Documentation

4X Movie®

4.2 version

4X Technologies ®

Copyright

« Dracula » All rights reserved. With the authorisation of the Index company.

« Paris by Ducati » All rights reserved. With the authorisation of the Reflex company.

« Evolva » All rights reserved.
With the authorisation of the Virgin Interactive company and the Computer Artworks company.

Content of this Manual

Installation – general information

As the 4X Movie compression tools are only available for PC, you must install the PC version whatever the target platform used (PC, Macintosh or Dreamcast).
If you use a Macintosh or Dreamcast version of 4X Movie, you must also install the libraries and source codes on the selected platform (Macintosh or Dreamcast). The installation procedures for each platform are explained in the following pages.

PC Installation

Insert the CD-ROM into the drive and wait while the set-up starts automatically. If the automatic program launch is not activated on your machine, start the installation by double-clicking on the Setup.exe file which you will find at the root of the CD-ROM.

During the installation, follow the instructions. It is better to install 4XMovie© in the directory proposed by default (to avoid confusing versions).

WARNING

4X Movie® 4.2 is not compatible with files produced with versions older than version 4.0.

Required configuration

Compression:	*PC type Pentium II 400 MHz with 128 Meg RAM.*
Decompression:	*PC type Pentium 166 MHz minimum with 32 Meg RAM, a graphics card which can display at least 32,768 colours.*

Operating systems

Windows 95/98
Windows 2000
Windows NT 4

DirectX and DirectXMedia

In order to use the player by default, you need at least DirectX 3.
If the player does not work on your machine (with Windows 95 for example), you will find the installation of DirectX7 on the CD-ROM in the DirectX directory.
If you use Windows NT, you must have at least ServicePack5 on your machine. If this is not the case, you can obtain ServicePack6 at the following address:
http://www.microsoft.com/ntserver/nts/downloads/recommended/SP6/allSP6.asp

To use the compressor, you will need the latest version of DirectXMedia.
If you don't have it, you will find the installation for it on the CD-ROM in the DirectX directory.

Example files

The 4X Movie® example files ***are not copied on to your hard disk****. They are accessible on the CD via the shortcuts "sample 1" and "sample 2" in the menu of your Windows desktop:*

Start → Programs → 4X Technologies → 4X Movie 4.2	*→ sample 1*
… …	*→ sample 2*

Documentation

The documentation for this product is in PDF format and is found in the menu of your Windows desktop in:

Start → Programs → 4X Technologies → 4X Movie 4.2 → Documentation

If you do not have Acrobat Reader for reading PDF files, you will find it on the 4X Movie CD in the directory \\Acrobat\acrbtrdr.exe at the root.
You can also download it from the Adobe site at the following address: http://www.adobe.fr/products/acrobat/readermain.html

Macintosh Installation

Since the 4X Movie compression tools are only available for PC, you must install the PC version regardless of the platform to be used (PC, Macintosh or Dreamcast).

To use the player, copy the StuffIt file 4xm_play.exe.sit which you will find in the MAC/player directory of the CD into a directory on your hard disk.
Once it is copied, decompress it with the StuffIt software by double-clicking on this file. A 4xm_play.exe file will be created; you can use it by double-clicking on it.

To use the SDK, copy the StuffIt 4xm_sdk.sit file which you will find in the MAC/SDK directory of the CD into a directory on your hard disk.
Once it is copied, decompress it with the StuffIt software by double-clicking on this file. A tree structure will be created which includes all of the SDK files. You will find here the inclusion files as well as the libraries. You will also have the source codes of the player as a use example.

Machine configuration required

Compression:	*PC type Pentium II 400 MHz with 128 Meg RAM.*
Decompression:	*Macintosh G3 with 32 Meg RAM.*

Operating system

Mac0s 8

Dreamcast Installation

Since the 4X Movie compression tools are only available for PC, you must install the PC version whatever the platform to be used (PC, Macintosh or Dreamcast).

Copy the files contained in the Dreamcast directory onto the hard disk of your PC.
This tree structure contains the include files, libraries, and the code of the player in examples provided in C and C++.

4X Movie®

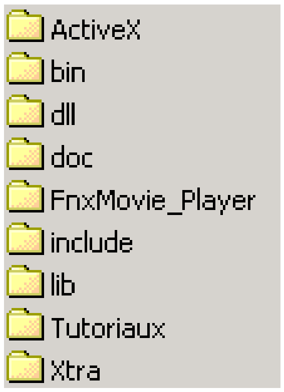
General Information

The 4X Movie development kit comprises the following modules :

- *4XM - SDK library : this allows low-level access for the control and operating functions of 4X Movie (available on PC, Mac and Dreamcast).*
- *4X Movie library (only on PC) : this allows simplified (thus faster) access to operating and control functions of 4 X movie and for the system, graphic and audio layers on PC. The input code of the FNX MOVIE player are supplied as examples.*
- *The tools : several tools are supplied with the 4XMOVIE development kit*

4XM_COMP	*Video compression module*
4XM_PLAY	*Software package for reading compressed 4XM files*
4XM_ SOUND	*Interlacing software for 4XM soundtracks*
FnxMovie_Player	*Software for reading compressed 4XM files, created on the basis of the FNX library, with input codes given as examples*
4XMovieManager	*Interface which includes the above-mentioned tools*
Xtra	*Xtra Director7 for PC and MAC*
ActiveX	*For integrating the decompression of the 4XM file into Intranet applications (e.g.: Internet Explorer)*

Directory treeing


ActiveX
bin
dll
doc
FnxMovie_Player
include
lib
Tutoriaux
Xtra

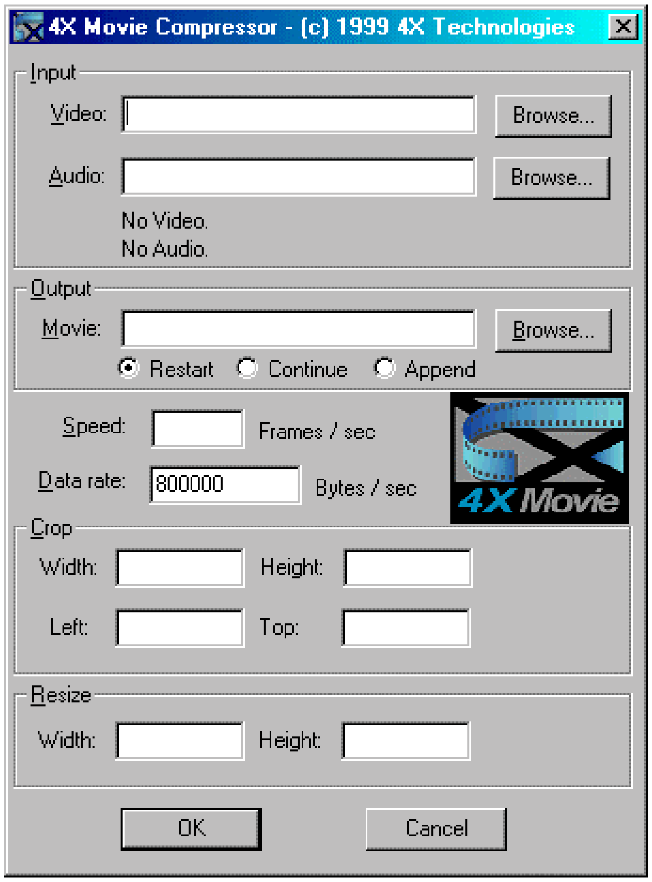
ActiveX	*Files needed for an Intranet application for PC*
bin	*Tools, player and 4XMOVIE programs*
dll	*DLL file. for FNX MOVIE and 4XM_SDK libraries*
doc	*Documentation files*
include	*.H files for FNX MOVIE and 4XM_SDK*
lib	*.LIB file for FNX MOVIE and 4XM_SDK*
FnxMovie_Player	*Example input files of the use of the FNXMOVIE library*
Tutoriaux	*Resource files for the tool use tutorial*
Xtra	*Files needed to use Xtra PC*

The tools

4XM_COMP	*Video compression module (PC)*
4XM_PLAY	*Software package for reading compressed 4XM files (PC, Mac, Dreamcast)*
4XM_ SOUND	*Interlacing software for 4XM soundtracks (PC)*
4X_BATCH	*Software for compression by batch*
FnxMovie_Player	*Software for reading compressed 4XM files, created on the basis of the FNX library, with input codes given as examples (PC)*
4XMovieManager	*Interface which includes the above-mentioned tools*
Xtra	*Xtra Director7 (PC, Mac)*
ActiveX	*For integrating the decompression of the 4XM file into Intranet applications (e.g.: Internet Explorer) (PC)*

Location (by default) : C :\Program Files\4X Technologies\4X Movie 4.2\bin

4XM_Comp



No Video.
No Audio.

