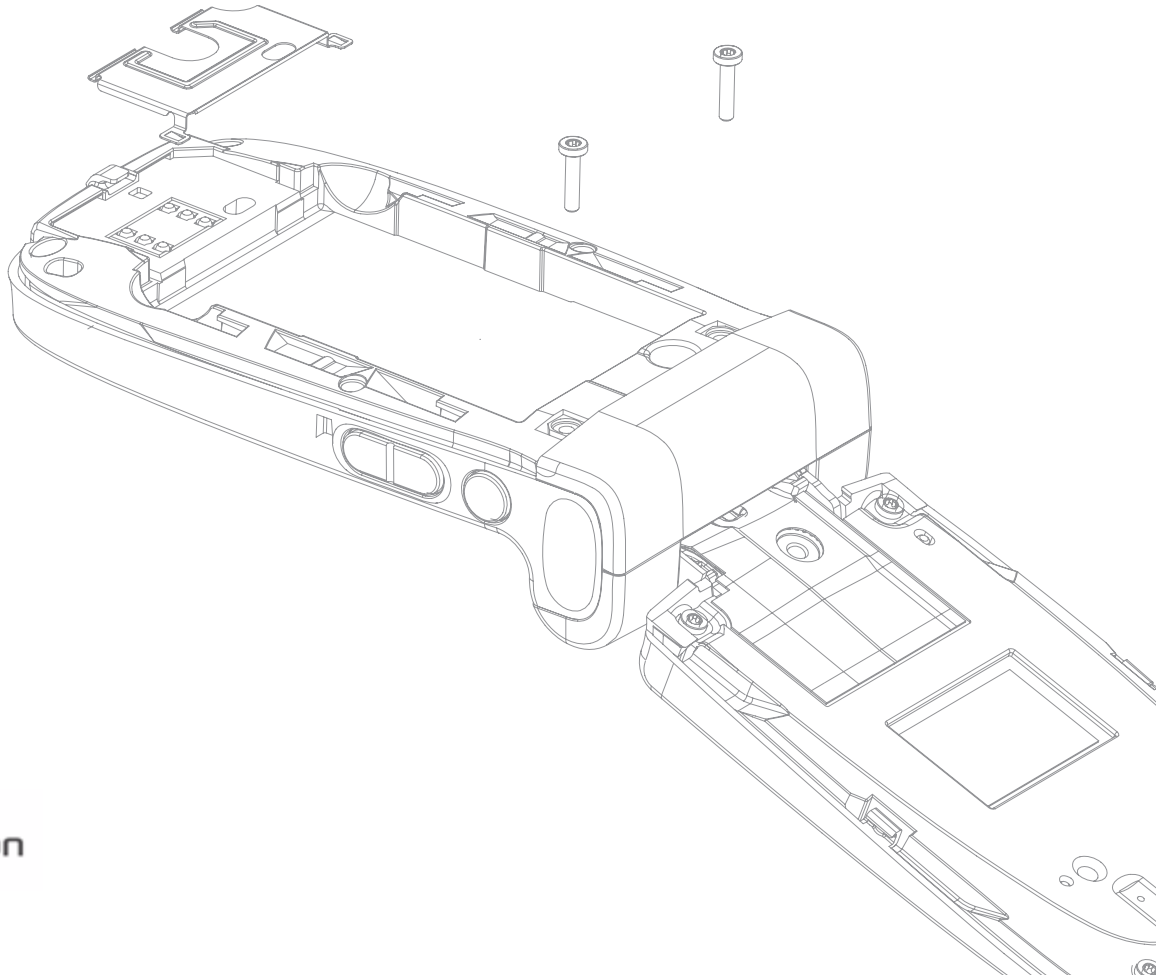
Tutorial

DEVELOPER WORLD THE FAST TRACK FROM MIND TO MARKET

March 2007

Getting Started with Sound in Macromedia™ Flash Lite™ 1.1

in Sony Ericsson phones



Sony Ericsson

Preface

About this tutorial

This tutorial was authored by Hayden Porter, www.aviarts.com. Hayden is a Flash mobile developer and musician with a special interest in integrating sound and music with mobile devices. He has written extensively on the subject of interactive sound and is the editor for www.sonify.org, a leading online resource for information about interactive sound for the web and mobile.

This tutorial describes how to get started using sound in Macromedia™ Flash Lite™ 1.1 content for Sony Ericsson phones.

The reader should have intermediate knowledge of using ActionScript and familiarity with sound terminology. Knowledge of Macromedia™ Flash™ 4 scripting is also helpful but not essential.

Sony Ericsson Developer World

On www.sonyericsson.com/developer, developers find documentation and tools such as phone White papers, Developers guidelines for different technologies, SDKs (Software Development Kits) and relevant APIs (Application Programming Interfaces). The Web site also contains discussion forums monitored by the Sony Ericsson Developer Support team, an extensive Knowledge base, Tips and tricks, example code and news.

Sony Ericsson also offers technical support services to professional developers. For more information about these professional services, visit the Sony Ericsson Developer World Web site.

These Developers guidelines are published by:

Sony Ericsson Mobile Communications AB,
SE-221 88 Lund, Sweden

Phone: +46 46 19 40 00
Fax: +46 46 19 41 00
www.sonyericsson.com/

First edition (March 2007)
Publication number: EN/LZT 108 9433 R1A

Document conventions

Abbreviations

MIDI	Musical Instrument Digital Interface
SMF	Standard MIDI File
SP-MIDI	Scalable Polyphony MIDI
SMAF	Synthetic music Mobile Application Format developed by Yamaha
MFi	Melody for i-mode™
WAV	Uncompressed audio format common to the Microsoft® Windows® platform
MP3	Audio encoded with MPEG layer 3

Typographical conventions

Code is written in Courier font: `<html>..</html>`

Trademarks and acknowledgements

Document history

Change history		
2007-03-13	Version R1A	First Edition published on Developer World

Contents

Introduction

What you need

You should have a copy of Macromedia™ Flash™ IDE Professional version to create Flash Lite 1.1 compatible SWFs and the Flash Lite 1.1 CDK, available from the Adobe mobile site at www.adobe.com/devnet/devices/development_kits.html. If you intend to edit pre-existing MIDI files or create your own MIDI music or sound effects, you should also have MIDI sequencing software.

The "Getting Started with Sound in Adobe Flash Lite 1.1" download includes example SWFs, FLAs and MIDI files that you can examine while reading the tutorial. In addition, you should review the other Getting started tutorials for Flash Lite, the Sony Ericsson Developer Guidelines for Macromedia™ Flash Lite™ 1.1, and the Sony Ericsson Developer Guidelines for Ringtones. You should also have a Flash Lite 1.1 enabled Sony Ericsson phone to listen to MIDI files and evaluate sound quality.

