Using .XA In PlayStation Development

SCEE Technical Note

June 1998

Overview

The PlayStation can be used to access CD-ROM XA file formats. This file format can be very versatile and can be used for numerous techniques. This technical note will attempt to produce a set of examples that could be incorporated into a PlayStation title.

.XA Formats

The one problem with using .XA files is the numerous ways of presenting the files, the issues surrounding certain formats and the differing ways of accessing and controlling the data from CD. The following is a list of possible combinations that can make up an .XA file; for each item in the list this article will go through a series of steps showing how to create and use the technique.

1. .XA file. Normally this is a PCM (raw audio data) or .wav file converted to .XA.
2. Interleaved .XA file. This file type normally consists of a number of separate audio files or channels interleaved together. In normal operation the file is either completely made up of audio samples or alternatively one channel is used as a data channel. This file is of a different format to a PlayStation .STR file.
3. PlayStation .STR with audio, is a type of .XA file, mainly due to the fact that the audio needs to be .XA so that during playback the audio is passed directly through the sound system without the CPU or programmer having to do anything. It is possible to interleave more than one audio track together with a video stream to allow one video sequence to have two different audio tracks.

Part 1 - Using .XA Files On Their Own

Audio data converted to .XA files can be played without programmer or CPU intervention, this data is similar to CD-DA except that the seek time is much faster than CD-DA. The playing of certain .XA files can be done at double speed, meaning that the CD system does not have to change speed to read data and play audio during game play. Another side effect is that if you use .XA as your background music then users cannot play these audio tracks on their CD players as standard DA tracks.

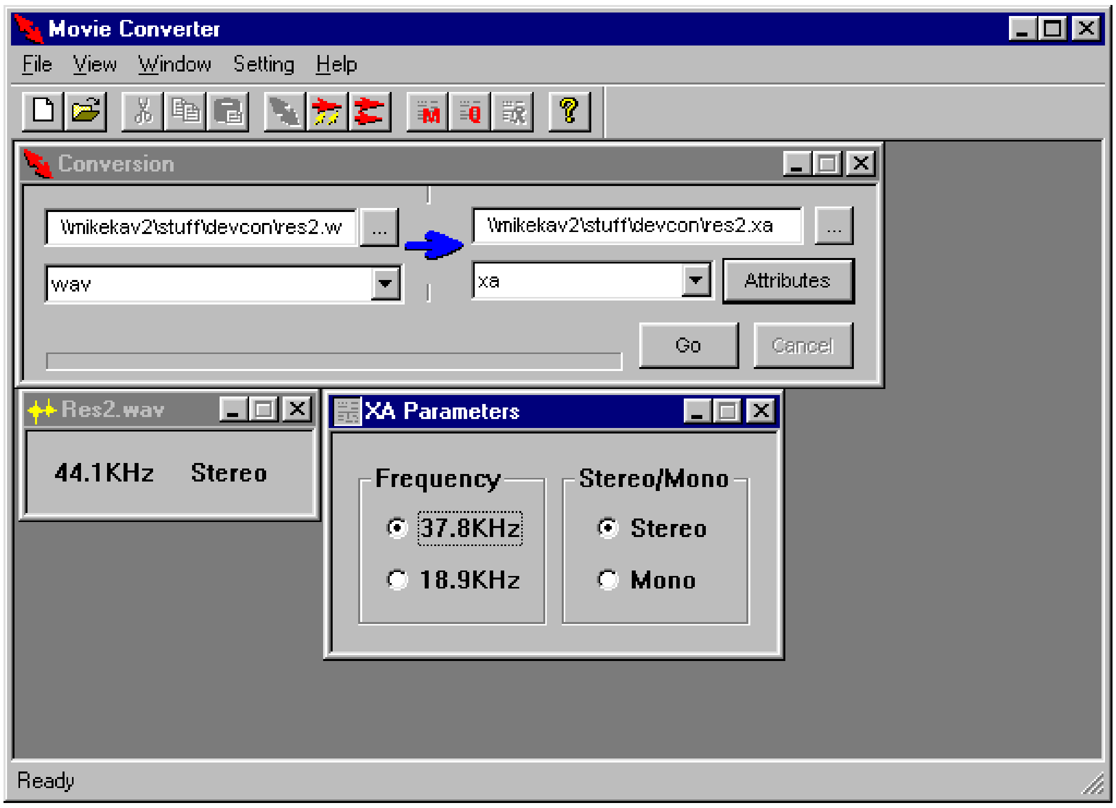
Creation

The following example uses two .XA files. One a short, seven second speech sample and the other a two minute music track.

Initially the sound source is grabbed at 44.1kHz stereo, 16 bit sample. The following are the initial properties of each file.