Restart [✓] Continue Append

Speed: Frames / sec
Data rate: 800000 Bytes / sec

Crop — Width: Height: Left: Top:

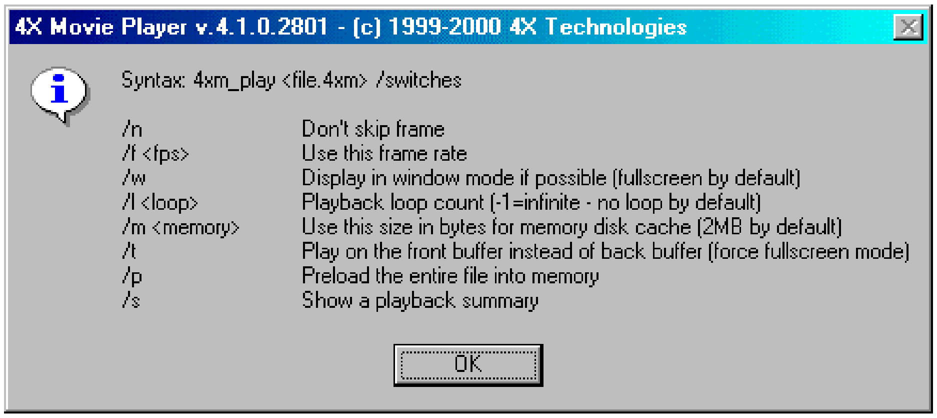
Resize — Width: Height:

Video Input :	*Enter the name of the video input file to be compressed.*
Audio Input :	*Enter the name of the audio file to be compressed. The audio file may be included in the original video file. In this case, enter the same name in each field.*
Movie Output :	*Enter the name and path of the compressed file to be created (4XM)*
Restart Output :	*Default option : any file having the same name will be deleted and replaced by the new file.*
Continue Option :	*Allows resumption of compression after an interruption. In this case, you must enter the name of the 4XM file created during the first phase of compression.*
Append Output :	*Allows you to add the compression to be performed to an existing 4XM file. Enter the name of the 4XM file to which you*

intend to add a new sequence. This sequence is then necessarily added after the existing file.

Speed (frames/sec)	*Specify the number of images per second in your video recording. This parameter is only used when reading the 4XM file in order to fix the running speed for the images.* *NB : 4XM_Comp does not set a time scale.* *If you have used a video recording running at 25 images/second and you enter 15 in the « Speed » box, 4XM_Comp will compress all the images contained in the input video, but will play the compressed file at a speed of 15 frames/sec, producing a slow-motion effect.*
Data Rate	*Defines the throughput in bytes per second. The default value for this parameter is 800 000 i.e. 780 Kb per second corresponding to a CDX8 drive. You should count about 100 Kb per second for a single-speed drive. This calculation takes into account both images and sound. If there is no sound it is for the images alone.*
Crop	*This option allows you to cut the video sequence to desired dimensions. Do not enter anything in this box unless you wish to change the size of the video sequence that is to be compressed.*
Crop Width	*Width in number of pixels of the window to be compressed*
Crop Height	*Height in number of pixels of the window to be compressed*
Crop Left	*X coordinates in pixels of the window to compress, starting from the left-hand side of the image*
Crop Top	*Y coordinates, in pixels, from the top of the image, of the window to compress.*
Resize	*This option allows enlargement or reduction of the size of the video to be compressed. Enter nothing here unless you wish to modify the size of the video that is to be compressed. The Crop and Resize functions may be used simultaneously. The Crop function will then be performed first and afterwards the rest of the video will be reduced or enlarged to the dimensions specified by the Resize function.* *This function uses bilinear filtering algorithms which may substantially increase compression times.*
Resize Width	*New width - in pixels - of the video to be compressed.*
Resize Height	*New height - in pixels - of the video to be compressed*

4XM_Play


4X Movie Player v.4.1.0.2801 - (c) 1999-2000 4X Technologies
OK

Syntax: 4xm_play <file.4xm> /switches

```
/n            Don't skip frame
/f <fps>      Use this frame rate
/w            Display in window mode if possible (fullscreen by default)
/l <loop>     Playback loop count (-1=infinite - no loop by default)
/m <memory>   Use this size in bytes for memory disk cache (2MB by default)
/t            Play on the front buffer instead of back buffer (force fullscreen mode)
/p            Preload the entire file into memory
/s            Show a playback summary
```

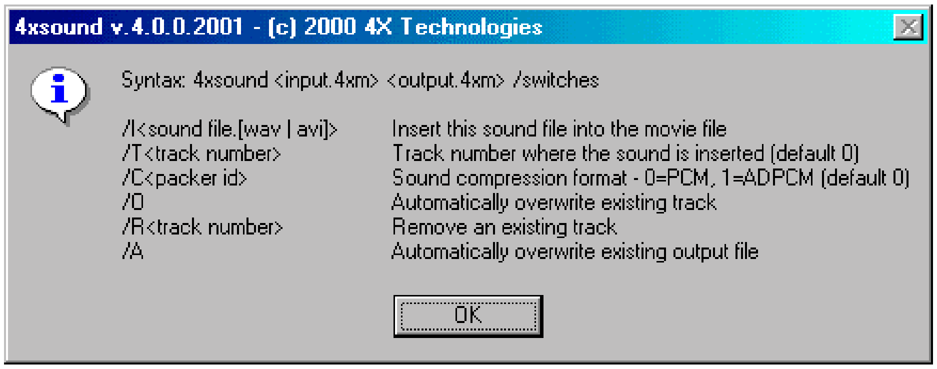
4XM _Play allows a compressed video to be played back by using 4XM_Comp.

You can use 4XM _Play either in command mode or by placing the icon of the videosequence to play back on the 4XM _Play icon (drag and drop)

To use 4XM _Play in command mode, refer to the parameters given in the dialog box shown above.

If there are several soundtracks in the file, you can use the F1 key during playback to change tracks.

4Xm_Sound



```
Syntax: 4xsound <input.4xm> <output.4xm> /switches

/I<sound file.[wav | avi]>    Insert this sound file into the movie file
/T<track number>              Track number where the sound is inserted (default 0)
/C<packer id>                 Sound compression format - 0=PCM, 1=ADPCM (default 0)
/O                            Automatically overwrite existing track
/R<track number>              Remove an existing track
/A                            Automatically overwrite existing output file
```

4Xsound enables you to add or delete a soundtrack in a 4XM file. Thus, you can create 4XM video files containing several soundtracks. This option is frequently resorted to for the creation of multilingual productions.

4Xsound is used in command mode.

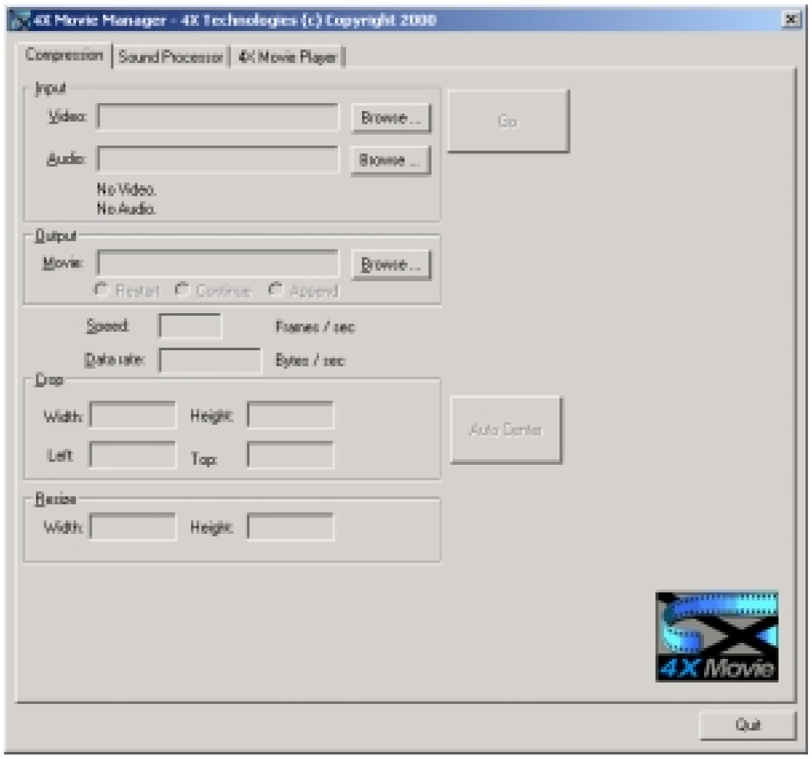
Syntax :