About Flash Lite

An important difference between Flash Lite for mobile devices and Flash for desktop PCs is that Flash Lite supports two different sound systems. Some versions of Flash Lite 1.1 support "native audio" features similar to the desktop PC player. Others support "device sound", a new feature enabling Flash Lite to play sound in formats supported by a phone operating system. Sony Ericsson Flash enabled phones only support “cached device sound”, which means that sounds to be played, first have to be stored in the phone, before it can be redirected to the audio player.

In the device sound implementation, Flash Lite does not play sound on its own. Instead, it interfaces with the phone operating system to play sound. A phone operating system may support a variety of sound formats, however Adobe intended Flash Lite 1.1 to interact with MIDI, Yamaha SMAF or i-mode MFi, depending upon an operating system support for these formats.

Flash Lite 1.1 for the Sony Ericsson OSE platform supports sound in the MIDI format through the device sound implementation. MIDI is probably the most commonly used device sound format world-wide and is also the most widely supported sound format for Flash Lite. Understanding how to use MIDI with Flash Lite ensures that our content is compatible with the widest variety of phones. Sony Ericsson i-mode phones also support the MFi sound format.

In this tutorial we focus on the sound capabilities of Flash Lite 1.1 for the Sony Ericsson OSE phone platform. In addition, we can apply these concepts to Flash Lite 1.1 for Sony Ericsson i-mode phones, which also support the MIDI format.

Tutorial

Introduction to MIDI

Many developers are familiar with using WAV and MP3 formats in the desktop version of Flash, but may be new to the MIDI format. Unlike digitized sound formats such as WAV or MP3, a MIDI file contains music performance data instead of audio. A common analogy is that a MIDI file is like an old-fashioned player piano roll, essentially a timeline listing a musician's decisions and actions during a performance such as the pitch, duration, time, volume and assigned synthesizer sound of a note played.

Representing sound in this manner has several advantages:

- MIDI file size is very small compared to a MP3 recording
- MIDI playback can be less CPU intensive than decoding a MP3 file
- An application can generate or manipulate MIDI data in real time for highly interactive sound tracks.

A phone plays a MIDI file through a sequencer that sends MIDI commands directly to a synthesizer to generate sound using the built-in sound banks of the phone.

The General MIDI (GM) specification defines two sets of sounds, that all GM compatible synthesizers must support:

- "GM instrument bank" - 128 common instruments
- "GM percussion bank" - 47 percussion instruments.

Almost all phones on the market, including all Flash enabled Sony Ericsson phones, have a GM compliant synthesizer and support these two sound banks, giving us a total of 175 built-in sounds to work with. For GM sound bank listings, see “Appendix 3 GM instrument bank” on page 24.

Included with the tutorial download is a Flash Lite 1.1 "MIDI Sound Browser" SWF application to browse through all 175 sounds on your phone, and compare sounds on different Flash Lite enabled phones. Each MIDI file is a one note or chord example played using one of the 175 synthesizer sounds. The download includes the SWF and 175 MIDI files.

One limitation of MIDI is that its sound quality depends upon the instrument timbres in the GM sound banks of a given phone synthesizer. One phone synthesizer can have a different sounding piano than another so that the same piano music in a MIDI file sounds differently on different phones. However, MIDI files played on any Sony Ericsson OSE phone sound the same or very similar because these phones share similar instrument timbres.

Another issue is that the MIDI specification does not support seamless looping. It is possible to restart a MIDI file with ActionScript to create a repeating sound, though this inserts a pause between repeats. With this technique we can have looping backing tracks for music that does not require rhythmically precise loops.

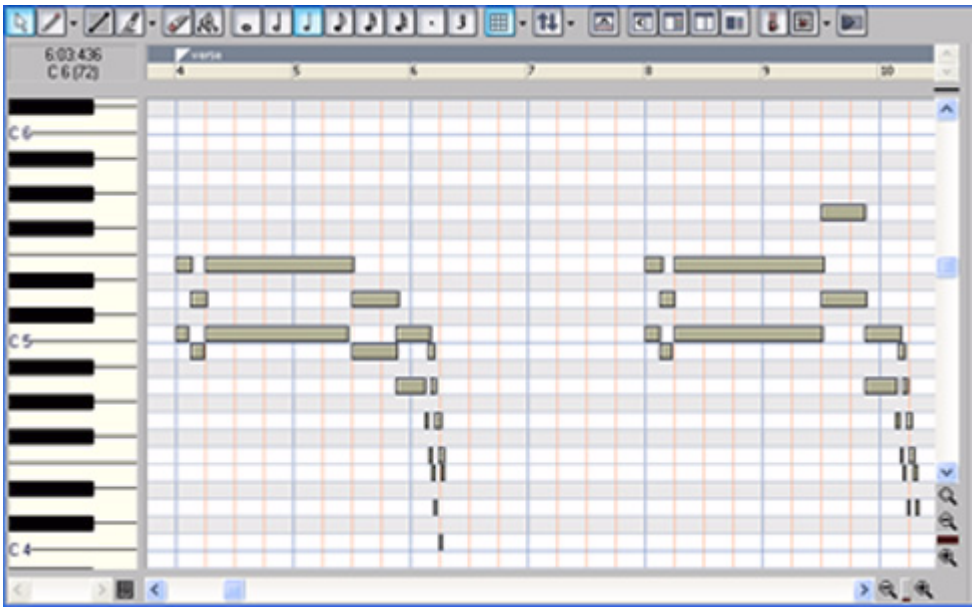
Finally, while MIDI can be effective for instrumental music and sound effects, it cannot contain audio, and is not capable of playing recorded sounds or vocal sounds. For this reason, we do not edit MIDI files in audio editing software or convert WAV and MP3 recordings directly into the MIDI format. Instead we create or edit MIDI files using MIDI sequencer software.

About MIDI sequencers

This is a brief introduction to using a MIDI sequencer. For more information on MIDI sequencing you should refer to one of the many books in publication on the topic. We record, create and edit MIDI files with a MIDI sequencer, such as the Cakewalk Music Creator™.

MIDI files consist of musical parts organized into MIDI tracks that are assigned to a MIDI channel. A MIDI channel represents a sound. We assign a sound from the GM Instrument bank to a channel. All notes in any track assigned to a channel play through our selected GM instrument sound. For example, we might record right and left hand piano parts in separate tracks for convenient editing and assign both tracks to the same MIDI channel corresponding to a piano sound.

A common way of displaying MIDI tracks is through a sequencer Piano Roll Editor screen. On this screen, we record, enter or edit notes for a track. The piano roll editor displays notes as horizontal bars on a timeline. Each note is a discrete event that we can manipulate. The vertical position of a note corresponds to its pitch, its duration is the length of the bar, and its position left to right is its time in the sequence.