Filename	Res2.wav	Breath.wav
Media Length	7.00 seconds	2 minutes 0.00 seconds
Audio Format	PCM,44.1 kHz, 16 Bit, Stereo	PCM,441 kHz, 16 bit, Stereo
File Length	1,234,880 bytes	21,168,052 bytes

These samples are downsampled and converted to .XA format by MovConv 3.1, resulting in the following file properties:



\\mikekav2\stuff\devcon\res2.w → \\mikekav2\stuff\devcon\res2.xa

wav → xa

Res2.wav: 44.1KHz Stereo

XA Parameters

Frequency: (●) 37.8KHz ( ) 18.9KHz

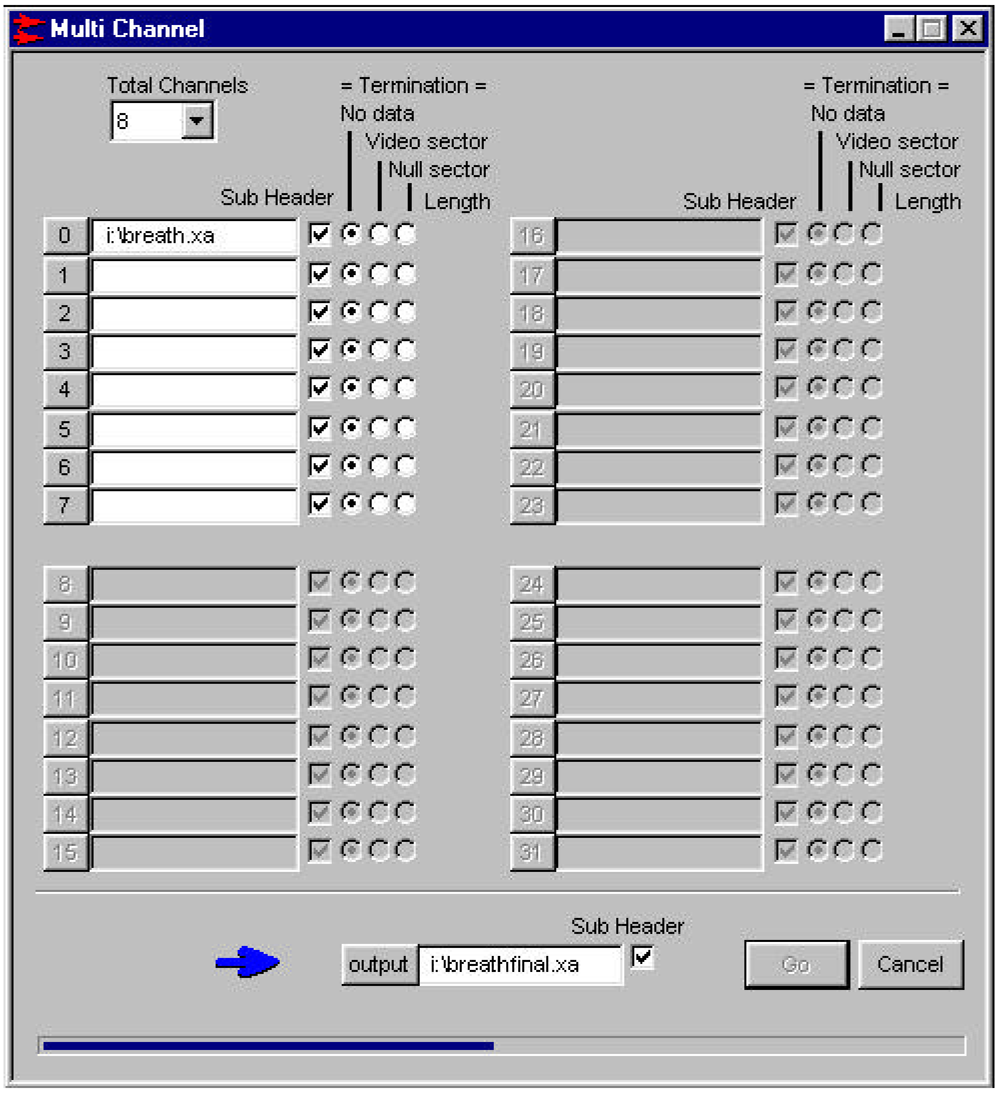
Stereo/Mono: (●) Stereo ( ) Mono

Filename	Res2.xa	Breath.xa
Media Length	7.00 seconds	2 minutes 0.00 seconds
Audio Format	XA,37.8 kHz, Stereo	XA,37.8 kHz, Stereo
File Length	308,352 bytes	5,256,000 bytes

These files now need to be converted to the correct format based on the playback speed and type. The following table shows the acceptable values for the .XA format.

Playback speed	Sampling frequency	Stereo / monaural	Data/gap ratio	Total number of channels
Double speed	37.8KHz	Stereo	1 sector / 7 sectors	8
Double speed	37.8KHz	Monaural	1 sector / 15 sectors	16
Double speed	18.9KHz	Stereo	1 sector / 15 sectors	16
Double speed	18.9KHz	Monaural	1 sector / 31 sectors	32
Standard speed	37.8KHz	Stereo	1 sector / 3 sectors	4
Standard speed	37.8KHz	Monaural	1 sector / 7 sectors	8
Standard speed	18.9KHz	Stereo	1 sector / 7 sectors	8
Standard speed	18.9KHz	Monaural	1 sector / 15 sectors	16

As we are just intending to use the .XA samples without interleaving the best choice from the table is row one. This allows us to have double speed access using the values we want for the actual sample regarding quality and output type.



