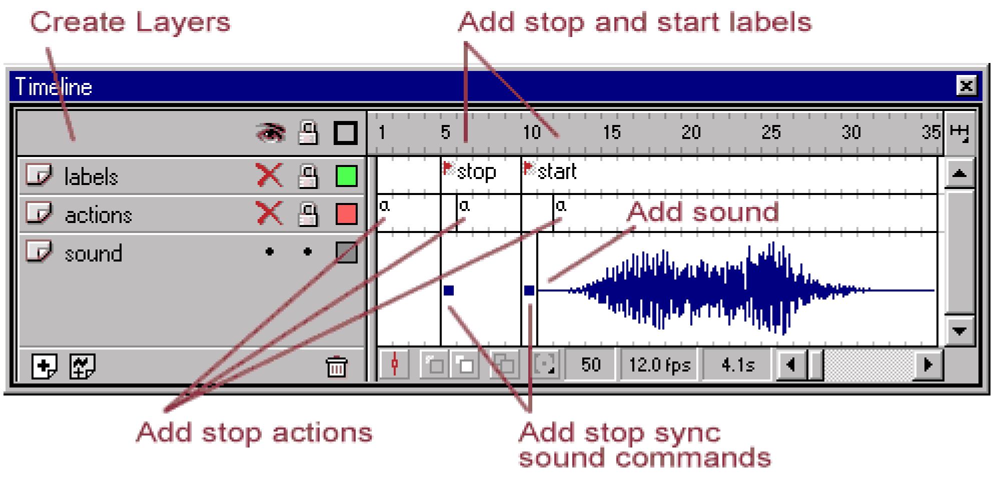


Figure 9: Example of playing a stop sync sound in a frame immediately before the actual sound

To achieve interactive control over event type sounds, you can use a movie clip structure like the example above as a container for sound, and move the playhead, using the `gotoAndPlay()` command, to the frames labeled "start" and "stop" to respectively start and stop the sound.

```
// code to start a sound
tellTarget("targetpathtosoundmc"){gotoAndPlay("start");}

// code to stop a sound
tellTarget("targetpathtosoundmc"){gotoAndPlay("stop");}
```

Tip: You can also stop all sounds that are currently playing with the `stopAllSounds()` command. This command does not prevent sounds from playing again in the future; it only terminates all sounds that are currently playing. The `stopAllSounds()` will stop a stream sync sound; however, Flash Lite must reset the playhead to the first frame of the stream sync sound before restarting it.

3.3 Using sound compression

Flash Lite 1.1 for S60 devices from Nokia can play and decode MP3-compressed sounds and sounds encoded with ADPCM (adaptive differential pulse code modulation) compression. Flash Lite does not support the "speech" codec included in the PC version of Flash.

You can apply compression to sound library items used as event type sounds in the Sound Properties window, which is accessible by double-clicking the sound in the Library window. Each event type sound in the library can have a different compression setting.

For stream sync sounds, you should set compression from the Flash Tab of the Publish settings dialog, which is accessible from the File menu. The Flash IDE applies this sound compression setting to all stream sync sounds throughout a FLA.

3.3.1 MP3 encoder

To use the Flash IDE MP3 encoder, select a bit rate from the MP3 bit rate list in the Sound Properties window, and test the resulting audio quality by clicking the test button. Lower bit rates reduce the file size and quality of sounds.

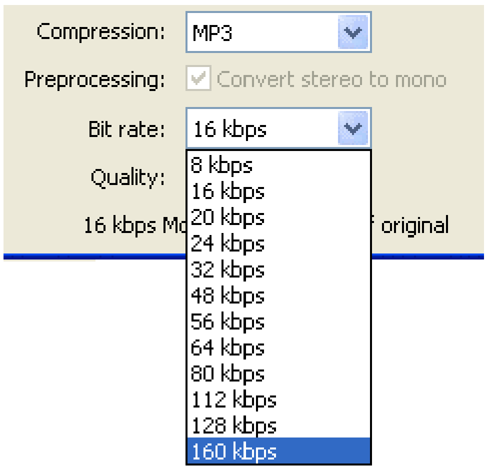

Compression: MP3
Preprocessing: ✓ Convert stereo to mono
Bit rate: 16 kbps
Quality:
16 kbps Mo… …f original

Figure 10: Setting a bit rate in the Sound Properties window

For mobile applications it is recommended to select the **Convert stereo to mono** check box. Mixing from stereo to mono may improve sound quality and be more CPU-efficient for Flash Lite to play; however, it does not significantly reduce MP3 file size. Also, most mobile devices do not yet support stereo separation through external speakers, so stereo content is not crucial, unless you intend for users to listen through headphones.

Tip: For precompressed MP3 sounds, check the "Use Imported mp3 Quality" check box and the Flash IDE will not recompress the sound when publishing or testing the SWF. Imported MP3 should be in CBR (constant bit rate) format and encoded at one of the bit rates listed in the Sound Properties window.

Note: Flash Lite 1.1 decodes MP3-compressed sound with its built-in MP3 decoder, which is more CPU-demanding than playing ADPCM-encoded sounds.