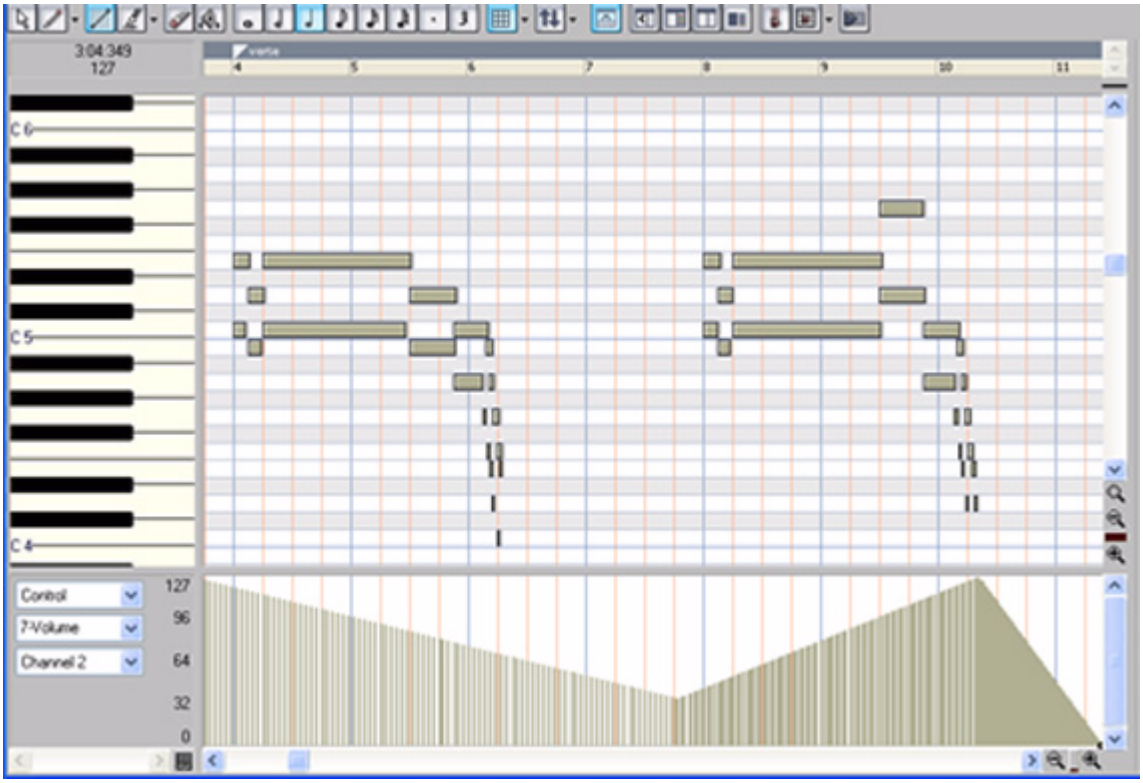


To use a sound from the GM percussion bank, we assign our track to MIDI channel 10. A GM compliant synthesizer plays tracks assigned to channel 10 using sounds from the percussion bank. A synthesizer maps each GM percussion instrument to a specific note number. We add a note to our track correspond-

ing to our desired percussion sound. For example, to create a percussion part consisting of a snare drum followed by a bass drum, we assign our track to channel 10 and enter note #38 (D3) followed by note #35 (B2). For GM percussion bank note numbers, see “Appendix 4 GM percussion bank” on page 27.

We can further manipulate a sound using MIDI controllers. Most phone synthesizers, including Sony Ericsson phones, support controllers for volume, pan, modulation, and pitch bend. Modulation is an oscillating change in pitch much like a vibrato. Pitch bend is a continuous change in pitch over time. We can also combine these different controllers for more sophisticated effects.

The MIDI controller screen displays controller data. Sequencers represent this information as a graph, where we intuitively draw a volume, pan or pitch bend envelope over time to alter a note or group of notes.



If you are new to MIDI and not familiar with how to use a MIDI sequencer, you might start by using the MIDI files provided in the download for sound effects or rely upon pre-made MIDI files. For certain projects, you might enlist a musician with ringtone development experience to help you develop MIDI music and sound effects.

Refer to the "Demo MIDI" SWF, included in the tutorial download, for examples of MIDI files with various effects.

MIDI and Sony Ericsson phones

An important phone capability is its synthesizer polyphony, which is the maximum number of notes or voices the synthesizer can play simultaneously. While some Sony Ericsson phones support as much as 72-note polyphony, Sony Ericsson recommends that MIDI files should not exceed 40-note polyphony for compatibility with all of its phones. However, for most efficient performance in CPU demanding Flash Lite applications, we may use MIDI files with even lower polyphony such as 16 voices or less.

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 phone 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 a the polyphony of a phone, a phone has a low battery charge or is executing CPU demanding animation and sound content.

For this last reason, it is safest to use music oriented MIDI files containing SP-MIDI channel priority messages to ensure that the synthesizer can scale our music to the essential parts in case of high CPU demands from playing both Flash Lite graphics and MIDI music simultaneously. All Sony Ericsson phones are SP-MIDI compliant.

Dedicated SP-MIDI content authoring tools, such as Beatnik Mobile Sound Builder™, streamline the process of adding channel priority rules to a MIDI file. These tools also analyze polyphony of a MIDI file, normalize its volume, trim unnecessary MIDI data and perform other useful tasks.

Sony Ericsson phones support both SMF (Standard MIDI Format) formats 0 and 1. There is no difference in sound between a format 0 or 1 MIDI file, though format 0 is smaller in file size, and requires less parsing than format 1. Sony Ericsson recommends SMF format 0 for most efficient CPU usage. Most sequencers can save MIDI in SMF format 0.

Review the Developers Guidelines for Ringtones for more information about the sound capabilities of Sony Ericsson phones.

Testing MIDI sounds

We should test MIDI sound directly on a phone, to verify that it plays as we intend before integrating it with Flash Lite. Our main concern is to that our MIDI sound and music project well through the phone external speaker. We should test at loud and soft volumes, through headphones and in different places to determine how it sounds against competing sounds from the environment.

If our MIDI sound does not project well through the speaker then we may need to increase the volume of the MIDI file. Most MIDI sequencers have a convenient means to normalize all note velocities to a louder value. We may also choose pitch ranges that project more loudly for a given sound, or choose a different set of sounds that project more clearly.

To test a MIDI file, we transfer it from our desktop PC computer to our Sony Ericsson phone sound or music folder and through the phone interface navigate to the sound folder, select a sound and press the enter button to play or stop the sound.

On Walkman series phones, we set the phone volume by first accessing the Walkman music player application, then pressing the external volume control buttons of the phone to change volume. After setting the volume, we return to the sound folder or Flash Lite application by closing the Walkman music player application.

