Defining speaker locations in multi-channel streams and WDM audio drivers using the WAVEFORMATEXTENSIBLE structure

Revision Number: 00.01.00

July 6, 2000

by Carsten Schulz, Emagic Soft- & Hardware GmbH

Introduction

With the introduction of the data structure **WAVEFORMATEXTENSIBLE** it is finally documented how to specify speaker locations for multi-channel audio. This new structure that is described in more detail at [1] looks as follows:

```
#define _WAVEFORMATEXTENSIBLE_
typedef struct {
    WAVEFORMATEX    Format;
    union {
        WORD wValidBitsPerSample;   /* bits of precision  */
        WORD wSamplesPerBlock;      /* valid if wBitsPerSample==0 */
        WORD wReserved;             /* If neither applies, set to zero. */
    } Samples;
    DWORD           dwChannelMask;  /* which channels are */
                                    /* present in stream  */
    GUID            SubF ormat;
} WAVEFORMATEXTENSIBLE, *PWAVEFORMATEXTENSIBLE;
```

In this documentation we are especially interested in only two of these fields: **nChannels** (contained in **WAVEFORMATEX**) and **dwChannelMask**.
nChannels specifies the number of individual interleaved channels [3] in a stream or the number of channels supported by an audio adapter.
The **dwChannelMask** field is used to specify the mapping of channels to spatial locations. The following bitmaps are currently defined (in **ksmedia.h** and **mmreg.h**, Windows 2000 DDK, May 10, 2000 WinHEC 2000, version) to describe the various speaker locations that could be used for the **dwChannelMask** field:

```
// Speaker Positions:
#define SPEAKER_FRONT_LEFT              0x1
#define SPEAKER_FRONT_RIGHT             0x2
#define SPEAKER_FRO NT_CENTER            0x4
#define SPEAKER_LOW_FREQUENCY           0x8
#define SPEAKER_BACK_LEFT               0x10
#define SPEAKER_BACK_RIGHT              0x20
#define SPEAKER_FRONT_LEFT_OF_CENTER    0x40
#define SPEAKER_FRONT_RIGHT_OF_CENTER   0x80
#define SPEAKER_BACK_CENTER             0x100
#define SPEAKER_SIDE_LEFT               0x200
#define SPEAKER_SIDE_RIGHT              0x400
#define SPEAKER_TOP_CENTER              0x800
#define SPEAKER_TOP_FRONT_LEFT          0x1000
#define SPEAKER_TOP_FRONT_CENTER         0x2000
#define SPEAKER_TOP_FRONT_RIGHT         0x4000
#define SPEAKER_TOP_BACK_LEFT           0x8000
#define SPEAKER_TOP_BACK_CENTER         0x10000
#define SPEAKER_TOP_BACK_RIGHT          0x20000

// Bit mask locations reserved for future use
#define SPEAKER_RESERVED                0x7FFC0000

// Used to specify that any possible permutation of speaker configurations
#define SPEAKER_ALL                     0x80000000
```

An audio driver could also use the following macros (from ksmedia.h) to specify a standard multi-channel speaker setup:

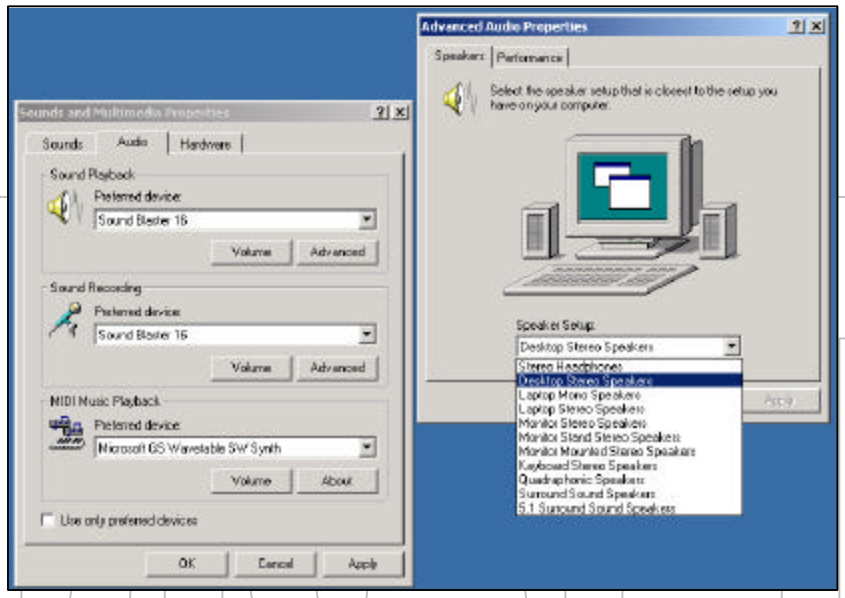
```
// DirectSound Speaker Config
#define KSAUDIO_SPEAKER_MONO     (SPEAKER_FRONT_CENTER)
#define KSAUDIO_SPEAKER_STEREO   (SPEAKER_FRONT_LEFT | SPEAKER_FRONT_RIGHT)
#define KSAUDIO_SPEAKER_QUAD     (SPEAKER_FRONT_LEFT | SP  EAKER_FRONT_RIGHT |  \
                                  SPEAKER_BACK_LEFT  | SPEAKER_BACK_RIGHT)
#define KSAUDIO_SPEAKER_SURROUND(SPEAKER_FRONT_LEFT | SPEAKER_FRONT_RIGHT |    \
                                  SPEAKER_FRONT_CENTER | SPEAKER_BACK_CENTER)
#define KSAUDIO_SPEAKER_5POINT1 (SPEAKER_FRONT_LEFT | SPEAKER_FRONT_RIGHT |    \
                                  SPEAKER_FRONT_CENTER | SPEAKER_LOW_FREQUENCY |    \
                                  SPEAKER_BACK_LEFT  | SPEAKER_BACK_RIGHT)
#define KSAUDIO_SPEAKER_7PO INT1 (SPEAKER_FRONT_LEFT | SPEAKER_FRONT_RIGHT |   \
                                 SPEAKER_FRONT_CENTER | SPEAKER_LOW_FREQUENCY |    \
                                 SPEAKER_BACK_LEFT | SPEAKER_BACK_RIGHT |    \
                                 SPEAKER_FRONT_  LEFT_OF_CENTER |
SPEAKER_FRONT_RIGHT_OF_CENTER)
```