Total Channels: 8

0: i:\breath.xa

output: i:\breathfinal.xa

Sub Header ✓

The following table shows the final .XA file properties:

Filename	Res2final.xa	Breathfinal.xa
Media Length	7.00 seconds	2 minutes 0.00 seconds
Audio Format	XA,37.8 kHz, Stereo	XA,37.8 kHz, Stereo
File Length	2,466,816 bytes	24,724,224 bytes

Building On To An Emulated Disk Image

The sample code used in the following examples will use both the PlayStation disk and any emulated image as data file storage only. The code will be designed to work on a development kit just by running the .CPE. The full code is available from the developers web site at:
https://www-s.playstation.co.uk/site2/ftp/developer/Sample_Code/samples/xatut.zip

The .CTI file that is used to build the image is very simple. The files are put on as XASource; this definition forces the emulator to use the sub-header information contained within the source files when building the image.

```
Disc CDROMXA_PSX
```

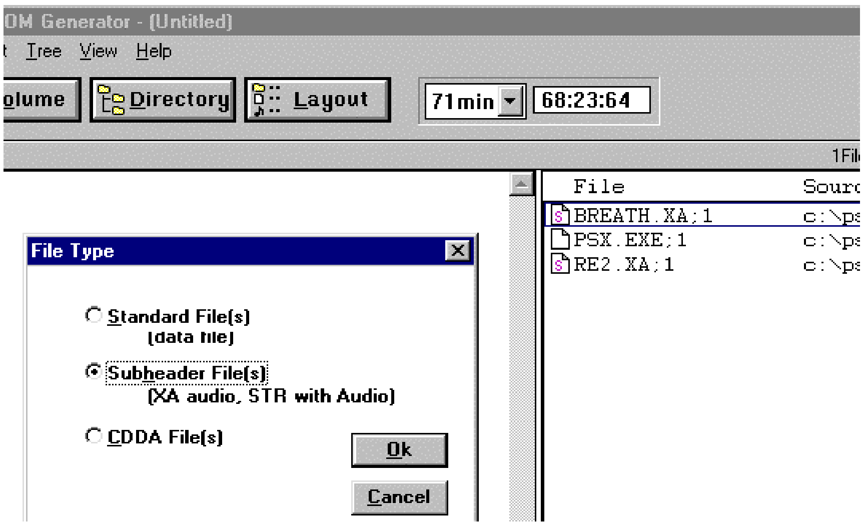
```
; LeadIn with 2 seconds of empty data
LeadIn XA
  Empty 150
  PostGap 150
EndTrack

Track XA
  Pause 150
  Volume ISO9660
    PrimaryVolume
      SystemIdentifier PLAYSTATION
      ApplicationIdentifier PLAYSTATION
      LPath
      OptionalLpath
      Mpath
      OptionalMpath

      Hierarchy

          File RES2.XA
            XASource i:\res2fi~1.xa
          EndFile

          File BREATH.XA
            XASource i:\breath~1.xa
          EndFile

      EndHierarchy
    EndPrimaryVolume
  EndVolume
EndTrack

LeadOut XA
  Empty 150
EndTrack

EndDisc
```

Then the image can be created using:

```
       buildcd (buildcd –s<id>:<partition> <.CTI filename>
```

Building On To A PlayStation Disk

To use the .XA created files on a gold disk use CD-Gen as normal, use the Directory View and the Put Files option to add the .XA files to the CD image. Then select the File Types button and select the <XA> option. This will set a small ‘s’ in the file icon in the directory view, indicating that the file is an .XA sub-headed file and will be burned onto the CD accordingly.



File Sourc
BREATH.XA;1 c:\ps
PSX.EXE;1 c:\ps
RE2.XA;1 c:\ps

File Type
( ) Standard File(s) [data file]
(•) Subheader File(s) (XA audio, STR with Audio)
( ) CDDA File(s)
Ok
Cancel

Code For Playing The .XA

As with DA there are a few operations that are needed to make full use of the .XA file. These include the ability to play and stop the audio playback as well as some initialisation functions.
The full source for this example is on the developers web site:
https://www-s.playstation.co.uk/site2/ftp/developer/Sample_Code/samples/xatut.zip

File Initialisation

Before playing the .XA the files on the CD need to be initialised. In this example we only need to (1)check the .XA files are present and (2)calculate a start and end position on the CD. Note that if you are using a CD emulator the SECTOR_SIZE must be changed to 2336 as the CD emulator incorrectly returns the full XA sector size rather than a standard sector size of 2048 bytes.