A MIDI file for Sony Ericsson phones should meet the following criteria:

- 40 note polyphony or less
- SP-MIDI channel priority rules for music oriented MIDI files
- SMF format 0 format
- Project well through the external speaker at various volumes and in different settings.

Integrating MIDI with Flash Lite

Once we have a MIDI file suitable for use, we can begin the process of integrating it with Flash Lite content.

We 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, such as a game. Flash Lite 1.1 plays a sound as soon as the graphic playhead enters the frame containing a sound.

Flash Lite 1.1 in Sony Ericsson phones only plays one MIDI sound at a time. If we restart an already playing sound, Flash Lite 1.1 stops and restarts the sound, or if we start a new sound it stops the old and starts the new with no sound overlap.

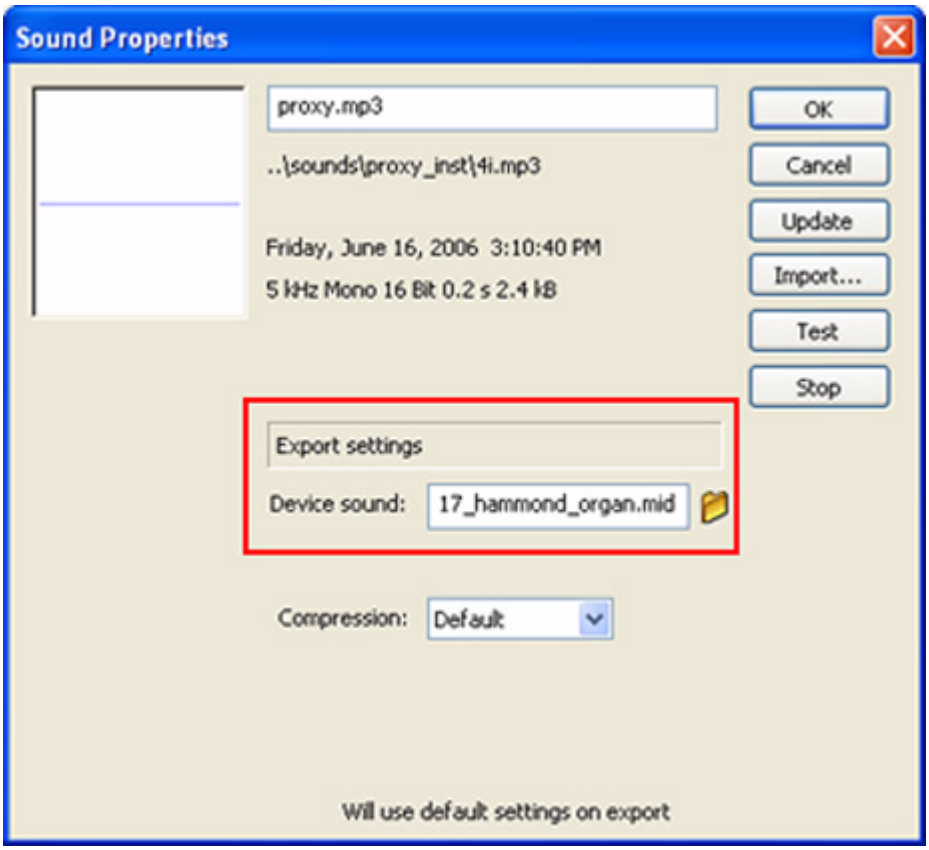
Embedding MIDI in SWF

The Flash IDE cannot directly import MIDI sounds. Instead, we import a WAV, MP3 or AIFF sound to act as a placeholder or "proxy" sound that the IDE replaces with our MIDI file when we publish the SWF or test it in the emulator.

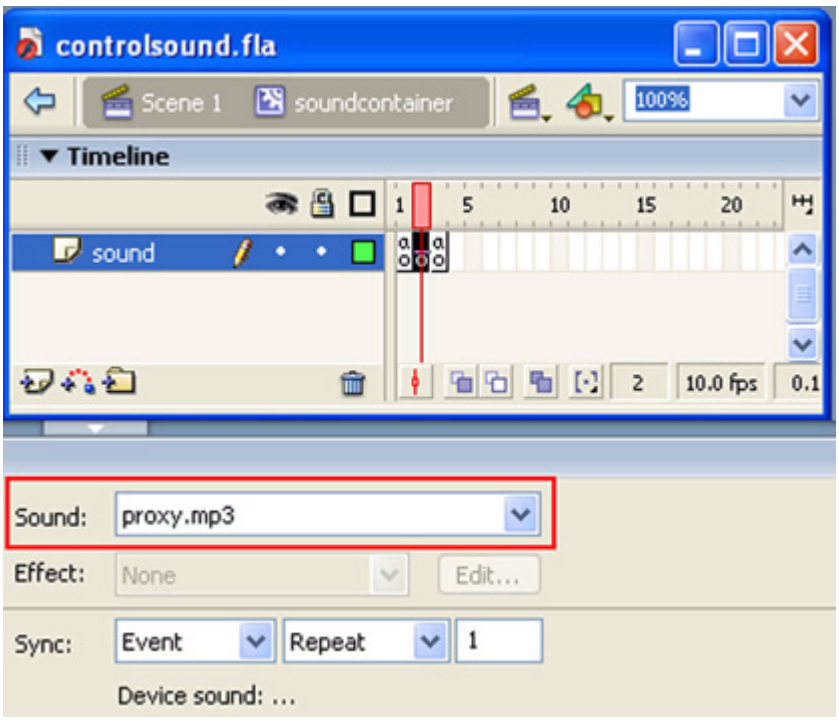
Embedding a MIDI file in a Flash Lite 1.1 SWF is a four-step process:

1. First, we import a WAV, AIFF or MP3 sound into our FLA library to act as a placeholder sound, referred to as a "proxy" sound, for the MIDI sound we intend to embed in the SWF.

2. Next, from the Library window, we open the sound properties window for our proxy sound and browse to the location of the MIDI sound.



3. Place the proxy sound asset from the library in the frame that we want the MIDI sound to play.



4. Last, we test or publish the SWF and the IDE replaces the proxy sound in the FLA with the MIDI sound, embedding it within the resulting SWF.

With the exception of assigning a MIDI file to a proxy sound, 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.

We can audition MIDI with Flash Lite in the desktop environment using the Flash IDE emulator. The emulator plays MIDI files through the QuickTime synthesizer. Note that the instrument timbres of the QuickTime synthesizer sound differently than those on Sony Ericsson phones and that some aspects of MIDI may not play correctly through the QuickTime synthesizer. It is best to test Flash Lite SWF containing MIDI sounds directly on our target phones.

ActionScript sound control

To interactively play a MIDI sound, we use ActionScript to move the playhead to the frame containing the sound. We call the `stopAllSounds()` command to stop a MIDI sound.