The **nChannels** and **dwChannelMask** fields can be used both by a WDM audio driver [2] and by an application that creates an audio stream.
An audio driver would normally specify the number of output channels and the locations of the attached speakers.
An application would normally specify the number of channels in a stream and the desired locations for each channel.
For an USB audio device these two fields are of special interest. An USB device can describe the spatial positions via the Audio Channel Cluster Format. This format is described in deeper detail at [4]. The cluster descriptor contains the **bNrChannels** and **wChannelConfig** fields. The standard system USB audio driver (**usbaudio.sys**) will translate the contents of the **bNrChannels** and **wChannelConfig** fields from the USB adapter to the **nChannels** and **dwChannelMask** from the **WAVEFORMATEXTENSIBLE** structure.

As an example, assume a 5.1 device. The driver for that device will return **nChannels** = 6 and **dwChannelMask** = 0x0000003F. For the **dwChannelMask** the driver could also use the predefined macro **KSAUDIO_SPEAKER_5POINT1**. This indicates that the six channels of the audio device are connected to speakers at the Front Left, Front Right, Front Center, Back Left and Back Right and to a Low Frequency Subwoofer.
An audio application would use the **dwChannelMask** field to describe the desired speaker locations in a stream.
It is not required that the **dwChannelMask** of a stream fits the mask of the driver. An intermediate system component, called the **Kernel Mixer (KMixer)**, will map the desired speaker locations of a stream as closely as possible to the real locations as exposed in the **dwChannelMask** of the driver.
As an example, assume a stream with **nChannels** = 4 and **dwChannelMask** = 0x00000033 that should be played via a 5.1 device. The **dwChannelMask** indicates that the audio channels are intended for playback to the Front Left, Front Right, Back Left and Back Right speakers. In this example **KMixer** will open the 5.1 device with all 6 channels, but will mix the four channels of the streams only to the four channels that are also present on the device. Thus playback will only appear on the Front Left, Front Right, Back Left and Back Right speakers.

Additionally an audio driver might implement a feature that allows a user to override the **nChannels** and **dwChannelMask** field. Imagine the case when there are not as many speakers connected to a multi-channel device as there are output channels, e.g. there is only one headphone connected to a 5.1 device. In that case a user might want to change the speaker configuration so he could also hear sound that would normally be directed to speakers that are currently not connected.
This feature is available from the "Sounds and Multimedia Properties" page. A click on the "Advanced" button for the prepared playback device will open the "Advanced Audio Properties" box, which contains the Speaker Setup tab, from which the user could choose his desired speaker configuration.

There is a new AC97 driver example at http://www.microsoft.com/hwdev/audio/DRMsup.htm that shows how to implement this feature in a driver:
A WDM audio driver should create a default speaker configuration when initializing. Using the **KSPROPERTY_AUDIO_CHANNEL_CONFIG** property would notify the driver when the speaker configuration has changed. With this property it is possible for the user to override the default configuration. The **KSPROPERTY_AUDIO_CHANNEL_CONFIG** is also used by **DirectSound** to determine the type of 3d audio algorithm to utilize.
Please note, that this driver example will currently only work under **Windows Millennium Edition**. But with a few changes it will also work under **Windows 2000**.
Another problem is, that currently there is no way to select other speaker configurations as the ones shown in the list box. E.g. there is no way to specify a 7.1 speaker configuration or a configuration without any speaker positions (**dwChannelMask = 0**).