NOTE: As from buildcd version 2.42 this SECTOR_SIZE bug is fixed so the SECTOR_SIZE can always be referred to as being 2048 bytes

```
#ifdef CD
#define SECTOR_SIZE 2048
#else
#define SECTOR_SIZE 2336
#endif

for(i=0;i<2;i++)
        {
        if(CdSearchFile(&fp, theXAFile[i].filename) == 0)
        {
                printf("%s: not found\n", theXAFile[fileNo].filename);
               return;
        }
        // get CD file start position
                theXAFile[i].startpos = CdPosToInt(&fp.pos);

        // get CD file end position, start pos + number of sectors -1
        theXAFile[i].endpos = theXAFile[i].startpos + (fp.size/SECTOR_SIZE) -1;
```

.XA Initialisation

All that is needed to set up .XA playback is a quick change of the mode. CdlModeSF specifies that sub-header filtering operation is on to allow the controller to play the correct audio channel and CdlModeSize1 is used so that we can track the sector number that is being played. This allows us to detect when the end of the sample is reached. A callback also needs to be hooked onto the CDReadyCallback(). This callback is detailed further on in the article.

```
        param[0] = CdlModeSpeed|CdlModeRT|CdlModeSF|CdlModeSize1;
        CdControlB(CdlSetmode, param, 0);
        CdControlF(CdlPause,0);

        return CdReadyCallback((CdlCB)cbready);
```

Shutdown

Again very simple, all that is needed is to reset the CD mode back to double speed only. As the CD mechanism is already at double speed no slow mechanical operation is required and the CD system just switches to no sub-header filtering.

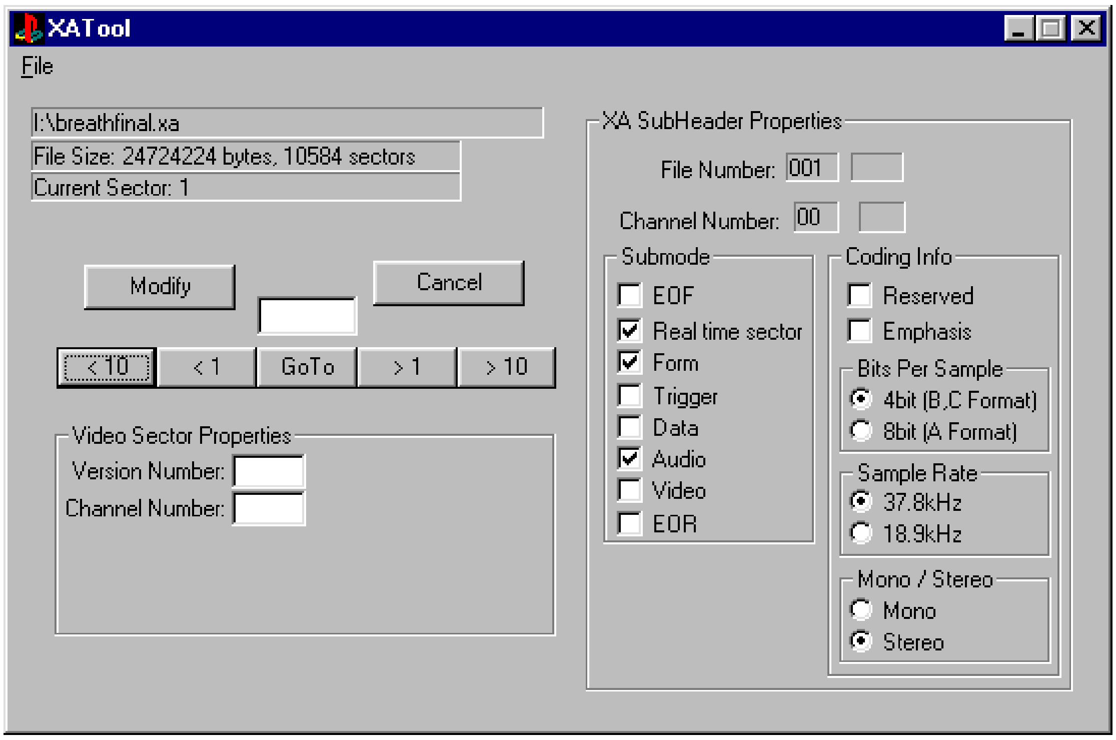
```
        CdControlF(CdlPause,0);
        param[0] = CdlModeSpeed;
        CdControlB(CdlSetmode, param, 0);
```

Playing

The playback of .XA involves the setting up of a filter; this specifies the file and channel number to actually read data from. The file and channel number are contained within the .XA sub-header information, which can be viewed on .XA files you have created that are still on your PC hard disk using XATOOL.EXE. This program can be found on the web site together with the source code for the example.

https://www-s.playstation.co.uk/site2/ftp/developer/Sample_Code/samples/xatut.zip

In this case viewing the .XA file shows that only one sector in every eight has both a channel number and a file number defined , the other sectors are null data sectors.



I:\breathfinal.xa
File Size: 24724224 bytes, 10584 sectors
Current Sector: 1

Video Sector Properties
Version Number:
Channel Number:

XA SubHeader Properties
File Number: 001
Channel Number: 00

Submode
- [ ] EOF
- [✓] Real time sector
- [✓] Form
- [ ] Trigger
- [ ] Data
- [✓] Audio
- [ ] Video
- [ ] EOR

Coding Info
- [ ] Reserved
- [ ] Emphasis

Bits Per Sample: [● 4bit (B,C Format)] 8bit (A Format)
Sample Rate: [● 37.8kHz] 18.9kHz
Mono / Stereo: Mono [● Stereo]