4Xsound_input file .4XM output file .4XM/switches

input file. .4XM	*Enter the name and where necessary the path of the original .4XM file*
output file .4XM	*Enter the name and, where necessary, the path of the .4XM file resulting from the addition or deletion of soundtracks. The name must be different from the input file.*
Sound file.avi	*File containing the soundtrack to be added. This file may be in either AVI or WAV format. e.g. : /I speak.wav*
/T track number	*Number of the audio track (default number : 0). As the 4X format can contain several audio tracks it is indispensable to number each one so that the player (4XM_Play) can identify them. By default, track 0 will be played. Example : /T 3*
/C packer.id	*Gives the sound compression format required for the inserted track. 0 = PCM, 1 = ADPCM. PCM is the default format*
/O	*Automatically overwrites the previous track if the track number selected belongs to a track that already exists in the 4XM input file.*
/R track number	*Deletes the track indicated. Example : /R 2 will delete track 2 in the 4XM input file.*
/A	*Replaces the output 4XM file selected if it already exists.*

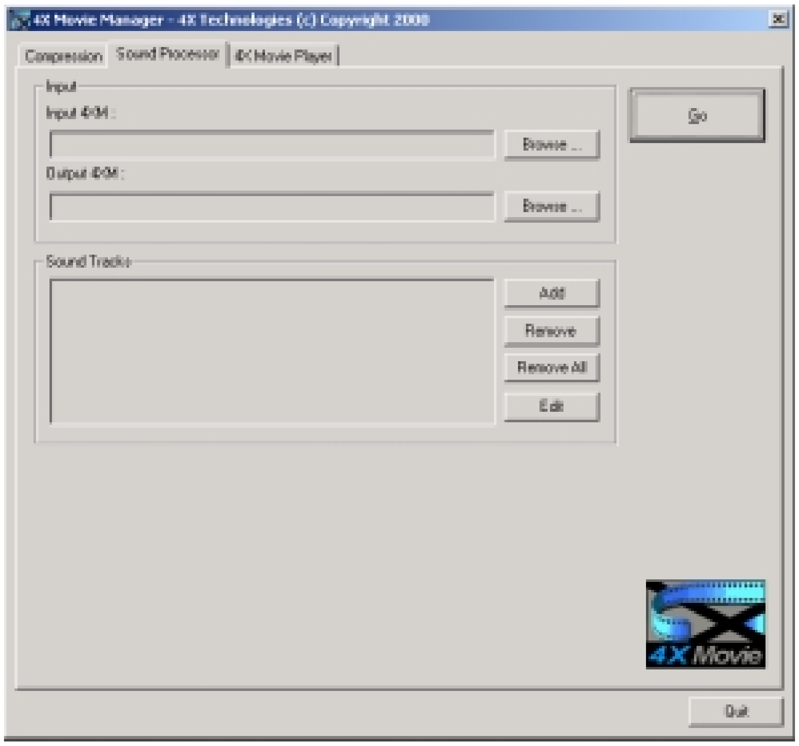
4XMovieManager

Compressor



With the interface contained in this tab, you can compress your files in 4X Movie format.
For an explanation of the settings, see the chapter on the 4xm_comp.exe utility.

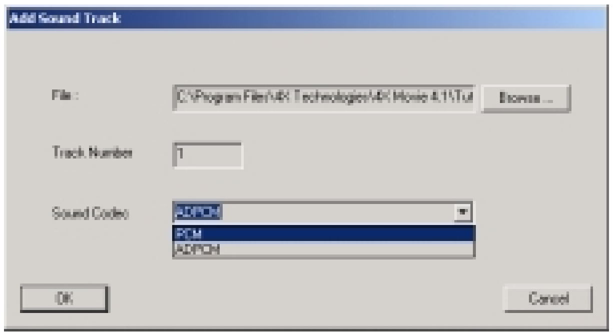
Sound Processor



With the interface included in this tab, you can adjust the audio tracks contained in the 4X Movie files more easily than with the DOS interface offered by the 4xm_sound.exe utility.

With this interface, you can do all of the operations offered by the 4xm_sound.exe utility.

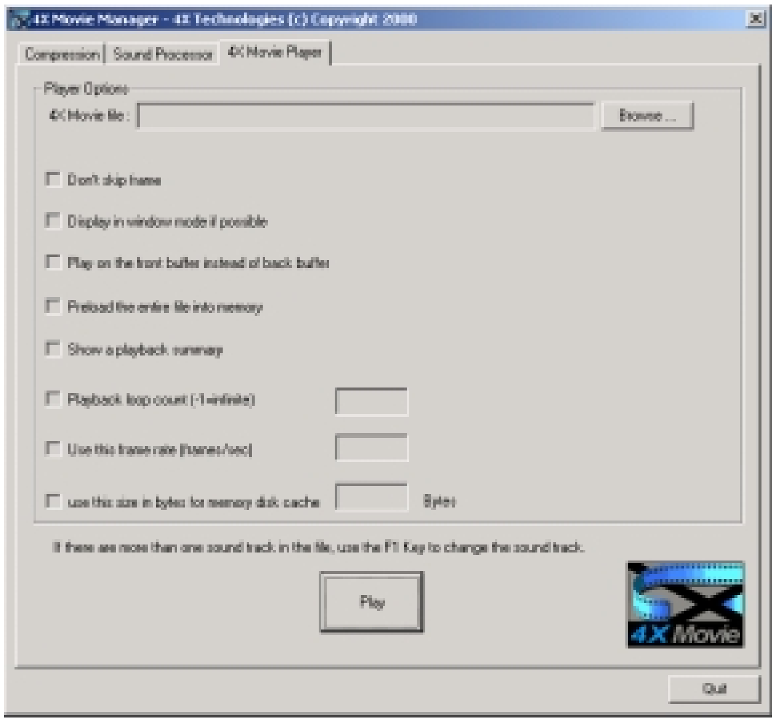
If you want to add a sound track, a second interface will appear.



File : C:\Program Files\4X Technologies\4X Movie 4.1\Tut
Track Number 1
Sound Codec ADPCM

This lets you choose the sound file, the number of the track, as well as the type of codec to use for the sound file. The file can be put in the 4X Movie file without being compressed (PCM), or it can be compressed before integration (ADPCM).

4X Movie Player



Player Options

4X Movie file :

☐ Don't skip frame

☐ Display in window mode if possible

☐ Play on the front buffer instead of back buffer

☐ Preload the entire file into memory

☐ Show a playback summary

☐ Playback loop count (-1=infinite)

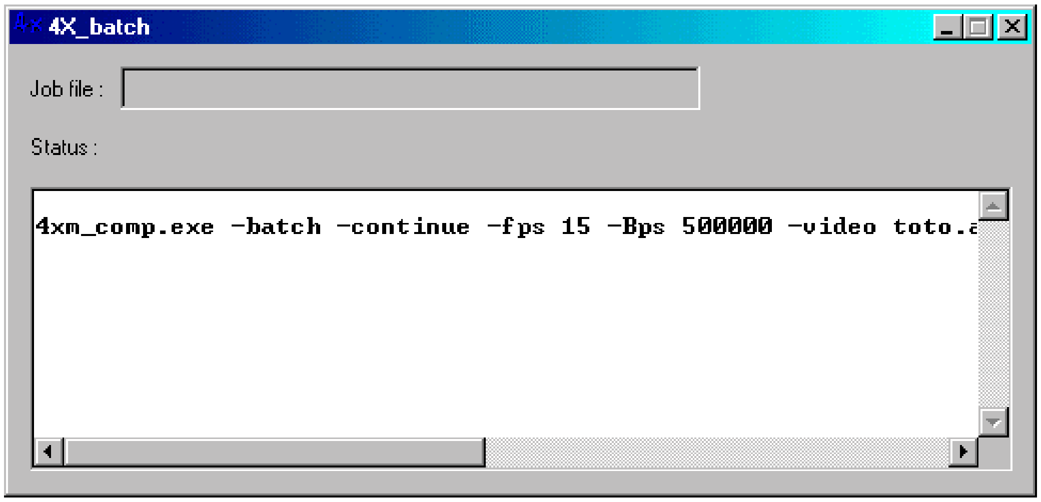
☐ Use this frame rate (frames/sec)

☐ use this size in bytes for memory disk cache Bytes

If there are more than one sound track in the file, use the F1 Key to change the sound track.