Examination of the nChannels and dwChannelMask fields

The following is an examination of the current implementation of the **nChannels** and **dwChannelMask** fields as found under **Windows 2000**. It has been tested with a multi-channel WDM audio driver and a Win32 application that uses the classic Wave API under **Windows 2000** Build 2195.

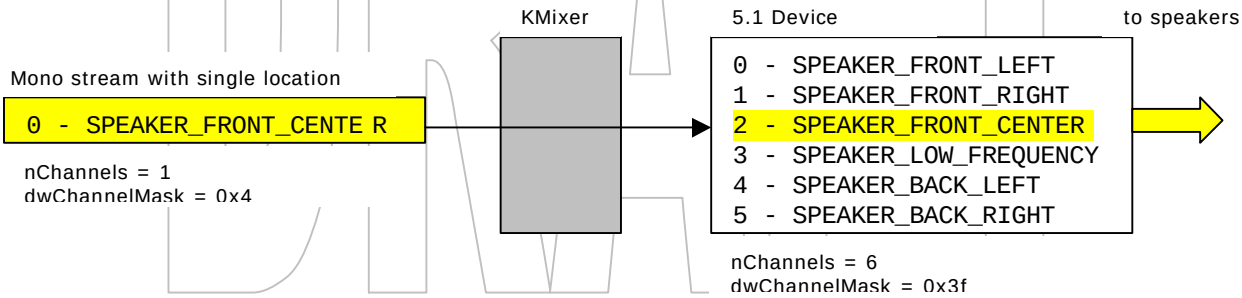
Audio device with specified speaker locations (dwChannelMask != 0)

As described earlier, an audio driver can use the **dwChannelMask** to describe the locations of the speakers connected to the physical channels of the device. An audio application can also describe a playback stream by using a **dwChannelMask** != **0**. **KMixer** is responsible to route the desired speaker locations in the stream to the existing locations as exposed by the driver.

In the following you will find a few examples on how **KMixer** will perform the routing of the various types of streams to a device that exposes channel locations via the **dwChannelMask** field.

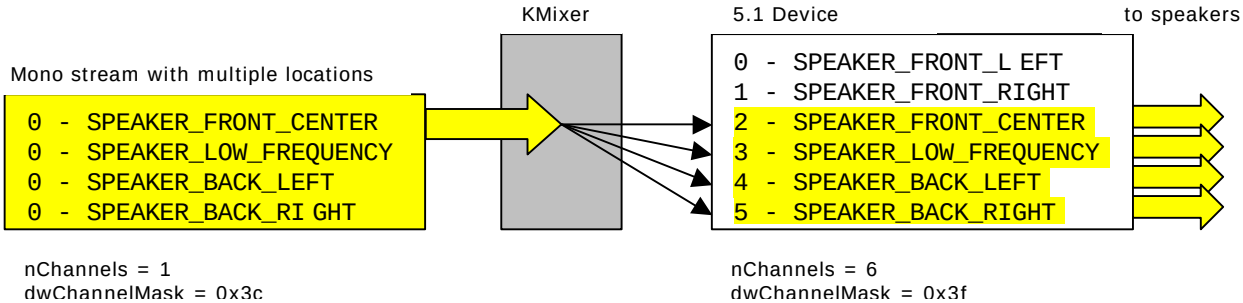
Mono stream with one single speaker location

KMixer will route a mono playback stream to the specified speaker location, if this is also present on the target device.



Mono stream with multiple speaker locations

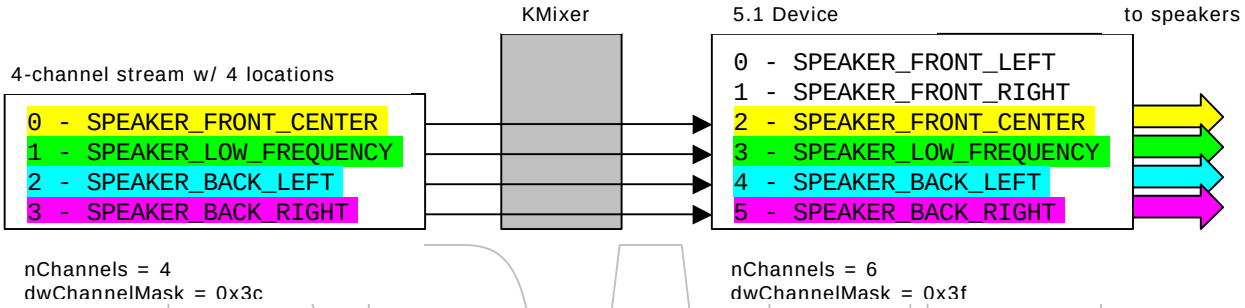
When it is required to route one mono stream to multiple speaker locations equally, it is possible to specify multiple speaker locations in the **dwChannelMask** field for the stream. **KMixer** will then split the mono stream and routes it equally to all specified locations.



NOTE: Microsoft did not document this special case! The documentation [1] says: "Should **nChannels** be less than the number of bits set in **dwChannelMask** then the extra (most significant) bits in **dwChannelMask** are ignored."