We need to set a filter to tell the CD system which sub-header to use, therefore our code needs to set a CdlFilter structure to file number one and channel zero. Once set we can play the .XA using CdlReadS using a start position of the file's location on the CD.

```
        CdlLOC  loc;
        CdlFILTER theFilter;

        theFilter.file=1;
        theFilter.chan=0;
        CdControlF(CdlSetfilter, (u_char *)&theFilter);

        // Starting position on CD
        CdIntToPos(startPos, &loc);
        CdControlF(CdlReadS, (u_char *)&loc);
```

Stopping

In this case the easiest way to stop the CD is to calculate the position of the end of the file and pass this in as the end position when choosing to play the sample. In the initialisation function for the .XA playback a CD callback can be set, which will occur every data sector. In this case seven out of eight sectors are data sectors. In the callback we can obtain the position of the CD as CdlModeSize1 returns the current sector number of the CD. Therefore within the callback all we simply do is retrieve this value and compare it against the calculated end position of the sample.

```
        if (intr == CdlDataReady)
        {
                CdGetSector((u_long*)buffer,585);
                currentPos = CdPosToInt( (CdlLOC *)buffer );
                if(currentPos >= theXAFile[fileNo].endpos)
                {
                      SsSetSerialVol(SS_SERIAL_A,0,0);
                      CdControlF(CdlPause,0);
                }
        }
```

Summary

Utilising .XA files this way is not the most effective use of the format, since for comparable quality, the converted file will be far larger. However the technique does give some benefits over CD-DA if the slight drop in quality is acceptable. In particular it allows you to switch quickly between files, and between playing music and loading. This is due to the fact that the CD motor speedup/slowdown is not used. It is also faster to seek between .XA samples than move from track to track using .DA.

Part 2 - Using Interleaved .XA Files

Various audio samples can be interleaved together to create an .XA file that acts as a sample bank. Basically this involves filling the empty data sectors that were shown in the first example with other useful data such as other audio samples.

There are two methods of building interleaved .XA which affect the code involved in playback:

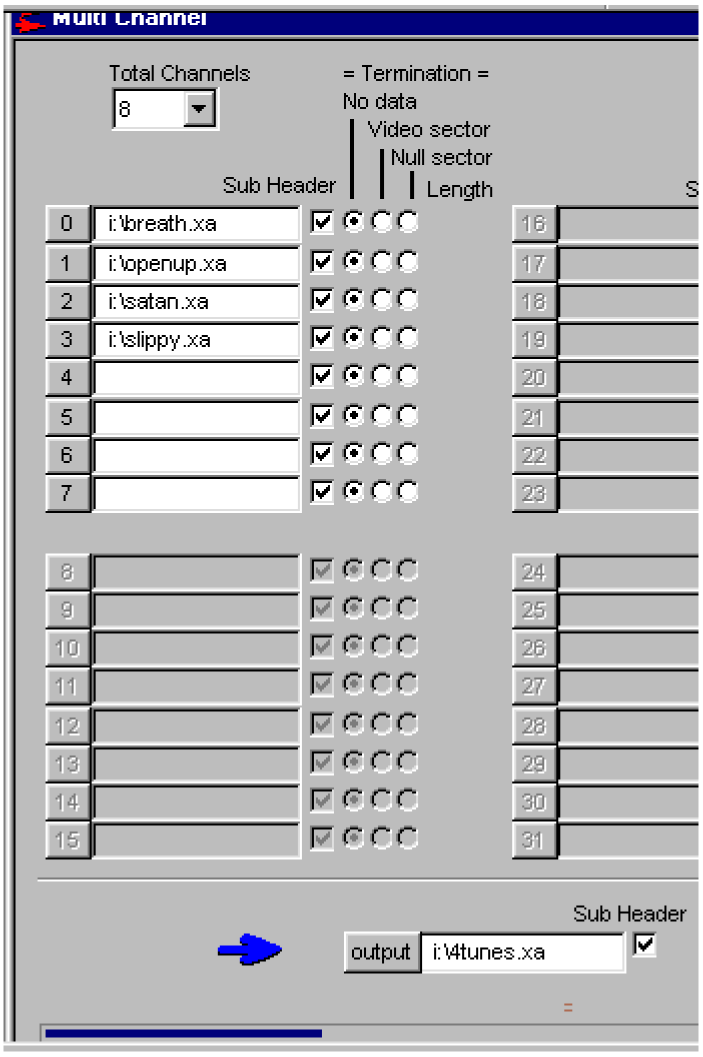
Method 1 – Interleaving using a data channel. This is just a move up from the original example, with more channels taken up with audio data.

Method 2 – Filling all channels with audio, tracking end of sample with either video sectors or data sectors. If you prefer not to have a wasted data channel but fill all the channels with audio then this is the option for you.

Method 1 - Creation

The initial samples can be built as in the first example, for this example four PCM,44.1 kHz, 16 bit. Stereo samples are to be converted from .WAV files to 37.8 kHz, Stereo .XA files.

The following interleaved file is then created, with the properties described above we still have four channels spare which we could fill with other audio tracks, as long as at least one channel is left blank for null data.