For example, when we press the 1 key, the code below moves the playhead to the frame labeled "sound" in the soundmc movie clip, and the Flash Lite player plays the sound. When we press the 2 key, Flash Lite stops the MIDI sound.

```
// play the frame 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 download, for an example of how we might set up a FLA for interactive MIDI sound control. Press the 1 key to start or restart sound, and the 2 key to stop sound.

Synchronizing MIDI and animation

Flash Lite 1.1 has no built-in means to resynchronize MIDI sound and animation, though it is possible to achieve a form of synchronization by starting both animation and MIDI sound at the same time. This technique is known as "trigger" synch.

Flash Lite 1.1 briefly pauses animation and ActionScript when starting a MIDI sound. Once the sound starts, Flash Lite 1.1 resumes playback. This may have an impact on the flow of animation or game play. To minimize the effect of the pause, we should play MIDI sounds at natural pause points in an animation. The duration of the pause may be more noticeable on some phones than others.

Refer to the "Sound Interrupt" SWF and FLA, included in the download, to assess the duration of the pause on your Flash Lite phone. This application uses a looping movie clip to play a sound every 20 frames and display elapsed time between frames including the pause after playing sound.

We have two approaches to trigger synch MIDI sound and animation. One approach is to start a single MIDI file containing a sequence of sound effects at the same time as an animation, so that each individual sound effect in the MIDI file plays at the same time as key frames in the animation.

This minimizes the number of pauses by reducing the number of MIDI files that Flash Lite 1.1 attempts to play. The disadvantage is that there is no guarantee that Flash Lite 1.1 animation and MIDI sound remain in synch. While MIDI always plays at a constant rate, the Flash Lite 1.1 frame rate can vary depending upon the time it takes to render a graphic or execute ActionScript for a given frame. Consequently, sound and animation can drift out of synch. This approach is most effective for synchronizing short animations with a short series of sound effects because it is less likely that the frame rate varies significantly over a short period of time.

The other approach is to distribute a series of MIDI files containing single sound effects throughout the frames of an animation so that Flash Lite 1.1 plays sounds at key frames of the animation.

In this case, sounds always play at the same time as key frames in the animation, regardless of a varying frame rate. The problem is that Flash Lite 1.1 could pause frequently if it plays many MIDI files in a short period of time, interrupting the flow of animation.

For the Sony Ericsson OSE platform, Flash Lite 1.1 runs within the web browser. The browser implementation of Flash Lite may render graphics at a slower rate than the standalone version of Flash Lite or the Flash Lite test player in the desktop IDE. MIDI sound and animation that appear in synch in the desktop IDE test player are not likely to synch on the actual device because the frame rate is slower. We need to test and verify sound and animation synchronization on the actual device.

Target Ball game example

A simple game called "Target Ball" is included in the download to demonstrate using MIDI sound effects in a Flash Lite 1.1 game for Sony Ericsson phones. The goal of the game is to set the correct spring tension to launch the ball into the target. To play the game, you press the up and down buttons to set the spring tension and the enter button to launch the ball or try again. You might try playing the game on a Sony Ericsson phone, with the browser set to display in full screen, while reading through the tutorial to hear the example sound effects.

MIDI sound effect design

The target ball game has five MIDI sound effects.

1. Launcher spring changing tension
2. Launcher spring release sound
3. Ball falling whistling sound
4. Ball landing outside of target
5. Ball landing within target

During the sound design and game development process I tested each MIDI file on W300i, W600i and W850i Sony Ericsson phones to verify that each phone had similar sounds and consistently synchronized sound with Flash Lite animation.

Launcher spring changing tension

The first sound effect we hear corresponds to the change in the tension of the launcher spring as we press the up or down buttons.

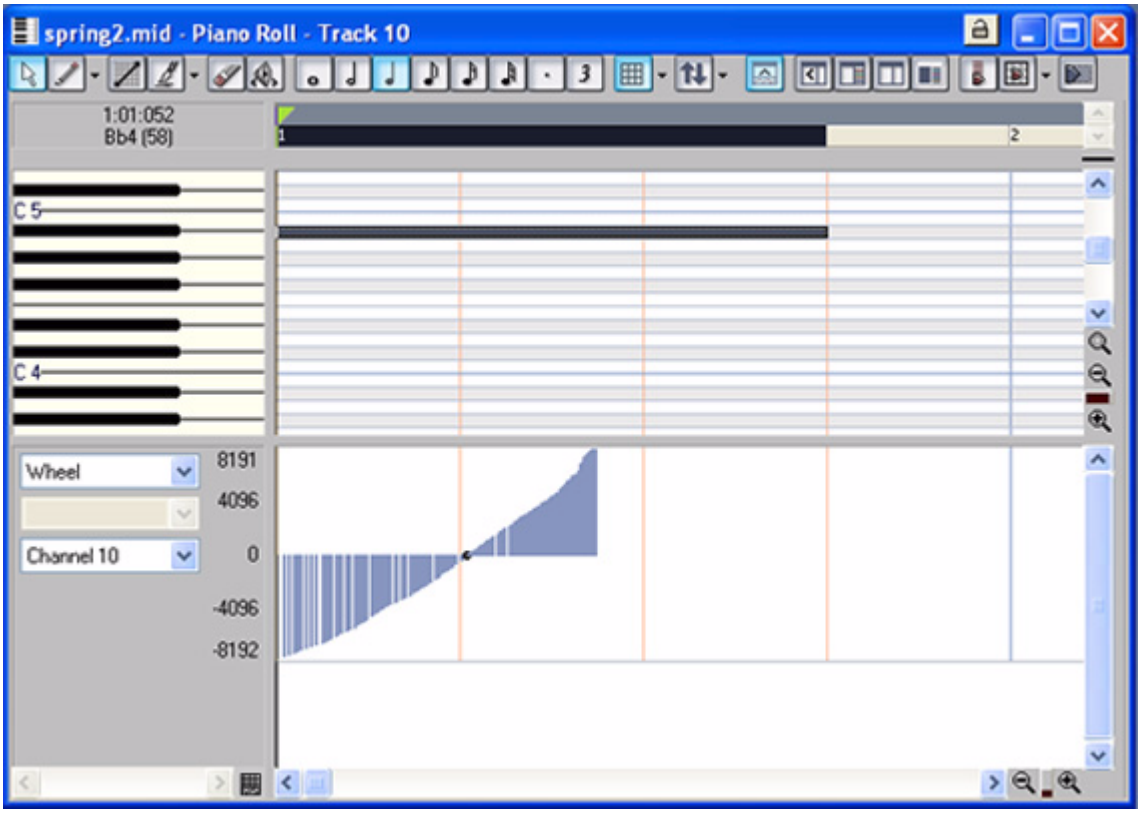
I selected the Xylophone sound from the GM instrument bank (program #14) because it is a metallic pitched percussion instrument with a short ring over producing a sound sufficient for the tightening of a coiled spring.