Multi-channel stream with multiple speaker locations

KMixer will route a multi-channel stream accordingly to its desired speaker locations as specified in the **dwChannelMask** field.

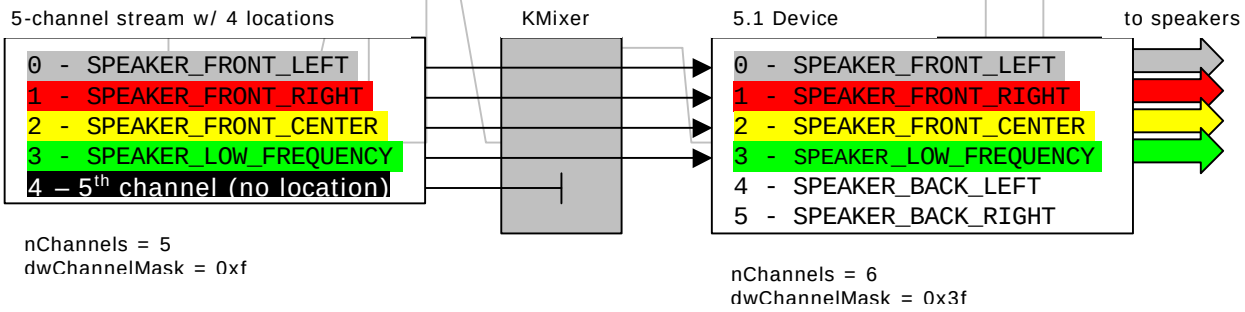


In this case each specified channel data would appear on its appropriate output channel.

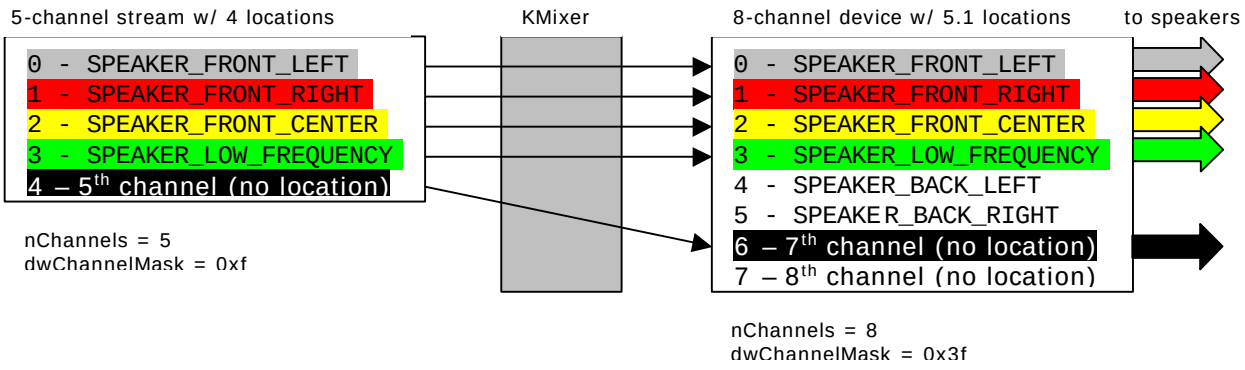
Special cases

*a) **nChannels** exceed the number of bits set in **dwChannelMask***

This case behaves as described in the documentation [1]: "Should **nChannels** exceed the number of bits set in **dwChannelMask** then the remaining channels are not assigned to any particular speaker location. "

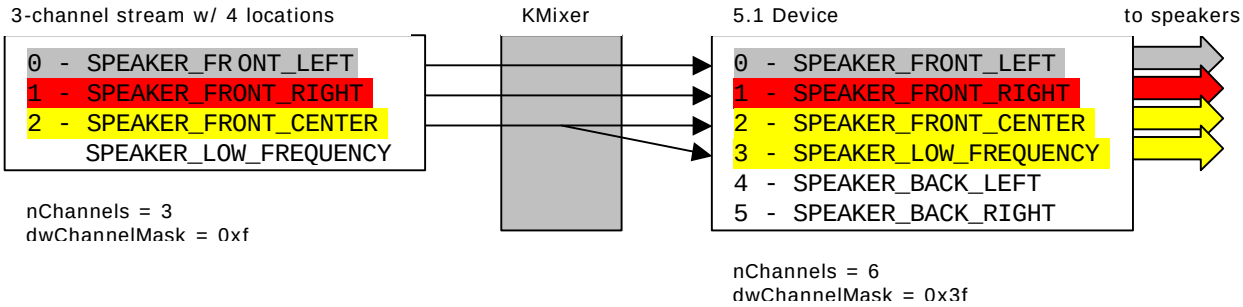


If there are remaining channels on the device that have no speaker locations, then **KMixer** will render the remaining channel data to these outputs.



*b) **nChannels** is less than the number of bits set in **dwChannelMask***

Should **nChannels** be less than the number of bits set in **dwChannelMask** then the destination channels of the extra (most significant) bits in **dwChannelMask** will contain the data of the last channel in the stream.