Total Channels: 8

		Sub Header	Termination
0	i:\breath.xa	✓	● ○ ○
1	i:\openup.xa	✓	● ○ ○
2	i:\satan.xa	✓	● ○ ○
3	i:\slippy.xa	✓	● ○ ○
4		✓	● ○ ○
5		✓	● ○ ○
6		✓	● ○ ○
7		✓	● ○ ○

output: i:\4tunes.xa Sub Header ✓

Method 1 - Building On To An Emulated Disk Image

This file can be built onto a CD image in the same way as part one, remembering to specify the file as an .XA source file:

```
        File 4TUNES.XA
           XASource i:\4tunes.xa
        EndFile
```

Method 1 - Building On To A PlayStation Disk

Burn a CD using CD-Gen as with the first example.

Method 1 - Code For Playing The .XA

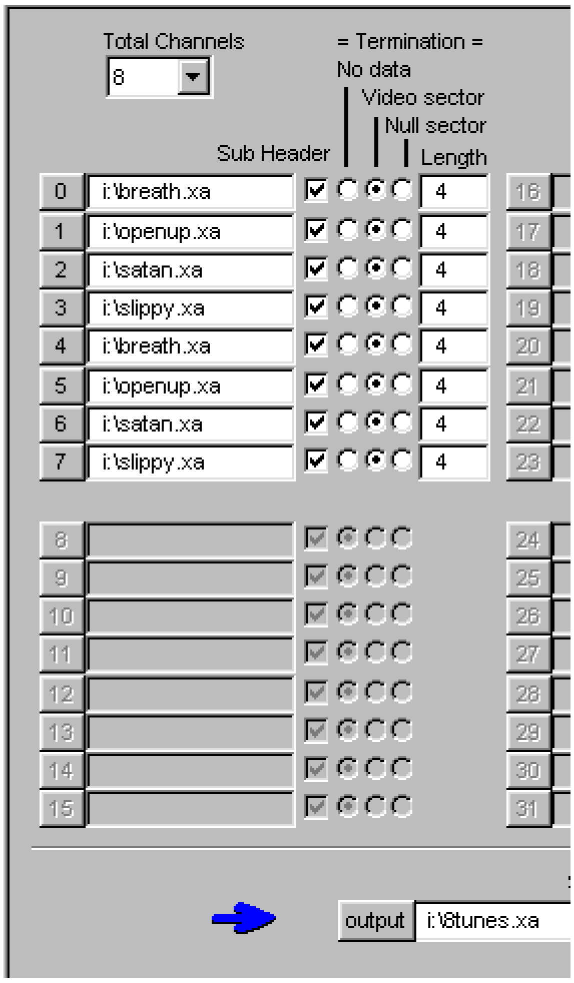
In this example all the tunes finish at the same time so we do not need to change the start and end positions, we only need to choose which channel to play. If the samples were of differing lengths we would need to calculate the lengths for each channel and pass the channel end position to the PlayXA() function.

The only other change is to add the setting of the .XA channel. This is achieved by modifying PlayXA() slightly so that the channel number is passed in rather than hard coded to be zero.

```
void PlayXA(int channel, int startPos, int endpos)
{

        CdlLOC  loc;
        CdlFILTER theFilter;

         // set volume
         SsSetSerialVol(SS_SERIAL_A,127,127);

        theFilter.file=1;
        theFilter.chan=channel;
        CdControlF(CdlSetfilter, (u_char *)&theFilter);

        // Starting position on CD
        CdIntToPos(startPos, &loc);
         currentPos=startPos;
        CdControlF(CdlReadS, (u_char *)&loc);
        return;
}
```

Method 2 - Creation

The initial samples are once again kept the same as the above example, however this time all the channels are filled with audio tracks. This time the channels are terminated by video sectors rather than null data, as below:



Total Channels 8

		Sub Header	No data	Video sector	Null sector	Length	
0	i:\breath.xa	✓		●		4	16
1	i:\openup.xa	✓		●		4	17
2	i:\satan.xa	✓		●		4	18
3	i:\slippy.xa	✓		●		4	19
4	i:\breath.xa	✓		●		4	20
5	i:\openup.xa	✓		●		4	21
6	i:\satan.xa	✓		●		4	22
7	i:\slippy.xa	✓		●		4	23
8		✓	●				24
9		✓	●				25
10		✓	●				26
11		✓	●				27
12		✓	●				28
13		✓	●				29
14		✓	●				30
15		✓	●				31

output i:\8tunes.xa

Method 2 - Building On To An Emulated Disk Image

This file can be built onto a CD image in the usual way.

Method 2 - Building On To A PlayStation Disk

Burn a CD using CD-Gen in the usual way.

Method 2 - Code For Playing The .XA

As the file no longer contains any data channels the CD system will not return a data sector until the end of the sample, where it will return the video sector that terminates the sample. If the samples are of different sizes then the shorter samples will return their data sectors while the longer samples are playing. Therefore to detect the end of the sample we need to check which video sector channel number is being returned within the data sector. The only code modification required is in the initialisation and the CDCallBack. Also for ease of use I have made the channel number global to allow testing in the callback.