With the interface included in this tab, you can play your 4X Movie files.
For an explanation of the settings, see the chapter on the 4xm_play.exe utility.

4X_Batch



```
4xm_comp.exe -batch -continue -fps 15 -Bps 500000 -video toto.a
```

This tool allows optimization of compression times by using the resinputs available on a Local Area Network.

1. *Place the 4X_Batch.exe, 4X_Batch.txt, and 4XM_Comp.exe files in the same directory on a device that is visible to the whole network (compression server).*
2. *Edit the 4X_Batch.txt compression file according to the syntax defined hereafter.*
3. *From each workstation available for compression, run the 4X_Batch.exe program on the « compression server » device.*

Syntax

Edit the compression script 4X_Batch.txt using Notepad or Wordpad and enter the list of files to compress, following the syntax of the following example :

```
4xm_comp.exe -batch -continue - fps 15 - Bps 500000 - video toto.avi -out toto.4Xm -sound
toto.avi - crop 10, 10, 640, 480, -resize 320, 240
```

4xm_comp.exe	*Run the compression program*
-batch	*compulsory parameter for use of the « batch » mode*
-continue	*allows you to resume compression after an interruption. You must then enter the name of the 4XM file created during the initial phase of compression.*
-restart	*Any file bearing the same name will be deleted and replaced by the new file.*
-append	*Enables you to add the new compression to be performed to an existing one. The name of the 4XM file to which you*

	want to add a new sequence must be entered. This new sequence is then entered after the existing one.
-fps 15	Enter the number of images per second of your videosequence. This parameter is only used when reading the 4XM file in order to determine the image running speed. NB : 4XM_Comp does not set a time scale. If you have used a videorecording running at 25 images/second and you enter 15 in the « Speed » box, 4XM_Comp will compress all the images contained in the input video, but will play the compressed file at a speed of 15 images/second, producing a slow-motion effect.
-bps 500000	Defines the throughput in bytes per second. The example indicates 500 000 i.e. about 480 Kb per second corresponding to a CD x 4 drive. You should count about 100 Kb per second for a single-speed drive.
-video toto.avi	Enter the name of the video input file to compress.
-sound toto.4XM	Enter the name of the compressed file to be created. (.4XM)
-audio toto.avi	Enter the name of the audio file to compress. The audio file may be included in the original video file and in this case the same name must be entered in both fields.
-crop 10,10,640,480	-crop x,y, width, height
-resize 320,240	resize width, height.

ActiveX

The Active X « 4X MOVIE PLAYER »

This component is simply the standard 4X Movie player presented as an ActiveX controller
Developers can thus easily integrate it into their Windows application in order to obtain high-quality video.

Applications that accept ActiveX :

- *Internet Explorer*
- *Macromedia Director*
- *Borland Delphi*
- *Visual Basic*
- *Visual C++*

Installation

Files
The ActiveX controller runtime of the 4X MOVIE Player comprises of the following files :

- *4xmovie.ocx*
- *4Xm_sdk.dll*

WARNING: for ActiveX to work, you must copy 4xm_sdk.dll into the Windows system directory or in the same directory as the executable of your application (e.g.: for Director, copy it into the same directory as director.exe).

Registration
The ActiveX controller is made visible by all applications through the following instruction :

`regsvr32.exe` *installation path***`\4xmovie.ocx`**

The ActiveX controller
`regsvr32.exe /u` *installation path* **`\4xmovie.ocx`**

Controls

Properties

ScenePath
This must be the path for a .4xm file. The path may be absolute or relative in relation to the current directory.

ReadyState
Informs whether the state of the controller is ready

Events
ReadyStateChange
Triggered when the ReadyState property has changed

EndOfStream
Triggered when the « movie » is finished

Class ID
8D1AB0C1-F021-11d3-BADC-525400E7CEDF

Xtra for macromedia Director

Installation

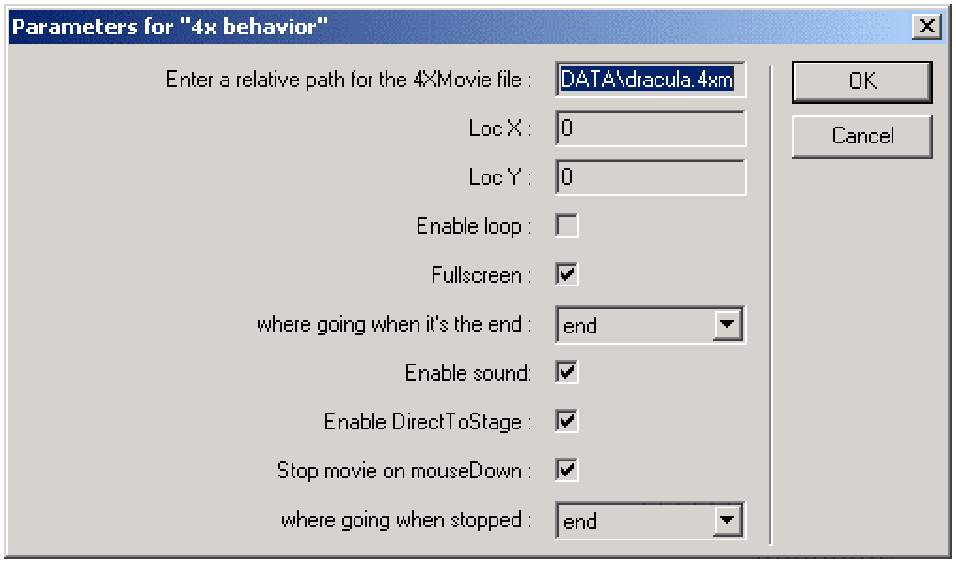
Files

These files are to be copied into the Xtras directory of Macromedia Director :

- *FourXMovie.x32*
- *4xm_sdk.dll*

Behaviour Management

Once Xtra has been incorporated into the Director Score window, you can open the « Behaviours » window thanks to the « Behaviour Inspector » icon.



Parameters for "4x behavior"

Enter a relative path for the 4XMovie file :	DATA\dracula.4xm
Loc X :	0
Loc Y :	0
Enable loop :	☐
Fullscreen :	✓
where going when it's the end :	end
Enable sound:	✓
Enable DirectToStage :	✓
Stop movie on mouseDown :	✓
where going when stopped :	end

Path : *first of all, you must specify the relative path of the 4XM to be played*
LocX,locY : *coordinates for the upper left-hand point of the video*
EnableLoop : *forces the video to loop once the last image decompressed*
Fullscreen : *Enables you to play the video fullscreen*
Where going when it's the end : *you can specify the behaviour of the Xtra when the video finishes*
Enable Sound : *Enables you to activate or disactivate the sound during the video sequence*
EnableDirectToStage : *Allows you to specify whether DirectX is used or not (valid only for PCs, must be entered on Mac)*
Stop Movie on mouseDown : *Enables you to stop the video with a click on the mouse*
Where going when stopped *: Lets you specify the behaviour of the application at the end of the movie.*

TUTORIALS

Tutorial 1 compression of a movie

In this tutorial we are going to try to rebuild the Step1.4xm file which is in the directory :

Installation directory / Tutorials / Step1.4xm

- After installing 4xMovie you will find in the Start Program /4X Technologies/4X Movie menu, a shortcut to the compressor (4X Movie Compressor). Start it
- I*n the first field, get the file Step1.avi » (which you will find in Installation directory / Tutorials / Step1.avi) by using the « browse » button*
  The compressor must automatically detect whether there is a video and an audio track in the file and must automatically fill in the data on the audio track
- *Then, with the « browse » button, indicate the path where the compressor must write the output 4X Movie file.*
- *Next, in the « data-rate » field, place as value 500 000 bps*
- *You are then ready to start compression and may click on the « OK » button.*

The compression of this video should take about 10 minutes and 20 seconds on a Celeron 450 MHz with 128 Mb Ram, i.e. about 25 seconds of compression time per image.