Note: Not all S60 devices from Nokia will seamlessly loop MP3-compressed sounds. For widest compatibility, use RAW or ADPCM compression for sounds intended for seamless looping.

3.3.2 ADPCM encoder

Compared to MP3, ADPCM is more CPU-efficient to decode but not as efficient at compressing sound. The Flash IDE supports a proprietary range of ADPCM encoding bit values from 5 – 2bit, reducing the file size to 31%, 25%, 19%, or 13% of the original file size. Smaller ADPCM bit values yield smaller sound file sizes and reduced sound quality.

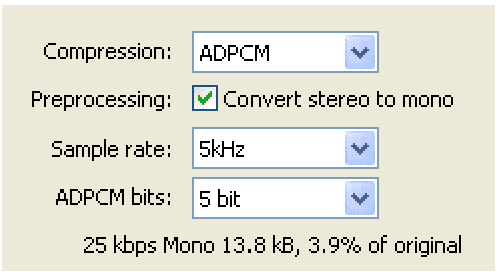

Compression: ADPCM
Preprocessing: ✓ Convert stereo to mono
Sample rate: 5kHz
ADPCM bits: 5 bit
25 kbps Mono 13.8 kB, 3.9% of original

Figure 11: ADPCM encoder

3.3.3 Sound reformatting

Developers can also use sound reformatting techniques in combination with ADPCM to further reduce the file size of a sound. The file size of a sound depends on the following parameters:

- Sound duration
- Stereo or mono
- Sample rate (44.1kHz, 22.05kHz, 11.025kHz, or 5kHz)
- Bit depth (16bit or 8bit)

Warning: For forward compatibility with Flash Lite 2.0, do not use the 5kHz sample rate. Flash Lite 2.0 plays 5kHz sounds incorrectly at a slower speed and lower pitch.

Reducing any one of these parameters removes data and decreases the file size of a sound. For mobile applications, you should mix sound from stereo to mono, saving 50% on file size. Lowering the sample rate, called "down sampling", decreases the file size by 50% for each level of sample rate, but will also remove high-end frequencies from the recording and may introduce distortion. Dropping the bit depth from 16bit to 8bit reduces the file size by 50% but can add undesirable hiss.

To use 8bit sound, you must first convert sound to 8bit before importing because the Flash IDE does not convert between 16bit and 8bit. Note that ADPCM and MP3 require 16bit-formatted sound and that these sound reformatting techniques do not apply to MP3-encoded sounds.

3.4 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. De-emphasize 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.

On some Nokia devices there is no significant sound quality difference between native audio with 44kHz sample rate and 11kHz sample rate when played through either the external speaker or headphones. Consequently, it can be more efficient to use 11kHz sample rates because this format is more CPU-efficient and smaller in file size than 44kHz. For MP3-compressed sounds, choose bit rates that automatically downsample to 11kHz such as 24 (mono), 20, 16, and 8 kbps.

For forward compatibility with Flash Lite 2.0, do not use the 5kHz sample rate. Flash Lite 2.0 plays 5kHz sounds incorrectly at a slower speed and lower pitch.

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.

Limit the total file size of sound content concurrently loaded in the Flash Lite player to less than 1 MB to avoid out-of-memory error messages.

3.5 Using native audio and device sound together

Flash Lite for S60 devices from Nokia does not support playing native audio and device sound simultaneously, but it is possible to alternate between the two sound systems. For situations where device sounds and native audio sounds overlap or play simultaneously, Flash Lite gives priority to sounds played through the native audio system.

When Flash Lite encounters a frame containing both a native audio sound and a device sound, it will play the native audio sound and ignore the device sound.

When Flash Lite encounters a frame containing a native audio sound, it will stop any device sound that is currently playing and start the native audio sound.

When Flash Lite encounters a frame containing a device sound, and a native audio sound is currently playing, it will ignore the device sound and continue playing the native audio sound.

Note: For Flash Lite 1.1, playing a native audio sound that is set to stream sync will disable device sound. A workaround for this is to play the stream sync sound, call `stopAllSounds( )`, and then play a device sound.

3.6 Comparing device sound and native audio

The primary advantages of device sound are its almost ubiquitous support across devices and its tendency for smaller file size and more CPU-efficient content compared to the Flash Lite native audio assets. Another advantage is that device sound can have higher quality output than the native audio implementation on S60 devices from Nokia which automatically alters audio sample rates.

On Flash Lite 1.1-enabled S60 and Series 40 devices from Nokia, device sound is limited to playing single MIDI files. MIDI files depend on the built-in sounds of the device synthesizer and cannot contain recorded audio or vocal audio content.

In contrast, Flash native audio has capabilities that are more suitable for multimedia applications than device sound. Native audio offers the same sound experience on compatible devices, has an MP3 decoder, can reliably synchronize sound and graphics, and can layer multiple sounds simultaneously.

Compared to MIDI device sounds, native audio assets are not as CPU-efficient, can be much larger in file size, and may not yield high-quality results due to Flash Lite downsampling during playback. Also, native audio features are not supported by all Flash Lite-enabled Nokia devices.