Initialisation

As we no longer need to track the sector number the CD mode needs to be set appropriately:

```
        param[0] = CdlModeSpeed|CdlModeRT|CdlModeSF;
        CdControlB(CdlSetmode, param, 0);
        CdControlF(CdlPause,0);

        return CdReadyCallback((CdlCB)cbready);
```

Stopping

Now that the audio samples are terminated by video sectors we just need to check that the sector in the CD buffer is a video sector and then check to make sure it is the particular channel number we require.

```
void cbready(int intr, u_char *result)
{
        if (intr == CdlDataReady)
        {
                CdGetSector((u_long *)buffer,8);

                ID = *(unsigned short *)(buffer+3);

// video sector channel number format = 1CCCCC0000000001
                currentChannel = *((unsigned short *)(buffer+3)+1);
                currentChannel = (currentChannel&31744)>>10;

                // If this is a video sector then check that this is the channel
                // you want then stop playing the .XA sample
                if( (ID == 352) && (currentChannel == gChannel) )
                {
                        SsSetSerialVol(SS_SERIAL_A,0,0);
                                CdControlF(CdlPause,0);
                }

        }
}
```

Summary

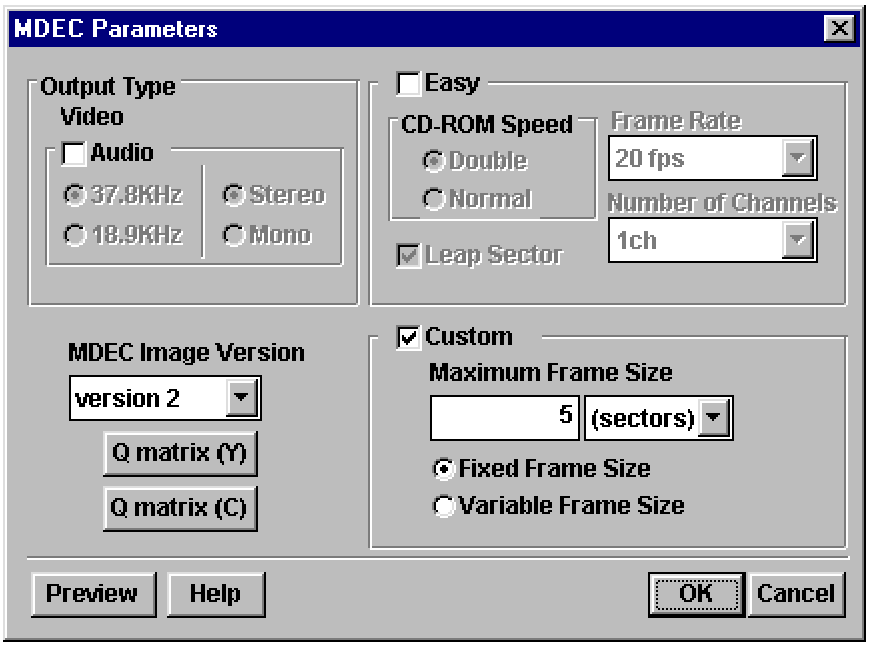
Because this technique allows you to quickly switch from sample to sample, as well as being able to detect when the sample has finished playing, it is possible to use this technique for real-time speech effects ranging from just one comment to combining comments to provide a sports style commentary.

Part 3 - Advanced Interleaving Techniques – Interleaving 2 Audio Tracks Together With One Video Stream

To save space on CD it is possible to interleave multiple audio tracks together within a video stream, this allows the programmer to specify which audio track to play when playing the video stream.

Creation

Create the sound samples as usual, using 37.8kHz for the frequency and stereo output. Then convert the .AVI that you wish to use as the video using the following parameters.



MDEC Parameters

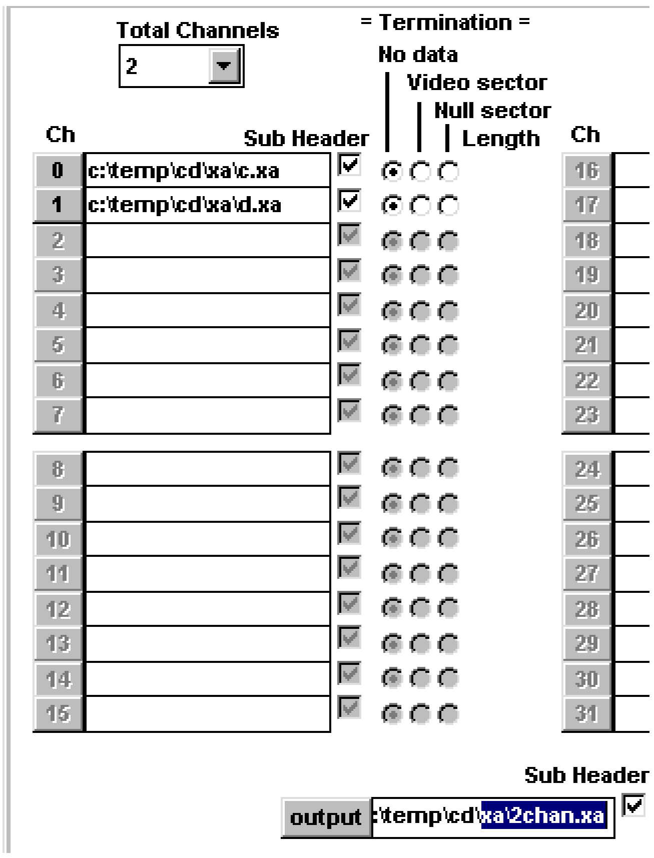
Output Type
Video
☐ Audio
● 37.8KHz ○ 18.9KHz
● Stereo ○ Mono