I created a short ascending two-note music effect, with the first note ringing over the second to create the reverberation. Then, I produced five versions of this effect transposing each to a higher pitch to represent five levels of increasing spring tension.

Launcher spring release

The second sound effect is the release of a tightly wound spring. I chose the Vibraslap sound from the GM percussion bank (note #58) because it sounds like a spring vibrating.

I assigned a track in my MIDI file to channel 10 and added the note Bb4 (note #58) corresponding to the vibraslap sound, with a duration suitable for the effect. I also applied an ascending pitch bend to enhance the sound of the spring being released.

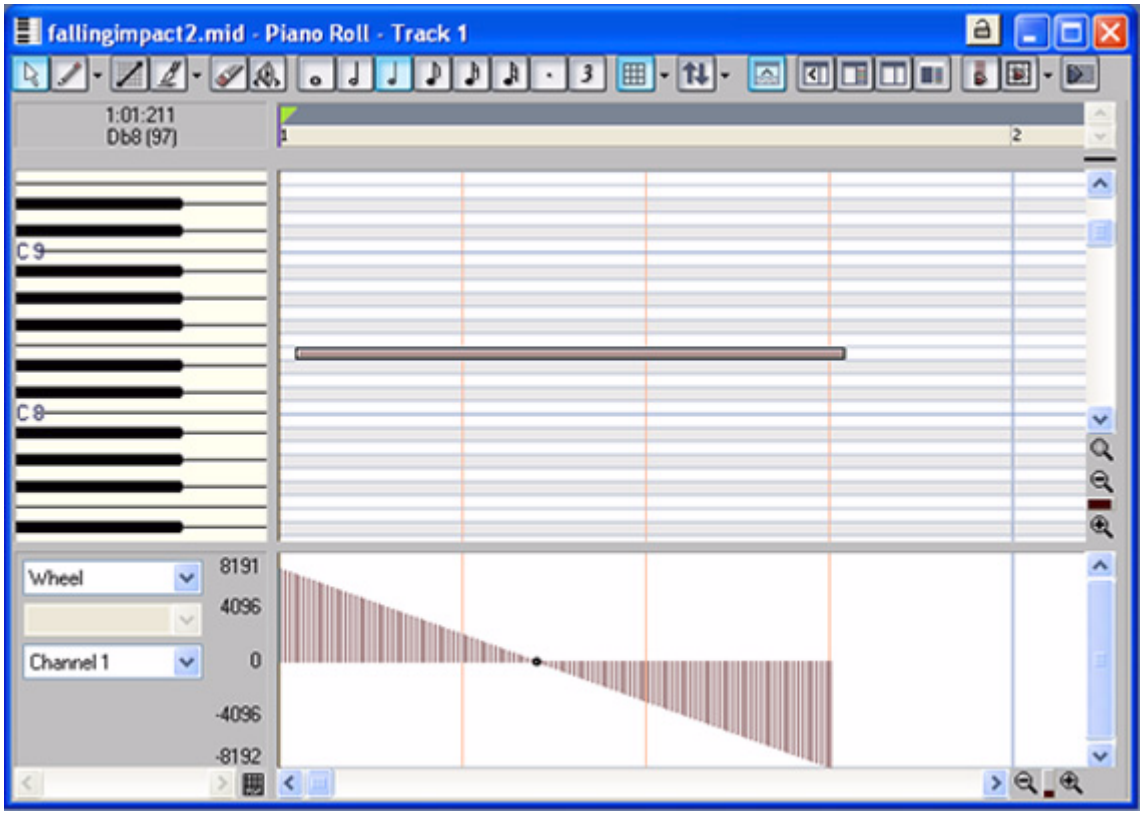


Ball falling through the air

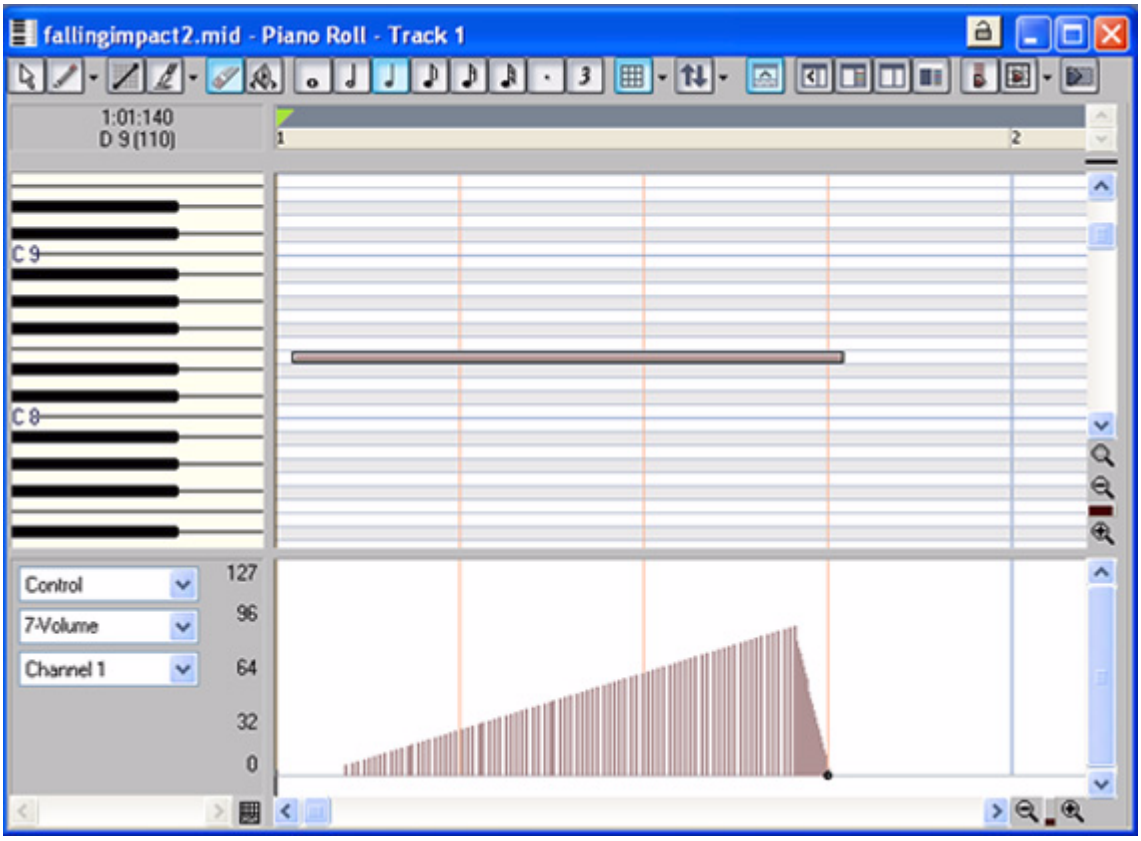
The ball falling sound requires a "whistling" timbre that changes in pitch from high to low while growing louder as the ball approaches the ground. For this effect, I selected the flute sound from the GM instrument bank (program #74) because it provides an effective whistle-like timbre when played at high pitch. I shaped the change in pitch and volume over time using the pitch bend and volume controller.

First, I selected a high-pitched note (E8) with a suitable duration for the effect. Then, I created the pitch bend and volume change contours.

Normally a synthesizer defaults to a pitch bend range of two semitones above and below a sound's starting pitch, which is not enough variation in pitch for this sound effect. Fortunately, we can reprogram a synthesizer pitch bend range by including a RPN (Registered Parameter Number) command in a MIDI file. (Read your MIDI sequencer documentation for instructions on how to change pitch bend range.) Once I created a suitable pitch bend range in the phone synthesizer, I was able to aurally experiment with different changes until I found the most effective pitch bend contour for the effect.



To emulate the sound of an object growing in volume as it approaches the ground, I started the volume controller at 0, just after the flute note starts, and gradually increased to a louder volume level, about 90% through the note duration, then rapidly decreased volume to 0 at the end.



Starting the volume change after the sound starts, hides the attack of the flute. Our ears tend to identify instrument sounds by their attack. By removing the attack, we only hear the remaining "whistle" timbre and do not recognize the sound as a flute. Reducing the volume to 0 near the end of the effect, builds expectation, just before hearing our next sound effect, the "impact" sound.

Ball impacting outside of the target

For this effect, I selected the gunshot sound from the GM instrument bank (program #128) and set it to play at a low pitch so that it sounds more like a short explosion or heavy thud. I sequenced both the ball falling whistle sound effect and the impact sound in a single MIDI file for integration with Flash animation.

Ball landing inside of the target

Finally, I needed a sound indicating success when the ball lands inside the target. Success sounds in a game generally have a positive quality, which we often associate with major keys in music. I created a short musical passage of a major chord played by the steel drum sound from the GM instrument bank (program #115), separating the notes in a harp like strum. I added this bit of music to the end of the ball falling whistle sound effect to make a second MIDI file specifically for the sound effect of the ball landing within the target.

In total, I created eight MIDI files for the project; five MIDI files corresponding to changes in the launcher spring tension, one for the spring release sound effect and two combining the ball falling whistle sound effect with the impact sound, or the musical chord representing the ball landing inside the target. These eight MIDI sound effects added only 6KB to the SWF file size.

Sound on and off control

For games, it is a common practice to provide an option to turn sound on or off because there are times when it is not appropriate for a phone to play sound. Normally we synchronize sound and animation by placing them in the same movie clip, however this makes it impossible to play animation without sound.

For more flexibility, I separated sound and animation into independent movie clips so that animation can play with or without sound. If a user has sound turned on, then ActionScript plays the sound movie clips in addition to the animation, otherwise ActionScript only plays the animation.

For the sound on and off user interface, I created button code to toggle the variable "sndplay" to 1 for sound on or 0 for sound off when the user presses the phone 1 key. The code also updates the interface to display sound status.

