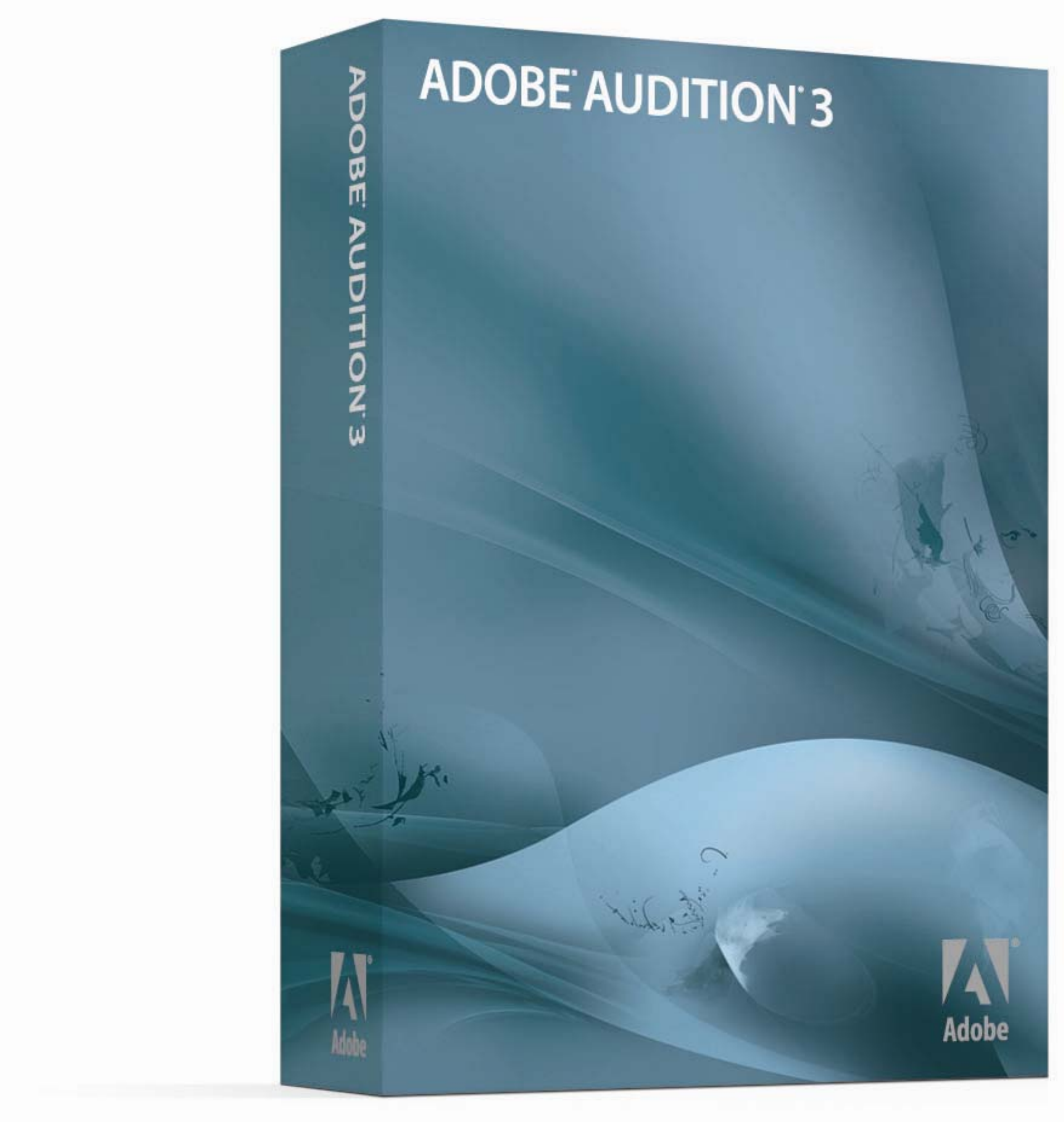

The Session Format (SSF) API

Version 3.0

The Adobe® Audition® Engineering team supplies the material in this document.

Notes on this release of Adobe® Audition® SDK

Please feel free to contact us with any API-related technical questions.

mailto:auditionsdk@adobe.com?subject=Audition_API_Question

Adobe® Audition® can support any number of multitrack session formats. New session formats can be written by following this API. A session format module is a DLL with exported functions Audition calls to read and write audio data.

When Audition starts, it checks the program directory for any files ending in .SSF. If it finds any, it checks to see if the file contains the functions PlugInSessionNew and PlugInSessionGetName, and if so, it calls those functions. If those function calls are successful, then it is assumed to be a valid session format module DLL.