TUTORIAL 2 : Introduction of an extra sound track

In this tutorial we are going to try and rebuild the Step2.4xm file which is in directory :

Installation directory / Tutorials / Step2.4xm

In the previously mentioned directory you will find a file called Step2.bat. This file enables you to generate file Step2.4xm which we want to rebuild.
Let's analyze the content of this file.

```
..\bin\4xm_sound.exe ducati13s.4xm step2.4xm /Istep2.wav /T1 /C1
```

- *We call the tool enabling handling of sound tracks :* \bin\4xm\sound.exe
- *The input file which we want to modify : ducati13s.4xm*
- *Then the output file (which must be different from the input file) : step2.4xm*
- *Then the options : /Istep2.wav /T1 /C1*
  - *Option /I allows you to specify the input sound file*
  - *Option /T allows you to specify the number of the track which is being modified*
  - *Option /C allows you to specify the type of compression that you wish to perform on this audio track.*

If you start the Step2.bat file a such, a dialog box will ask whether you wish to overwrite the Step2.4xm file. If you answer yes, it must create exactly the same file.

FNXMOVIE.H et 4XM_SDK.h

Handling of 4X Movie files

Both « Open-file » methods enable you to create the 4XMOVIE file handler and to open a file of this kind.

```
bool Open_File (char *Filename,XM_OVERLOAD *overload=0);
```

This allows the creation of an XM_HANDLE and enables you to open the 4X MOVIE file from a storage medium (CD ROM or HARD DISK)

```
bool Open_File (void *memory,unsigned int size,XM_OVERLOAD *overload=0);
```

This allows the creation of An XM_HANDLE and enables you to open the 4X MOVIE file from a memory segment specified by the user.
Memory : pointer to memory segment
Size : size of memory segment

```
struct XM_OVERLOAD
{
       float   fps;           // use this playback frame rate in frame per
                              //second (0=use default)
       unsigned int   page_count;    // use "page_count" 2KB pages for the disk cache
                              //- default 1024 pages ie 2MB - (0=use default)
       int     loop;          // loop the playback "loop" times (-1=infinite)
       bool    skip;          // boolean witch indicate if we can skip some
                              //frame while playback (true by default)
       bool    no_background_read;
       // boolean witch indicate we don't want to use background reading (false by default)
};
```

*The **XM_overload** structure which is described in the 4xm-sdk.h file allows you to define specific performance characteristics for decompression of the film.*

- *The « fps » field allows you to define a specific frame-rate. If the value is « 0 », the rate discovered in the file will be used.*

- *The « page-count » field allows you to specify the size of the readout cache for the file.*
  If « 0 » is entered, the cache size will be 2Mb.
  The minimum cache is 256 Kb (2 x128 Kb).
  The maximum size is 4096 Kb (2 x 2048 Kb)

- *The « loop » field is used to specify the number of times you wish to play back the film. The default value is 0 (it is played once ; if the value is 1, it is played back once, therefore twice in all). If « -1 » is entered, the film is played indefinitely.*

- *The « skip » field allows you to state whether you wish all the images of the film to be displayed when the field is on « false » (this can slow down the « frame-rate » on some hardware) or to « skip » images when there is not enough time to display them, in order to keep the frame-rate of the film when the field is on « true ».*
- *The «no_background_read» field allows you to state whether you wish to read the datas from the disk with a Thread or to let them read with the default process. (The Thread reading will generally be faster than the other one, except on some very low-end configurations)*

```
void Close_File (void);
```

« Close_file » enables you to close the active 4X MOVIE file and to destroy the 4X MOVIE handle.

```
void Destroy_Handle (void);
```

« Destroy_Handle » enables to get the all memory free for the 4XM handling.

```
XM_HANDLE * Get_Xm_Handle (void);
```

« Get_XM_Handle » lets you retrieve the handle for the 4X MOVIE file.

```
XM_ECODE       Restart(void);
```

With « Restart » it is possible to reopen the 4X MOVIE file and restart compression from scratch.

There is an also error code : XM_ECODE which can take the following values :

XM_ECODE_OK	*= the operation has been correctly performed*
XM_ECODE_FAIL	*= it has failed*

```
const XM_STANDARD_HEADER * Get_Standard_Header(void) const;
```

This procedure : « Get_Standard_Header » allows you to retrieve the 4X MOVIE file heading. An XM_STANDARD_HEADER structure is sent back containing complete information on the file.

```
const char *   Get_Name(void) const;
```

Enables you to retrieve the name of the movie, if there is one.
Output : a pointer to the movie name or NULL.

```
const char *   Get_Desc(void) const;
```

Enables you to obtain a description of the movie, if there is one.
Output : a pointer to the description or NULL.

```
unsigned int Track_Count(unsigned int track_type=XMD_TRACK_ALL);
```

Supplies the number of « track_type » tracks
Input : the desired track type (option)
Output : the number of tracks

```
const XM_TRACK *      Get_Track_Info(unsigned int track_number,unsigned int
track_type=XMD_TRACK_ALL) const;
```

Obtains information on a specific track numbered « track_number » and of the tracktype indicate by the instruction « track_type »
Input : the track number
the track « family » (option)

```
const char * Get_Track_Name(unsigned int track_number,unsigned int track_type=XMD_TRACK_ALL)
const;
```

Obtains the name of the track, if there is one.
Input : the track number
the track « family » (option)
Output : a pointer to the track name or NULL

```
const char * Get_Track_Desc(unsigned int track_number,unsigned int track_type=XMD_TRACK_ALL)
const;
```

Gives a description of a track, if there is one.
Input : the track number
the track « family » (option)
Output : a pointer to the description or NULL

```
bool Unpack_Video_Track(unsigned int track_number,int x,int y,int width,int height,int
pitch,void *surface,XM_SURFACE_TYPE type);
```

Decompresses a given video track.
Input :

- *the number of the video track ;*
- *horizontal X axis reference of the upper left-hand crop point*
- *vertical Y axis reference of the upper left-hand crop point*
- *width of the part of the image that you wish to blit into the surface*
- *height of the part of the image which you want to blit in the area*
- *width of a line across the surface (Scan Line)*
- *the surface that you wish to blit into*
- *the type of surface*

Output :

- *A boolean function indicating whether or not the blit has actually been performed (it depends on the time taken).*

```
unsigned int Sound_Packet_Size(unsigned int track_number);
```

Obtains the size of the current sound packet for a designated soundtrack
Input : the number of the soundtrack
Output : the size of the packet

```
void Unpack_Sound_Track(unsigned int track_number,unsigned char *buffer0,int size0,unsigned
char *buffer1,int size1);
```

Decompresses a designated soundtrack.
Input :

- *the number of the soundtrack*
- *a pointer to the first buffer where the data must be placed*
- *the size of the first buffer. If the size is smaller than the size of the sound packet, the second buffer will be used*
- *a pointer to the second buffer where data can be placed. If it is not necessary to use this buffer, its value must be null*
- *the size of the second buffer. If it is not necessary to use this buffer, its size must be equal to 0*

```
void Enable_Track(unsigned int track_number,unsigned int track_type);
```

Permits internal decompression of a track.
CAREFUL : you may not authorize decompression of a single video track from the first frame of the film.
There are no such constraints for the other kinds of track.
Input : track number
track type

```
void Disable_Track(unsigned int track_number,unsigned int track_type);
```

Forbids internal decompression of a track.
Input : track number
track type

```
bool Is_Track_Enable(unsigned int track_number,unsigned int track_type);
```

Informs whether a given track is decompressed, internally or not.
Input : track number
track type
Output : a boolean function indicating whether the track is decompressed or not.

```
unsigned int Get_Frame(void);
```

Indicates the current frame of the movie.
Output : the current frame

```
Bool Next_Frame(void);
```

Decompression of the next frame
Output :

- *a boolean function indicating whether there is a next frame.*

```
Bool Ready(void);
```

Indicates whether the decompression engine is ready to decompress the next frame (not to be used to play back movies at full speed)
Output : boolean function indicating whether the engine is ready.

```
void Pause(void);
```

Either causes a pause on the movie or causes it to continue.

```
bool Is_Paused(void);
```

Answers whether or not the movie is paused
Output :

- *a Boolean function stating whether the movie is paused or not.*

```
void Summary(XM_SUMMARY *summary);
```

Fills in a summary structure
Input :

- *a pointer to the summary structure*

Macros definitions

Packer ID type for video tracks

```
XMD_VIDEO_4XMOVIE_16_RGB_565
XMD_VIDEO_4XMOVIE_32_RGB_8888
XMD_VIDEO_4XMOVIE_12_YUV_411
```

Packer ID type for soundtracks

```
XMD_SOUND_4XMOVIE_PCM
XMD_SOUND_4XMOVIE_ADPCM
```