Features	MIDI device sound	Native audio
Supported devices	S60 and Series 40	S60
CPU usage	Minimal	Can be significant
File size	Minimal	Can be large depending on assets
Audio codecs	n/a	MP3, ADPCM
Looping	Timeline repeat	Seamless
Sound layering	Single sound only	4 – 8 sounds
Synchronizing sound and animation	Limited to trigger synchronization	Stream sync resynchronization behavior

Table 1: Comparison between MIDI device sound and native audio

4 Terms and abbreviations

Term or abbreviation	Meaning
ADPCM	Adaptive differential pulse code modulation. A CPU-efficient form of audio encoding that generally reduces file size to 25% of the original, giving a 4:1 compression ratio.
AIFF	Uncompressed Audio Interchange format common to the Apple Computer platform.
FLA	Format of the Flash IDE software, not applicable to the player. Executing the test or publish command within the IDE converts the FLA into SWF.
MFi	Melody for imode, a MIDI-based sound format developed by Faith Inc. for NTT DoCoMo i-mode service.
MIDI	Musical instrument digital 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	Compiled format for Flash Lite player.
WAV	Uncompressed Audio Interchange format common to the Microsoft Windows platform.

Appendix A Getting device volume

Flash Lite 1.1 has a set of commands known as “FSCommand2” functions for returning information about or interacting with a device. Flash Lite 1.1 for some Nokia devices supports FSCommand2 functions to get the current volume of a device and its maximum allowed volume. Note that it is not possible to set the volume using Flash Lite 1.1 ActionScript.

```
// FSCommand2 to get volume
maxvol = FSCommand2("GetMaxVolumeLevel"); // maximum allowed volume
currentvol = FSCommand2("GetVolumeLevel"); // current volume
```

Flash Lite-enabled S60 devices from Nokia support a range of 0 to 10 and Series 40 devices a range of 0 to 100 for these two parameters. Some Series 40 devices, such as Nokia 6125, do not support these parameters and return a value of –1. The default volume settings and values for `GetVolumeLevel` vary depending on the device.

Note: Flash Lite 1.1 for S60 devices from Nokia supports a volume setting option that can be set in the application Options menu. This feature is not available for Flash Lite 1.1 for Series 40 devices. Many Nokia devices have external volume control keys to facilitate changes in device volume.

You can use the two FSCommand2 functions to check that the device volume is set to an acceptable level to project sound clearly.

```
// check the device's current volume
currentvol = FSCommand2("GetVolumeLevel");
if(currentvol == 0){ // phone volume at 0
   textbox = "Phone volume is set to 0. Please set volume to 50%.";
}
```

Appendix B Using sound capability properties

Flash Lite has a set of read-only global properties that provide information about the device including its sound capabilities. Table 2 explains each sound property and the value Flash Lite 1.1 should return depending on the platform.

Property	Description	S60	Series 40
`_capCompoundSound`	Returns the value of 1 if the device supports compound sound. A compound sound is a sound bundle containing different device sound formats.	1	1
`_capMFi`	Returns the value of 1 if the device supports the i-mode MFi format.	0	0
`_capMIDI`	Returns the value of 1 if the device supports the MIDI format.	1	1
`_capMP3`	Returns the value of 1 if Flash Lite has a built-in MP3 decoder.	1	0
`_capSMAF`	Returns the value of 1 if the device supports the Yamaha SMAF format.	0	0
`_capStreamSound`	Returns the value of 1 if the device supports stream synch sounds in the native audio implementation.	1	0

Table 2: Sound properties

Note: Not all devices report accurate values for the capability properties listed in Table 2. For example, Flash Lite for Nokia 5300 (Series 40) incorrectly reports a value of 1 for both the `_capMP3` and `_capStreamSound` properties even though it does not support the native audio implementation.

You can use these variables to get information about a device during the development phase or inform a user about the sound limitations of a device. It is not necessary to use a target path to access the value of global properties.

The following code example demonstrates how to detect whether a device supports the Flash Lite built-in MP3 encoder and informs the user accordingly.

```
// notify the user that a Series 40 device does not support sound
// for a game containing stream synch sounds
if(_capMP3 < 1){ // device does not support MP3-encoded sound
   textbox = "Game sound is not compatible with this phone.";
}
```

Appendix C Launching the sound player application

In addition to playing sound from ActionScript, Flash Lite 1.1 for S60 devices from Nokia can also launch the device sound player application to play a sound. Note that this technique changes focus from the Flash Lite player to the sound player application. The sound player application will launch and display in front of the Flash Lite player covering any Flash Lite content. Users must explicitly exit the sound player to return to the Flash Lite application.

You can use the ActionScript `getURL()` function to trigger the device to launch the sound player application and play the sound at the file location specified in the `getURL()` argument. The following code triggers a device to open an MP3 file with the sound player application.

```
// open an MP3 file from the same directory as SWF
// with device’s built-in sound player application
getURL(“song.mp3”);
```

Note: This feature is not supported by Flash Lite 1.1 for Series 40 devices.

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 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.