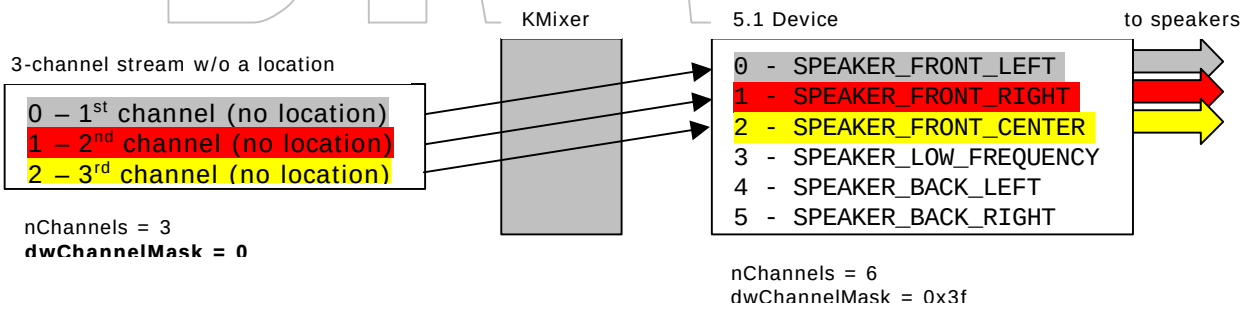


This is nearly the same behaviour as for the mono-stream case that contains multiple channel locations: The last channel in the mono stream (that is the only channel, because it's mono ;-) will be routed equally to first speaker location and additionally to all other locations specified in the extra (most significant) bits in **dwChannelMask**.

NOTE: Microsoft did not document this special case! The documentation [1] says: "Should **nChannels** be less than the number of bits set in **dwChannelMask** then the extra (most significant) bits in **dwChannelMask** are ignored."

*c) **dwChannelMask** of a stream is not used (**dwChannelMask** = **0**)*

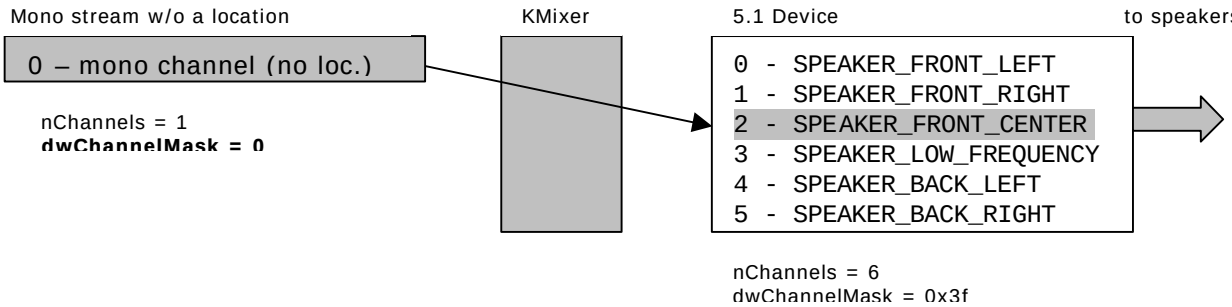
For a **dwChannelMask** of **0** the documentation [1] says: "If, for example in a multi-channel audio authoring application, no speaker location is desired on any of the mono streams, the **dwChannelMask** should explicitly be set to **0**. A **dwChannelMask** of **0** tells the audio device to render the first channel to the first port on the device, the second channel to the second port on the device, and so on. This also means that if the device doesn't know how to process the raw



audio streams, it should not accept the multi-channel stream with a **dwChannelMask** of **0**. A device like a digital mixer or a digital audio storage device (hard disk, ADAT, and so on) might want to accept formats without particular speaker locations."

A special case has been found for a mono stream that doesn't specify a speaker location. In that case **KMixer** will interpret the signal as a "real" mono signal that should only appear on the Front Center speaker (if any).

If there is no **SPEAKER_FRONT_CENTER** location present on the target device then the mono stream will appear on some other channels that represent the Front Center location, e.g. it might appear on the Front Left and Front Right speakers equally.



NOTE: Microsoft did not document this special case! The documentation [1] says: "A **dwChannelMask** of **0** tells the audio device to render the first channel to the first port on the device… "

d) Special cases for 7.1 (or greater) devices

We have found some special behaviour for multi-channel devices that exposes a **dwChannelMask** greater or equal than **SPEAKER_FRONT_LEFT_OF_CENTER**. As shown earlier this mask is e.g. used in the macro **KSAUDIO_SPEAKER_7POINT1**. The following table shows the appearance of a mono stream on the appropriate channels of a 7.1 device when using the various **SPEAKER_xxx** masks for a playback stream:

channels and locations of the 7.1 device **dwChannelMask** describing the mono stream	0 FL	1 FR	2 FC	3 LF	4 BL	5 BR	6 FLC	7 FRC
0	-	-	X	-	-	-	X	X
SPEAKER_FRONT_LEFT	X	-	-	-	-	-	X	-
SPEAKER_FRONT_RIGHT	-	X	-	-	-	-	-	X
SPEAKER_FRONT_CENTER	-	-	X	-	-	-	X	X
SPEAKER_LOW_FREQUENCY	-	-	-	X	-	-	-	-
SPEAKER_BACK_LEFT	-	-	-	-	X	-	-	-
SPEAKER_BACK_RIGHT	-	-	-	-	-	X	-	-
SPEAKER_FRONT_LEFT_OF_CENTER	X	-	X	-	-	-	X	-
SPEAKER_FRONT_RIGHT_OF_CENTER	-	X	X	-	-	-	-	X
SPEAKER_BACK_CENTER					X	X		
SPEAKER_SIDE_LEFT	X	-	-	-	X	-	-	-
SPEAKER_SIDE_RIGHT	-	X	-	-	-	X	-	-
SPEAKER_TOP_CENTER	-	-	X	-	-	-	-	-
SPEAKER_TOP_FRONT_LEFT	X	-	-	-	-	-	-	-
SPEAKER_TOP_FRONT_CENTER	-	-	X	-	-	-	-	-
SPEAKER_TOP_FRONT_RIGHT	-	X	-	-	-	-	-	-
SPEAKER_TOP_BACK_LEFT	-	-	-	-	X	-	-	-
SPEAKER_TOP_BACK_CENTER	-	-	-	-	X	X	-	-
SPEAKER_TOP_BACK_RIGHT	-	-	-	-	-	X	-	-
SPEAKER_RESERVED (0x7FFC0000)	-	-	-	-	-	-	-	-
SPEAKER_ALL (0x80000000)	-	-	-	-	-	-	-	-
0xffffffff	X	X	X	X	X	X	X	X

The table shows that there is obviously an interpretation when using a **dwChannelMask** >= **SPEAKER_FRONT_LEFT_OF_CENTER**. The highlighted fields in the table show where an unexpected behaviour has been found.

E.g. a mono stream with the speaker location **SPEAKER_FRONT_LEFT_OF_CENTER** will obviously not be interpreted by **KMixer** as a separate speaker connected to the audio card. Instead, the channel data is also mixed to the **SPEAKER_FRONT_LEFT** and to the **SPEAKER_FRONT_CENTER** channels of the 7.1 device. Another example is a mono stream that should only appear on the Front Left, the Front Right, or the Front Center speaker. In these cases **KMixer** will also mix the channel data to the "appropriate" Front Left or Right of Center locations (besides, as described earlier, a **dwChannelMask** = **0** would be interpreted as **SPEAKER_FRONT_CENTER**).
Thus, it is not possible to address the channels 6 and 7 separately! It is not clear if this is an issue of the current implementation of **KMixer** or if this behaviour is intended. Unfortunately, Microsoft currently does not document this!
For the second part of the table (**dwChannelMask** >= **SPEAKER_BACK_CENTER**) it looks meaningful that the channel data of the mono stream is mixed to some other locations, since these speaker locations doesn t exist for a 7.1 device.
The last interesting thing is that a mono stream with a **dwChannelMask** = **SPEAKER_ALL** (defined as **0x80000000**) will not be splitted to all available channels, as implied by the macro name. Instead **KMixer** will ignore the channel data. The mono stream will not appear on any channel. A mono stream will only be splitted to all available channels when using a **dwChannelMask** = **0xFFFFFFFF**.

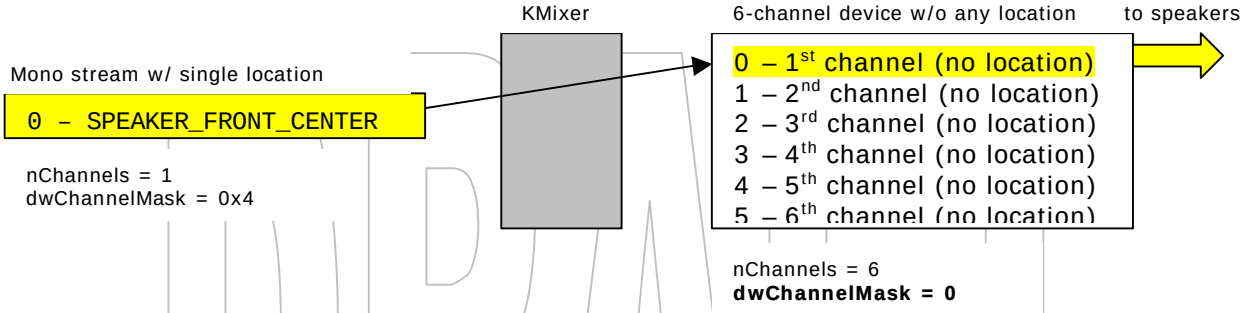
Audio device with no specified speaker locations (dwChannelMask = 0)

If an audio driver doesn t require any speaker locations to describe the physical outputs of the audio adapter then it should return **dwChannelMask** = **0**. This is true e.g. for a device that is intended to connect to a mixing desk or to a digital audio storage device (ADAT, etc…). In this case speaker locations for the output channels wouldn t make any sense.