The functions required to access your session format are PlugInSessionNew() and PlugInSessionGetName(). PlugInSessionNew() needs to return a pointer to a newly created CPlugInSession class. This class needs to have the following functions implemented: GetFlags(), Understands(), ReadSession(), CloseSession(), WriteSession. The header file CPlugInSession.h defines the CPlugInSession class and all needed structures, types, and macros for creating your own plugin session format type.

The structures, data types, and information needed to access the multitrack session information are in two different header files. SessionSdkStructs.h defines the session info in primitive C++ types (including raw pointers) with FULL_SDK_SESSION_INFO being the top-level parent struct holding all of the session data. SessionSerializerStructs.h defines the session info in more modern STL types (vectors instead of raw pointer arrays) with COMPLETE_SESSION_INFO_NEW being the top-level parent struct holding all of the session data. Converting between the two are the SessionSdkInterpretter classes defined in SessionSdkInterpretter.h. The CPlugInSession class uses the FULL_SDK_SESSION_INFO, because we don’t want to pass STL types between modules. You can certainly use FULL_SDK_SESSION_INFO directly if you like, but we recommend you convert it to a COMPLETE_SESSION_INFO_NEW, because it the STL types are easier to work with. That is exactly what the SDK sample does... This documentation is written with the assumption that you are working with COMPLETE_SESSION_INFO_NEW.

The SDK consists of a small sample project, several header files and source files, which define all the needed structures and types for you to communicate with Adobe® Audition®. The SDK is intended for use with C++ projects running on any Microsoft Windows™ operating system that is supported by Adobe® Audition®.

It is assumed that the reader understands how to create a DLL and export functions from that DLL. After writing your DLL as described below simply rename it from *.dll to *.ssf and copy that file into your Adobe® Audition® working folder. Adobe® Audition® will automatically detect the presence of *.ssf files on program startup.

Overview of functions called by Adobe® Audition®

An Adobe® Audition® Session Format DLL must implement and expose the following API functions:

PlugInSessionGetName	Gets information about this format's name and what file extensions is supports.
PlugInSessionNew	Returns a CPlugInSession object that is used to read and write session files

HRESULT PlugInSessionGetName(SESSION_NAME_INFO* lpSessionNameInfo)	Required
SESSION_NAME_INFO *lpSessionNameInfo	[OUT] Name and supported file extension information

At startup, Adobe® Audition® will load your DLL and call **PlugInSessionGetName ()** to get name and extension information about your session format. This function may be called at any time. After filling in the **SESSION_NAME_INFO** structure with your information, return S_OK if your device is connected and able to be started or E_FAIL, if not.

CPlugInSession* PlugInSessionNew(void)	Required

At startup, Adobe® Audition® will load your DLL and call **PlugInSessionNew()** to get information about your session format. When this function is called, your DLL should create a new instance of your session format class (derived from CPlugInSession) and return the pointer to Adobe® Audition®. Adobe® Audition® will then control the lifetime management of the session format object and will release it appropriately.

Overview of the CPlugInSession class

```
class CPlugInSession
{
public:
	CPlugInSession() {};
	virtual ~CPlugInSession() {};

	virtual HRESULT GetFlags(DWORD * pdwFlags);

	virtual HRESULT Understands(LPCTSTR szName);

	virtual HRESULT	ReadSession(LPCTSTR szFilename,
FULL_SDK_SESSION_INFO** outSessionInfo);

	virtual HRESULT	WriteSession(LPCTSTR szFilename,
FULL_SDK_SESSION_INFO* inSessionInfo);

	virtual HRESULT	CloseSession(void);

};
```

Every session format must be handled by a class that derives from CPlugInSession. Adobe® Audition® will use your CPlugInSession class to perform the reading and writing of session files.

CPlugInSession class member functions:

HRESULT GetFlags(DWORD* pdwFlags)

DWORD* pdwFlags	[OUT] Flags that indicate what features this session format plugin supports

pdwFlags can contain any of the following flags:

```
#define PLUGINSES_CANOPEN            0x04
#define PLUGINSES_CANSAVE            0x08
#define PLUGINSES_HIDDEN             0x10
```

PLUGINSES_CANOPEN indicates that this plugin format supports the ability to open session format files (and thus, will be displayed as an available format in the ‘Open Session’ dialog.) PLUGINSES_CANSAVE indicates that this plugin format supports the ability to save session format files (and thus, will be displayed as an available format in the ‘Save Session As’ dialog.) PLUGINSES_HIDDEN indicates that, although this plugin format may support opening and saving, it should not be displayed as a supported type in the ‘Open Session’ or ‘Save Session As’ dialogs.