☐ Easy
CD-ROM Speed: ● Double ○ Normal
☑ Leap Sector
Frame Rate: 20 fps
Number of Channels: 1ch

MDEC Image Version: version 2

☑ Custom
Maximum Frame Size: 5 (sectors)
● Fixed Frame Size
○ Variable Frame Size

Take the two .XA audio files you have created and use the Multi-Channel option in MovConv or MovPack to interleave these files together using the following settings:

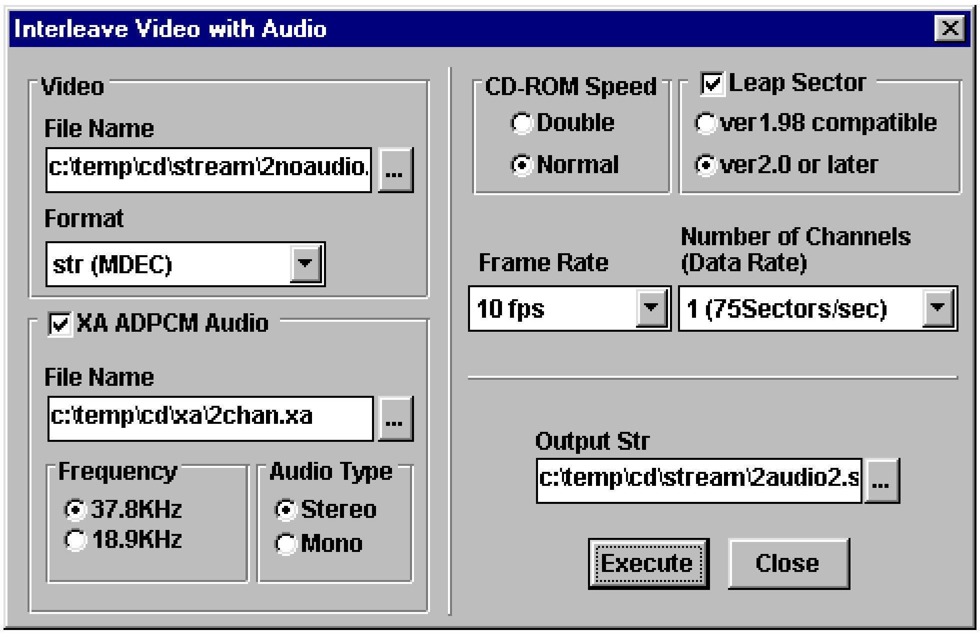

Total Channels: 2

= Termination =: No data / Video sector / Null sector / Length

Ch		Sub Header	No data	Video sector	Null sector	Ch
0	c:\temp\cd\xa\c.xa	✓	●	○	○	16
1	c:\temp\cd\xa\d.xa	✓	●	○	○	17
2		✓	●	○	○	18
3		✓	●	○	○	19
4		✓	●	○	○	20
5		✓	●	○	○	21
6		✓	●	○	○	22
7		✓	●	○	○	23
8		✓	●	○	○	24
9		✓	●	○	○	25
10		✓	●	○	○	26
11		✓	●	○	○	27
12		✓	●	○	○	28
13		✓	●	○	○	29
14		✓	●	○	○	30
15		✓	●	○	○	31

output: :\temp\cd\xa\2chan.xa Sub Header ✓

The interleaved audio can now be combined with the video footage using the 'Interleave Video With Audio' option from MovConv like so:


Interleave Video with Audio

Video
File Name: c:\temp\cd\stream\2noaudio.
Format: str (MDEC)

✓ XA ADPCM Audio
File Name: c:\temp\cd\xa\2chan.xa
Frequency: [✓ 37.8KHz] 18.9KHz
Audio Type: [✓ Stereo] Mono

CD-ROM Speed: Double [✓ Normal]
✓ Leap Sector: ver1.98 compatible [✓ ver2.0 or later]

Frame Rate: 10 fps
Number of Channels (Data Rate): 1 (75Sectors/sec)

Output Str: c:\temp\cd\stream\2audio2.s

Building On To An Emulated Disk Image And Gold CD

This stream file can be put onto both the CD emulator and a gold disk in the same way as a normal stream with audio file.

Modification Required For Standard Movie Code

As the file now contains two audio tracks, the audio track must be selected before playing the movie. For normal streams the audio channel filter number is set to one, however when creating streams using the above technique the audio channels are numbered zero and one. The code required to set the filter correctly is detailed below where channel can be zero or one. This can be set at the start of the video stream or alternatively while the stream is actually playing.

```
CdlFILTER theFilter;

theFilter.file=1;
theFilter.chan=channel;
CdControlF(CdlSetfilter, (u_char *)&theFilter);
```