The following series of experiments will examine how **KMixer** behaves in the case of an multi-channel audio device that exposes no speaker locations.

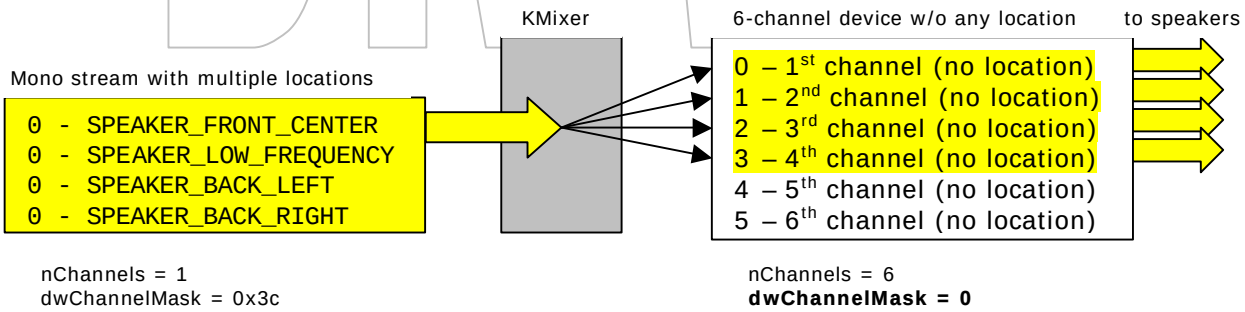
Mono stream with one single speaker location

A mono stream with a single speaker location will always be routed to the first present output channel, regardless of the speaker location that describes the stream.



Mono stream with multiple speaker locations

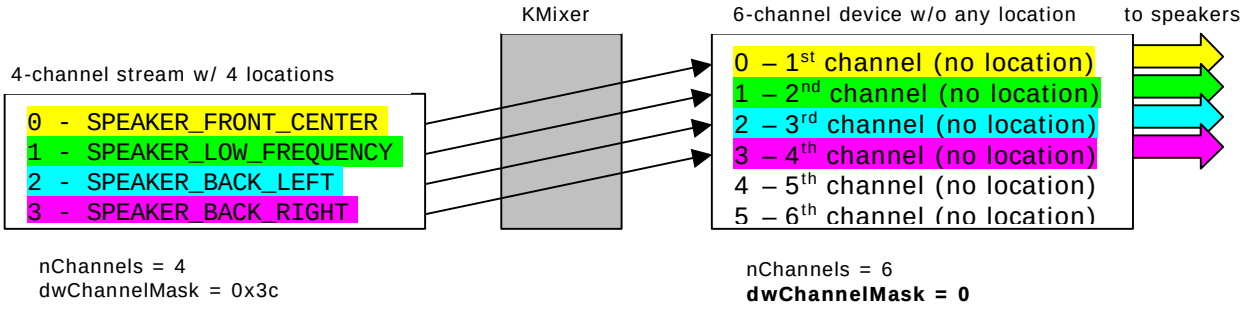
A mono stream that contains multiple speaker locations is splitted by **KMixer** to multiple destination channels starting at the first present output channel.



NOTE: Microsoft did not document this special case! The documentation [1] says: "Should **nChannels** be less than the number of bits set in **dwChannelMask** then the extra (most significant) bits in **dwChannelMask** are ignored."

Multi-channel stream with multiple speaker locations

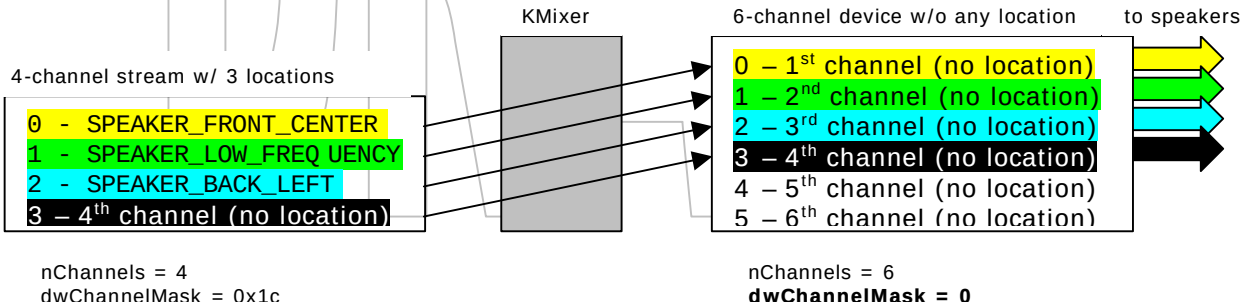
An n-channel stream with n channels in the **dwChannelMask** will be routed to the first n channels of the device. The location information in the **dwChannelMask** will be ignored if the driver doesn t provide any speaker locations.