Track types

```
XMD_TRACK_VIDEO
XMD_TRACK_SOUND
XMD_TRACK_SUBTITLE
XMD_TRACK_UNKNOWN
XMD_TRACK_ALL
```

Surface type. Internally, we use a 565 format surface, which is thus faster

```
XM_SURFACE_RGB_555
XM_SURFACE_RGB_565
XM_SURFACE_INVALID
```

Standard title heading for 4X Movie

```
struct XM_STANDARD_HEADER
```

Standard information on 4X Movie tracks

```
struct XM_TRACK
```

Specific information on 4X Movie video tracks

```
struct XM_VIDEO_TRACK:public XM_TRACK
```

Specific information on 4X Movie soundtracks

```
struct XM_SOUND_TRACK:public XM_TRACK
```

4X Movie decompression information

```
struct XM_SUMMARY
```

MANAGEMENT OF THE AUDIO DEVICE (Sound Peripheral)

Macro definitions for sound formats

```
XMD_MONO
XMD_STEREO
XMD_22KHZ
XMD_44KHZ
XMD_8BITS
XMD_16BITS
XMD_24BITS

bool Create_Sound  (HWND hwnd,
                         unsigned int channel_count=XMD_MONO,
                         unsigned int sample_rate=XMD_22KHZ,
                         unsigned int bits_per_sample=XMD_16BITS);
```

Creation of the audio device manager
Input :

- *a pointer to a Windows window (which may be created by FNX MOVIE thanks to the Operating System Manager).*
- *the number of audio channels*
- *the number of samples per second*
- *the number of bits per sample*

Output :

- *Sends back a boolean function (true if successfully performed, false if not).*

```
void Destroy_Sound (void);
```

Destroys the audio device manager

```
unsigned int add_Track (unsigned int channel_count=XMD_MONO,
                           unsigned int sample_rate=XMD_22KHZ,
                           unsigned int bits_per_sample=XMD_16BITS);
```

Adds a soundtrack.
Input :

- *number of channels*
- *the frequency*
- *the number of bits*

Output :

- *Answers « 0 » if unsuccessful, otherwise supplies the size of the secondary buffer.*

```
void   Lock_Sound     (unsigned int sound_size,
                           void **ptr1,unsigned long *size1,
                           void **ptr2,unsigned long *size2,
                           unsigned int num_player=0);
```

Locks the secondary buffer for a given track and returns the pointers for the secondary buffer.
Input :

- *amount of space to be locked in secondary buffer.*
- *the first pointer to the buffer*

- *size of the first pointer*
- *second pointer to the buffer (only to be filled if the first is close to the end of the buffer, which is circular).*
- *size of the second buffer (0 if the second buffer is NULL)*
- *number of track player*

```
void    Unlock_Sound     (void *ptr1,unsigned long size1,
                             void *ptr2,unsigned long size2,
                             unsigned int num_player=0);
```

Unlocks the secondary buffer for a given track player
Input :

- *first pointer to the buffer*
- *size of the first pointer*
- *second pointer to the buffer (only to be filled in if the first is close to the end of the buffer, which is circular).*
- *size of the second buffer (0 if the second buffer is NULL)*
- *number of track player*

```
void    Play_Sound          (unsigned int num_player=0);
```

Plays the track of a given player
Input : number of the track player

```
void    Stop_Sound          (unsigned int num_player=0);
```

Stops a give track player
Input : number of the track player

```
unsigned int Next_Sound_Frame   (unsigned int sound_size,unsigned int num_player=0);
```

After the writing of a sound frame, it allows pointers to the secondary buffer to be repositioned for the next frame.
Input :

- *size of the packet than has just been written into the secondary buffer*
- *number of the track player*

Output : number of the next sound frame to be written to

```
unsigned int Sound_Track_Count  (void);
```

Gives the number of the track player
Output : number of the track player

Management of the Graphic Device

```
Bool Create_Screen (unsigned int width,
                      unsigned int height,
                      unsigned int bpp,
                      HWND hwnd, bool fullscreen);
```

Creation of the Graphic Device Manager
Input :

- *width of images*
- *height of images*
- *number of bits per pixel*
- *a pointer to an Operating System window*
- *a boolean function to state whether you wish to display fullscreen or in a window (true = fullscreen, false = in a window)*

Output : sends back a boolean function (true if successful, otherwise false)

```
void Destroy_Screen (void);
```

Destroys the Graphic Device Manager

```
bool Lock_Back_Screen     (void **Screen,int *Scan_Line);
```

Locks the back buffer of the peripheral
Input :

- *address of a pointer where the address of the buffer will be placed*
- *address of a whole number for the Scan Line (in number of columns) of the buffer*

Output : gives a boolean function (true if successful, false if not)

```
void Unlock_Back_Screen  (void *Screen);
```

Unlocks the back buffer of the peripheral
Input : a pointer to the back buffer which was modified

```
void Show_Back_Screen    (void);
```

Enables the back buffer to be displayed. Flips between the front and back buffers

```
void Clear_Back_Screen   (unsigned long color);
```

Clears the screen with a specified colour
Input : colour for the clear

```
bool Lock_Front_Screen   (void **Screen,int *Scan_Line);
```

Locks the front buffer of the peripheral
Input :

- *address of a pointer where the address of the buffer will be placed*

- *address of a whole number for the Scan Line (in number of columns) of the buffer*

Output : gives a boolean function (true if successful, false if not)

```
void Unlock_Front_Screen (void *Screen);
```

Unlocks the front buffer of the peripheral
Input : a pointer to the front buffer which was modified

```
void Clear_Front_Screen  (unsigned long color);
```

Clears the screen with a specified color
Input : colour for the clear

```
void Set_Screen_Mode (unsigned int width,unsigned int height,unsigned int bpp);
```

Used to change the resolution of the peripheral
Input :

- *width*
- *height*
- *number of bits per pixel*

XM_SURFACE_TYPE Get_Surface_Type(void);

```
void Print (char *Texte,int x,int y,int Scan_Line, XM_SURFACE_TYPE Type,unsigned int
Color,void *Buffer);
```

Allows you to write a string of characters into a buffer
Input :

- *character string*
- *X axis in the buffer where you wish to write the string*
- *Y axis in the buffer where you wish to write the string*
- *buffer Scan Line*
- *type of surface returned by the previous function*
- *color of characters*
- *destination buffer*

```
unsigned int GetScreenWidth (void);
```

Allows you to retrieve the number of columns on the screen
Output : number of columns

```
unsigned int GetScreenHeight (void);
```

Allows you to retrieve the number of lines on the screen
Outpour : the number of lines

```
void Restore_Surface (void);
```

Allows you to restore the screens surfaces

OPERATING SYSTEM MANAGEMENT

```
bool Create_Window (HINSTANCE hInstance,char *appname, char *title,int x, int y, int width,int height);
```

Creation of a new system window.
Input :

- *Instance supplied by the operating system*
- *Name of the application*
- *Title of the window*
- *X axis of the window*
- *Y axis of the window*
- *width*
- *height*

```
void Destroy_Window (void);
```

Destroys the window

```
int* Get_Key (void);
```

Allows you to obtain a map of the keyboard
Output :

- *Table of 256 whole numbers (if a key has been used, the entry in the ASCII equivalence table is 1)*

```
void Center_Window (void);
```

Allows you to center the window

```
void Show_Window (void);
```

Allows you to display the window

```
HWND Get_Window (void);
```

Allows you to obtain the pointer to the window
Output : pointer to the system window

```
void SetMsgCallback (LRESULT (*callback)(HWND hWindow,UINT uMsg,WPARAM wParam,LPARAM lParam));
```

Enables to give the OS message manager callback.
Input : - pointer to the function managing the OS messages.

```
const char *Get_Last_Error(void) const;
```

Allows you to obtain the last error message triggered off
Output : the last error message

SDK PC : FnxMovie_Player.CPP

Introduction

FNX MOVIE is an API which uses the 4X MOVIE SDK.
This API enables you to use - in a simplified way - all the functions of 4xm_sdk as they are described in the 4xm_sdk.h.
With FNXMOVIE you can handle layers « OS », « GRAPHIC DEVICE » and « SOUND DEVICE »

FNXMOVIE allows all these operations through an FNXMOVIE type class.