```
// code for sound on/off button
on(keyPress "1"){
     // sound is on, disable sound
     if(getProperty("sndtoggle",_currentframe) == 1){
          sndplay = 0; // sound control variable off
          stopAllSounds(); // stop any currently playing sound
          tellTarget("sndtoggle"){gotoAndStop(2);} //display sound off graphic
     }

     // sound is off, enable sound
     else {
          sndplay = 1; // sound control variable on
          tellTarget("sndtoggle"){gotoAndStop(1);} // display sound on graphic
     }
}
```

I used `if – then` statements to check the value of the `sndplay` variable to determine whether or not to play a sound. For example, code below determines playback of the spring release sound movie clip.

```
// control playback of spring release sound movie clip
if(/:sndplay == 1){ // sound is on
     // play spring release sound movie clip
     //frame 2 contains the sound
     tellTarget("/springreleasesnd"){gotoAndPlay(2);}
}

// always play spring release animation
tellTarget("/launcher"){gotoAndPlay("launch");}
```

If `sndplay` has the value 1, then ActionScript plays the spring release sound and animation simultaneously. If the user turns off sound, then `sndplay` has a value of 0, and ActionScript only plays the animation movie clip.

Synchronizing MIDI and animation

Both the spring tension change and spring release animations are single frame graphics that change when a user presses a key. I used code similar to the previous example to trigger synch these sound and animation movie clips.

The ball falling whistle and impact sound effect sequence required more planning because the whistle sound effect begins playing before the ball falling animation starts and the impact sound must play when the ball animation reaches the ground.

I first used separate MIDI files, with the whistle sound effect starting in a frame before the ball falling animation and the impact sound in same frame as the graphic impacting the ground to synchronize sound and animation. However, playing consecutive MIDI files caused the Flash Lite player to introduce too much pause in the animation and between the sound effects.

To work around this problem, I sequenced the effects into a single MIDI file. This approach provided more control over sound effect timing and avoided Flash Lite player pauses. Flash Lite synchronized the impact sound and graphic in a consistent manner on the three devices that I tested on.

Conclusion

MIDI can be an effective sound format for Flash Lite content for Sony Ericsson phones. We can use MIDI for sound effects, user interface sounds, music, or repeating backing tracks in animations, games and applications. The MIDI format is the most widely supported sound format for Flash Lite and its small file size and CPU efficiency are important for optimal performance on mobile devices and for delivering content over the air.

As we have learned, the process of creating MIDI sounds for Flash Lite phones is different than using WAV or MP3 sounds in Flash for the desktop player. We do not convert WAV sounds into MIDI and do not edit MIDI using sound editing software. Instead, all MIDI music and sound effects are based upon the 175 built-in sounds of a phone synthesizer. We can use pre-existing MIDI files or create our own using a MIDI sequencer. We may also use a SP-MIDI authoring tool to prepare sounds for optimal performance on phones.

To integrate sound and animation, we need to plan ahead and work within the limits of the Flash Lite 1.1 device sound implementation. We develop our Flash Lite content to play only one sound at a time and rely upon trigger sync to synchronize sound and animation.

It is important that we thoroughly test our MIDI sounds and Flash Lite content on all target devices to make sure that sounds project well through the external speaker, and that sound and Flash Lite animation play as we intend.

Mobile devices impose limitations on all aspects of Flash Lite development, including sound. However, armed with a thorough understanding of sound capabilities and limitations, we can develop creative Flash Lite 1.1 content containing sound for Sony Ericsson phones.

Appendix 1

Getting device volume

Flash Lite 1.1 has a special set of commands called "FSCommand2" functions that return information about or interact with a device. Flash Lite 1.1 for Sony Ericsson phones support FSCommand2 functions to get a phone's current volume and its maximum allowed volume. Note that it is not possible to set volume using Flash Lite 1.1 ActionScript.

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

Sony Ericsson phones default to approximately 50% volume at start up and have a maximum allowed volume of 100%. If sound is an important feature of our Flash Lite content, we may need to check that the phone volume is set to an acceptable level to project our sound clearly. If the volume is too low, we can display a message requesting that the user to increase volume to a certain level.