Special cases

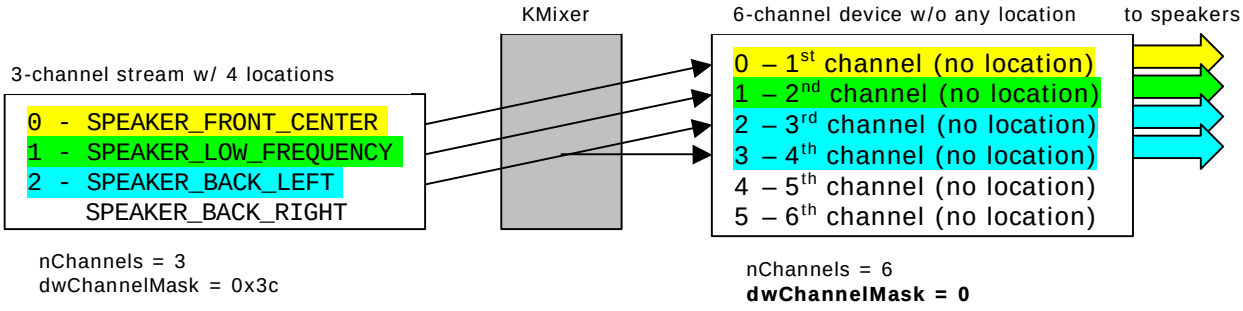
*a) **nChannels** exceed the number of bits set in **dwChannelMask***

In contrast to the situation where the driver exposes speaker locations via the **dwChannelMask** field, it doesn t matter if **nChannels** exceed the number of bits set in **dwChannelMask**. All channels in the stream will be routed to the same number of channels of the device starting at the first output.



*b) **nChannels** is less than the number of bits set in **dwChannelMask***

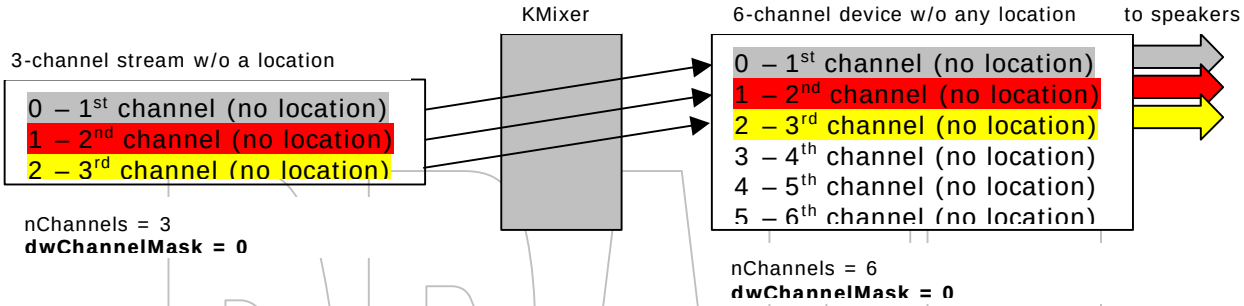
Should **nChannels** be less than the number of bits set in **dwChannelMask** then the destination channels of the extra (most significant) bits in **dwChannelMask** will contain the data of the last channel in the stream. Because in this case the destination location will also be ignored, the single streams will appear starting at the first output.



NOTE: Microsoft did not document this special case! The documentation [1] says: "Should **nChannels** be less than the number of bits set in **dwChannelMask** then the extra (most significant) bits in **dwChannelMask** are ignored."

*c) **dwChannelMask** of a stream is not used (**dwChannelMask** = **0**)*

For a **dwChannelMask** of **0** the documentation [1] says: "If, for example in a multi-channel audio authoring application, no speaker location is desired on any of the mono streams, the **dwChannelMask** should explicitly be set to **0**. A **dwChannelMask** of **0** tells the audio device to render the first channel to the first port on the device, the second channel to the second port on the device, and so on. This also means that if the device doesn't know how to process the raw



audio streams, it should not accept the multi-channel stream with a **dwChannelMask** of **0**. A device like a digital mixer or a digital audio storage device (hard disk, ADAT, and so on) might want to accept formats without particular speaker locations."

In contrast to the situation where the driver exposes speaker locations via the **dwChannelMask** field, a mono stream with **dwChannelMask = 0** will also be routed to the first available output channel.

References

[1] Microsoft Corp., "Enhanced Audio Formats for Multi-Channel Configurations and High-Bit Resolution", 1999, http://www.microsoft.com/hwdev/audio/multichaud.htm
[2] Microsoft Corp., "WDM Audio Drivers for Windows 2000" 1999, http://www.microsoft.com/hwdev/devdes/wdmaudio.htm
[3] Rossum, Dave, Creative Labs "WinHEC 99 White Paper: An Integrated Approach to Multi-Channel Audio" 1999
[4] USB Implementers Forum, "Universal Serial Bus – Device Class Definition for Audio Devices", 1998, http://www.usb.org/developers/data/devclass/audio10.pdf