In order to create this class, you must use the function :

```
FNXMOVIE *Xm_FnxMovie_New() ;
```

and to destroy such a class, use :

```
Xm_FnxMovie_Delete(FNXMOVIE *fnxmovie) ;
```

Within this class you have 4 types of method :

- *methods enabling you to manage 4XM files ;*
- *methods enabling you to manage the graphic peripheral ;*
- *methods enabling you to manage the sound (audio) peripheral ;*
- *methods enabling you to handle the interface with the Operating System.*

Description of the decompression and environment management procedure with FnxMovie

Initializations

```
fnxmovie = Xm_FnxMovie_New()
fnxmovie->Open_File(lpszCmdLine);
xm_handle=fnxmovie->Get_Xm_Handle();
xm_handle->Enable_Track(0,XMD_TRACK_VIDEO);
nb_track_sound = xm_handle->Get_Track_Cound(XMD_TRACK_SOUND);
xm_handle->Enable_Track(track_number,XMD_TRACK_SOUND);
fnxmovie->Create_Window(hInstance,"4XM_PLAY","4X Movie",0,0,640,480))
fnxmovie->SetMsgCallback(win_msg);
XM_VIDEO_TRACK * video = (XM_VIDEO_TRACK *) xm_handle->Get_Track_Info (0,XMD_TRACK_VIDEO);
fnxmovie->Create_Screen(video->width,video->height,16,fnxmovie->Get_Window(),false)
XM_SOUND_TRACK * sound = (XM_SOUND_TRACK *) xm_handle->Get_Track_Info
(track_number,XMD_TRACK_SOUND);
```

```
fnxmovie->Create_Sound(fnxmovie->Get_Window(),sound->channel_count,sound->sample_rate,sound-
>bits_per_sample))
fnxmovie->Show_Window();
```

Message Management Loop

```
LRESULT win_msg(HWND hWindow,UINT uMsg,WPARAM wParam,LPARAM lParam)
{
    PAINTSTRUCT                  ps;
    switch (uMsg)
    {
       case WM_PAINT:
           BeginPaint(hWindow,&ps);
           EndPaint(hWindow,&ps);
           if (fnxmovie)
           {
              fnxmovie->Restore_Surface();
              fnxmovie->Clear_Back_Screen(0x00000000);
              fnxmovie->Show_Back_Screen();
              fnxmovie->Clear_Back_Screen(0x00000000);
           }
           break;
    }
    return 0;
}
```

Main Loop

```
if (fnxmovie->Lock_Back_Screen(&Screen,&Scan_Line))
{

       frame_blit=xm_handle->Unpack_Video_Track(0,0,0,
                                                 640,480,
                                                 Scan_Line,
                                                 Screen,
                                                 fnxmovie->Get_Surface_Type());

       fnxmovie->Unlock_Back_Screen(Screen);

       if (frame_blit)
           fnxmovie->Show_Back_Screen();
}

if (nb_track_sound)
{
       sound_size=xm_handle->Sound_Packet_Size(track_number);
       fnxmovie->Lock_Sound(sound_size,(void **)&ptr1,&size1,(void **)&ptr2,&size2);
       xm_handle->Unpack_Sound_Track(track_number,ptr1,size1,ptr2,size2);
       fnxmovie->Unlock_Sound(ptr1,size1,ptr2,size2);
       sound_frame=fnxmovie->Next_Sound_Frame(sound_size);
}

Gestion_Message();

while (!xm_handle->Ready());

if (!xm_handle->Next_Frame())
 break;
```

End of Process

```
       fnxmovie->Close_File();
       fnxmovie->Destroy_Sound();
       fnxmovie->Destroy_Screen();
       fnxmovie->Destroy_Window();
       fnxmovie->Destroy_Handle();
       Xm_FnxMovie_Delete(fnxmovie);
```

SDK Macintosh : 4XM_SDK.h

4X Movie does not handle display and sound. The 4X Movie library only decompresses files in 4X Movie format.
The source code of the 4xm_play program is provided as an example however.

SDK Macintosh : 4xm_play

Initializations

```
xm_handle=Xm_New();

overload.fps=0.f;
overload.loop=0;
overload.page_count=0;
overload.skip=true;
xm_handle->Open((void *)&reply.sfFile,&overload);

if (!Get_Video_Track_Resolution(xm_handle,width,height,swidth,sheight))
        return 0;

xm_handle->Enable_Track(0,XMD_TRACK_VIDEO);

if (xm_handle->Track_Count(XMD_TRACK_SOUND))
{
        xm_handle->Enable_Track(0,XMD_TRACK_SOUND);
        Init_Sound((XM_SOUND_TRACK *)xm_handle->Get_Track_Info(0,XMD_TRACK_SOUND));
}
```

Main Loop

```
xm_handle->Unpack_Video_Track(0,0,0,width,height,pitch,(void*)&buffer[x+y*(pitch/2)],
XM_SURFACE_RGB_8888);


sound_size=xm_handle->Sound_Packet_Size(0);
Sound_Lock(sound_size,(void **)&ptr1,&size1,(void **)&ptr2,&size2);
xm_handle->Unpack_Sound_Track(0,ptr1,size1,ptr2,size2);
if (sound_frame==2)
        err=SndPlayDoubleBuffer(sound_channel,&sound_header);

sound_frame++;


while (!xm_handle->Ready());

if (!xm_handle->Next_Frame())
        break;
```

End of process

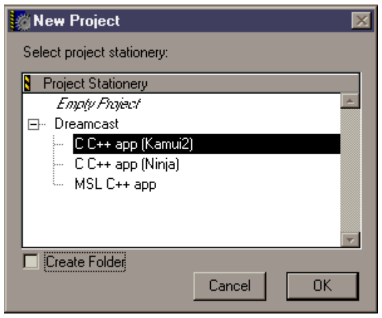
```
xm_handle->Summary(&summary);
xm_handle->Close();
Xm_Delete(xm_handle);
```

SDK Dreamcast : 4XM_SDK.h

4X Movie does not handle display and sound. The 4X Movie library only decompresses files in 4X Movie format.
The source code of the 4xm_play program is provided as an example however.

Create a 4X Movie project on Dreamcast (using Codewarrior)

1 . Open the IDE and make a new project


New Project

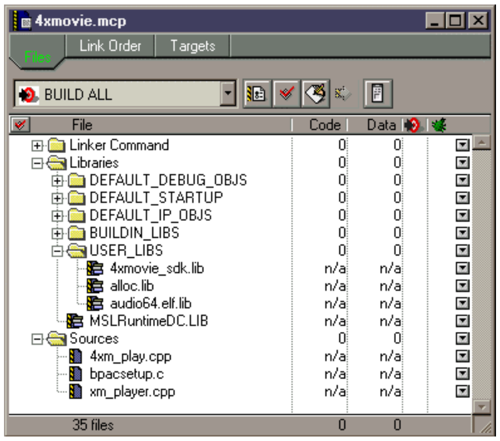
Select project stationery:

Project Stationery
- *Empty Project*
- Dreamcast
  - C C++ app (Kamui2)
  - C C++ app (Ninja)
  - MSL C++ app

[ ] Create Folder

Select C C++ app (Kamui2).
Deselect Create Folder.

2 . Files in the project


4xmovie.mcp

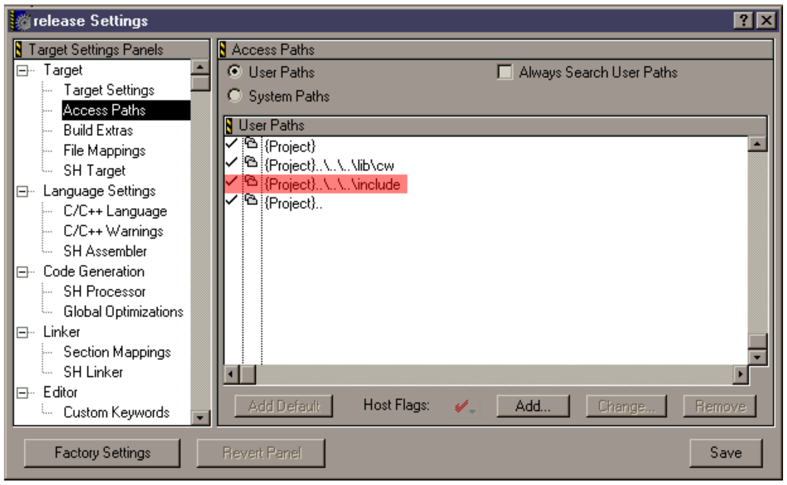
File	Code	Data
Linker Command	0	0
Libraries	0	0
DEFAULT_DEBUG_OBJS	0	0
DEFAULT_STARTUP	0	0
DEFAULT_IP_OBJS	0	0
BUILDIN_LIBS	0	0
USER_LIBS	0	0
4xmovie_sdk.lib	n/a	n/a
alloc.lib	n/a	n/a
audio64.elf.lib	n/a	n/a
MSLRuntimeDC.LIB	n/a	n/a
Sources	0	0
4xm_play.cpp	n/a	n/a
bpacsetup.c	n/a	n/a
xm_player.cpp	n/a	n/a
35 files	0	0

Remove the files Nindows.elf.lib (in USER_LIBS folder) and sbinit.c (in Sources folder) in all targets.