```
// check phone'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%.";
}
```

Read the Adobe Flash Lite 1.1 CDK for more information about FSCcommand2 functions.

Appendix 2

Sound capability properties

To enable developers to get information about a given Flash Lite device, Adobe created a set of read only global properties that provide information about the device including its sound capabilities.

There are six sound capability properties.

- `_capCompoundSound` has value of 1 if the device supports compound sound. A compound sound is a sound bundle containing different device sound formats
- `_capMFi` has value of 1 if the device supports the i-mode MFi format
- `_capMIDI` has value of 1 if the device supports the MIDI format
- `_capSMAF` has value of 1 if the device supports the Yamaha SMAF format
- `_capStreamSound` has the value of 1 if the device supports stream synch sounds in the native audio implementation.

Sony Ericsson OSE phones support MIDI and compound sound formats and should return 1 for `_capMIDI` and `_capCompoundSound`, and 0 for the other sound properties. We can use these variables to get information about a phone during the development phase or inform a user about the sound limitations of a phone. We do not need to use a target path to access the value of global properties.

```
// notify user that a Sony Ericsson phone does not support sound
// for a game containing stream synch sounds
if(_capStreamSound < 1){ // phone does not support stream synch sounds
     // print a message in a text box
     textbox = "Game sound is not compatible with this phone.";
}
```

Read the Adobe Flash Lite 1.1 CDK for more information about the capabilities table global properties.

Appendix 3

GM instrument bank

Prog #	Instrument	Prog #	Instrument
Pianos		*Reeds*	
1	Acoustic Grand Piano	65	Soprano Sax
2	Bright Acoustic Piano	66	Alto Sax
3	Electric Grand Piano	67	Tenor Sax
4	Honky-tonk Piano	68	Baritone Sax
5	Electric Piano 1	69	Oboe
6	Electric Piano 2	70	English Horn
7	Harpsichord	71	Bassoon
8	Clavi	72	Clarinet
Chromatic percussion		*Pipes*	
9	Celesta	73	Piccolo
10	Glockenspiel	74	Flute
11	Music Box	75	Recorder
12	Vibraphone	76	Pan Flute
13	Marimba	77	Blown Bottle
14	Xylophone	78	Shakuhachi
15	Tubular Bells	79	Whistle
16	Dulcimer	80	Ocarina
Organs		*Synth leads*	
17	Drawbar Organ	81	Lead 1 (square)
18	Percussive Organ	82	Lead 2 (sawtooth)
19	Rock Organ	83	Lead 3 (calliope)
20	Church Organ	84	Lead 4 (chiff)
21	Reed Organ	85	Lead 5 (charango)
22	Accordion	86	Lead 6 (voice)
23	Harmonica	87	Lead 7 (fifths)
24	Tango Accordion	88	Lead 8 (bass + lead)

Prog #	Instrument	Prog #	Instrument
Guitars		*Synth pads*	
25	Acoustic Guitar (nylon)	89	Pad 1 (new age)
26	Acoustic guitar (steel)	90	Pad 2 (warm)
27	Electric Guitar (Jazz)	91	Pad 3 (polysynth)
28	Electric Guitar (clean)	92	Pad 4 (choir)
29	Electric Guitar (muted)	93	Pad 5 (bowed)
30	Overdriven Guitar	94	Pad 6 (metallic)
31	Distortion Guitar	95	Pad 7 (halo)
32	Guitar Harmonics	96	Pad 8 (sweep)
Basses		*Synth effects*	
33	Acoustic Bass	97	Fx1 (rain)
34	Electric Bass (finger)	98	Fx2 (soundtrack)
35	Electric Bass (pick)	99	Fx3 (crystal)
36	Fretless Bass	100	Fx4 (atmosphere)
37	Slap Bass 1	101	Fx5 (brightness)
38	Slap Bass 2	102	Fx6 (goblins)
39	Synth Bass 1	103	Fx7 (echoes)
40	Synth Bass 2	104	Fx8 (sci-fi)
Solo strings		*Ethnic*	
41	Violin	105	Sitar
42	Viola	106	Banjo
43	Cello	107	Shamisen
44	Contrabass	108	Koto
45	Tremolo Strings	109	Kalimba
46	Pizziano Strings	110	Bag pipe
47	Orchestral Harp	111	Fiddle
48	Timpani	112	Shanai

Prog #	Instrument	Prog #	Instrument
Ensemble		*Percussive sounds*	
49	String Emsemble 1	113	Tinkle Bell
50	String Emsemble 2	114	Agogo
51	Synth String 1	115	Steel Drums
52	Synth String 2	116	Woodblock
53	Choir Aahs	117	Taiko Drum
54	Voice Oohs	118	Melodic Tom
55	Synth Voice	119	Synth Drum
56	Orchestra Hit	120	Reverse Cymbal
Brass		*Sound effects*	
57	Trumpet	121	Guitar Fret Noice
58	Trombone	122	Breath Noise
59	Tuba	123	Seashore
60	Muted Trumpet	124	Bird Tweet
61	French Horn	125	Telephone Ring
62	Brass Section	126	Helicopter
63	Synth Brass 1	127	Applause
64	Synth Brass 2	128	Gunshot

Appendix 4

GM percussion bank

Key #	Percussion sound	Key #	Percussion sound
35	Acoustic Bass Drum	59	Ride Cymbal2
36	Bass Drum 1	60	Hi Bongo
37	Side Stick	61	Low Bongo
38	Acoustic Snare	62	Mute Hi Conga
39	Hand Clap	63	Open Hi Conga
40	Electric Snare	64	Low Conga
41	Low floor Tom	65	High Timbale
42	Closed Hi-Hat	66	Low Timbale
43	High Floor Tom	67	High Agogo
44	Pedal Hi-Hat	68	Low Agogo
45	Low tom	69	Cabasa
46	Open Hi-Hat	70	Maracas
47	Low-Mid Tom	71	Short Whistle
48	Hi-Mid Tom	72	Long Whistle
49	Crash Cymbal 1	73	Short Guiro
50	High Tom	74	Long Guiro
51	Ride Cymbal 1	75	Clavas
52	Chinese Cymbal	76	Hi Wood Block
53	Ride Bell	77	Low Wood Block
54	Tambourine	78	Mute Cuica
55	Splash Cymbal 1	79	Open Cuica
56	Cowbell	80	Mute Triangle
57	Crash Cymbal 2	81	Open Triangle
58	Vibraslap		