Drag and drop the files 4xm_play.cpp, bpacsetup.c and xm_player in Sources folder and validate all the targets (default options).
Drag and drop the files 4xmovie_sdk.lib and alloc.lib from <4XMOVIE_SDK dir>\lib in USER_LIBS folder and validate all the targets.
Drag and drop the files audio64.elf.lib from the SEGA SDK directory in USER_LIBS folder and validate all the targets.

3 . Extra paths

Open the settings windows for each target (icon next to the target list).



Access Paths

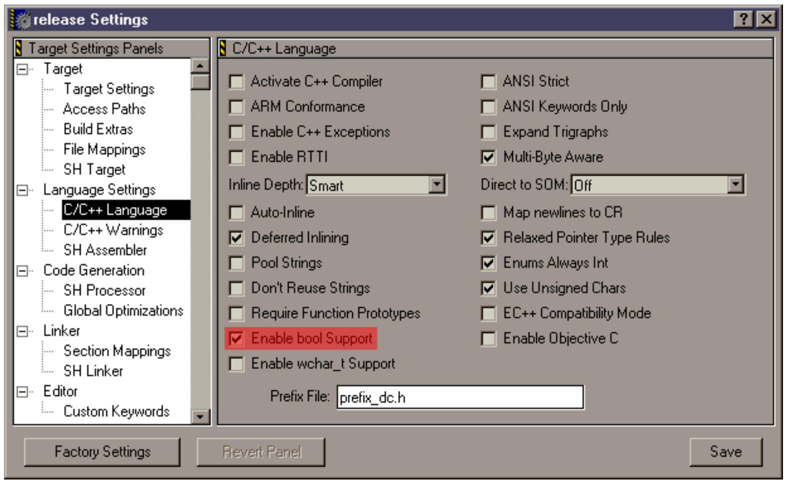
(●) User Paths
( ) System Paths
[ ] Always Search User Paths

User Paths:
- ✓ {Project}
- ✓ {Project}..\..\..\lib\cw
- ✓ {Project}..\..\..\include
- ✓ {Project}..

In all target, add the 4xmovie include path (<4XMOVIE_SDK dir>\include) in the access paths.
(You can remove the sbinit path).

4 . Extra options

You need to check <Enable bool Support>.

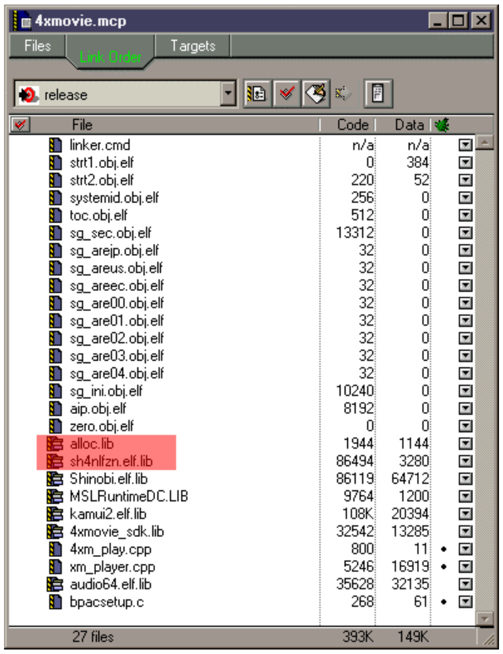


C/C++ Language

[ ] Activate C++ Compiler	[ ] ANSI Strict
[ ] ARM Conformance	[ ] ANSI Keywords Only
[ ] Enable C++ Exceptions	[ ] Expand Trigraphs
[ ] Enable RTTI	[✓] Multi-Byte Aware
Inline Depth: Smart	Direct to SOM: Off
[ ] Auto-Inline	[ ] Map newlines to CR
[✓] Deferred Inlining	[✓] Relaxed Pointer Type Rules
[ ] Pool Strings	[✓] Enums Always Int
[ ] Don't Reuse Strings	[✓] Use Unsigned Chars
[ ] Require Function Prototypes	[ ] EC++ Compatibility Mode
[✓] Enable bool Support	[ ] Enable Objective C
[ ] Enable wchar_t Support	

Prefix File: prefix_dc.h

5 . Link order



File	Code	Data	
linker.cmd	n/a	n/a	
strt1.obj.elf	0	384	
strt2.obj.elf	220	52	
systemid.obj.elf	256	0	
toc.obj.elf	512	0	
sg_sec.obj.elf	13312	0	
sg_arejp.obj.elf	32	0	
sg_areus.obj.elf	32	0	
sg_areec.obj.elf	32	0	
sg_are00.obj.elf	32	0	
sg_are01.obj.elf	32	0	
sg_are02.obj.elf	32	0	
sg_are03.obj.elf	32	0	
sg_are04.obj.elf	32	0	
sg_ini.obj.elf	10240	0	
aip.obj.elf	8192	0	
zero.obj.elf	0	0	
alloc.lib	1944	1144	
sh4nlfzn.elf.lib	86494	3280	
Shinobi.elf.lib	86119	64712	
MSLRuntimeDC.LIB	9764	1200	
kamui2.elf.lib	108K	20394	
4xmovie_sdk.lib	32542	13285	
4xm_play.cpp	800	11	•
xm_player.cpp	5246	16919	•
audio64.elf.lib	35628	32135	
bpacsetup.c	268	61	•
27 files	393K	149K	

Move the alloc.lib before the sh4nlfzn.elf.lib.
If you don't do that, the engine won't work because our own malloc system must be linked before the codewarrior's one.

6 . Build

The sample is ready and you can build it.
Launch the .elf files with Codescape.

SDK Dreamcast: 4xm_play

Version C++

In C++, a XM_PLAY class is also provided. It handles all communications with the sound and video peripherals, but it also decompresses 4X Movie files. The code of the main function is thus very simplified.

If you want to integrate the player in C++ in an application, use this class as an inspiration, either by using the peripherals declared within it, or by using yours. In the latter case, only the decompression part of the files interests you.

Initializations

Initialisation of the decompression mode.

```
void           Set_Config(XM_PLAYER_CONFIG *config)
{
    config->dont_skip_frame=false;
    config->force_fps=0.f;
    config->fullscreen=true;
    config->loop_count=-1;
}
```

Use of a call-back for all special functions which the user wants to add.

```
void                      Callback(XM_PLAYER_APP *xm_player)
{

}
```

The header of the main differs depending on the compiler used.

```
#ifdef __GNUC__
extern "C" int fnx_main(void)
#else
extern "C" int main(void)
#endif

xm_player=new XM_PLAYER_APP(&rpl_vertex_buffer,&kmsc);
```

Opening of the file and initialisation of the audio track.

```
if (xm_player->Open("matrix.4xm",config))
       return 1;

if (xm_player->Sound_Track_Count())
        xm_player->Enable_Sound_Track(0);
```

Main Loop

```
xm_player->Play(Callback);
```

End of process

The player application is closed and the memory is freed up

```
xm_player->Close(&summary);
delete xm_player;
```

Version C

Initializations

Initialisation of the decompression mode

```
void          Set_Config(XM_PLAYER_CONFIG *config)
{
    config->dont_skip_frame=false;
    config->force_fps=0.f;
    config->fullscreen=true;
    config->loop_count=-1;
    config->memory_cache_size=128; // we ask 128 pages (256Ko pre-buffer); default is 1024
}
```

Use of a call-back for all special functions which the user wants to add

```
void                     Callback(XM_PLAYER_APP *xm_player)
{

}
```

The header of the main differs depending on the compiler

```
#ifdef __GNUC__
int fnx_main(void)
#else
int main(void)
#endif
```

Creation of “the application player”

```
xm_player=New_XM_PLAYER_APP(&rpl_vertex_buffer,&kmsc);
```

Opening of the file and initialisation of the audio track

```
if (App_Open_File(xm_player,"matrix.4xm",&config))
        return 1;

if (App_Sound_Track_Count(xm_player))
        App_Enable_Sound_Track(xm_player,0);
```

Main Loop

```
App_Play(xm_player,Callback);
```

End of process

The player application is closed and the memory is freed up

```
App_Close(xm_player,&summary);
Delete_XM_PLAYER_APP(xm_player);
```