Return: S_OK

HRESULT Understands(LPCTSTR szName)

LPCTSTR szName	[IN] Full path and filename of a file on disk.

This function is a question from Adobe® Audition® to the plugin session format, asking if this plugin format understands the passed in file. (if it can open the file)

Return: S_OK, if this session format understands and can open the specified file. E_FAIL if not.

HRESULT ReadSession(LPCTSTR szFileName, FULL_SDK_SESSION_INFO outSessionInfo)**

LPCTSTR szFileName	[IN] Full path and filename of a file on disk.
LPCTSTR outSessionInfo	[OUT] The session information extracted from szFileName.

This function is called for the session format plugin to read in a session file from disk and to return the information in **outSessionInfo**. It should only be called if the plugin's flags include PLUGINSES_CANOPEN. Typically, this function will be called from Adobe® Audition® after the plugin has already indicated that it can open this file (Adobe® Audition® will first call Understands() to see if the plugin can open this file). The typical behavior is for the plugin to read in **szFileName** and store the session data in a COMPLETE_SESSION_INFO_NEW structure. Then, when finished, the COMPLETE_SESSION_INFO_NEW data is converted to a FULL_SDK_SESSION_INFO structure by using a SessionInterpretter_ToSDK object, and the FULL_SDK_SESSION_INFO pointer is returned via **outSessionInfo**. Then, when Adobe® Audition® calls CloseSession(), the SessionInterpretter_ToSDK object can be deleted, releasing all the allocated memory for the FULL_SDK_SESSION_INFO structure.

Return: S_OK if the file was read in correctly and outSessionInfo has valid data. E_FAIL if problems.

HRESULT WriteSession(LPCTSTR szFileName, FULL_SDK_SESSION_INFO* inSessionInfo)

LPCTSTR szFileName	[IN] Full path and filename of a file on disk.
LPCTSTR inSessionInfo	[IN] The session information that should be written to szFileName.

This function is called for the session format plugin to write out a session file to disk using the information in **inSessionInfo**. It should only be called if the plugin's flags include PLUGINSES_CANSAVE. The typical behavior is for the plugin to create a SessionInterpretter_FromSDK object and use it to convert **inSessionInfo** (FULL_SDK_SESSION_INFO) to a COMPLETE_SESSION_INFO_NEW structure and then interrogate the COMPLETE_SESSION_INFO_NEW for the data to write out to **szFileName**. No extra action is required when Adobe® Audition® calls CloseSession().

Return: S_OK if the file was written without problems. E_FAIL if there were problems.

HRESULT CloseSession()

This function is called from Adobe® Audition® after a call to ReadSession() or WriteSession(). Because ReadSession() returns data that uses memory allocated by the plugin, it is a good idea to release this memory in the CloseSession() function. If a SessionInterpretter_ToSDK object is used to get the FULL_SDK_SESSION_INFO struct (and all contained data), simply destroying the SessionInterpretter_ToSDK object will release all allocated memory in the FULL_SDK_SESSION_INFO struct.

Return: S_OK.

Overview of the SessionInterpretter_FromSDK class

```
class SessionIterpretter_FromSDK
{
public:

	SessionIterpretter_FromSDK();
	~SessionIterpretter_FromSDK();

	COMPLETE_SESSION_INFO_NEW&
GetSessionDataFromSdk(FULL_SDK_SESSION_INFO* inSdkSessionInfo);

	COMPLETE_SESSION_INFO_NEW& GetSessionInfo();

};
```

The SessionInterpretter_FromSDK class has a single purpose: convert a FULL_SDK_SESSION_INFO into a COMPLETE_SESSION_INFO_NEW. Because a COMPLETE_SESSION_INFO_NEW is rather large, this class avoids copying it altogether by returning a reference to a member.

SessionInterpretter_FromSDK class member functions:

COMPLETE_SESSION_INFO_NEW& GetSessionDataFromSdk(FULL_SDK_SESSION_INFO* inSdkSessionInfo)

FULL_SDK_SESSION_INFO* inSdkSessionInfo	[IN] the session info struct that you want to convert.

This function is the real purpose of this class: to convert a FULL_SDK_SESSION_INFO into a COMPLETE_SESSION_INFO_NEW. The return value is a reference to a member variable (to avoid a costly copy), and if the caller stores this return value in a reference, remember that its lifetime is controlled by the SessionInterpretter_FromSDK class.

Return: A reference to the COMPLETE_SESSION_INFO_NEW.

COMPLETE_SESSION_INFO_NEW& GetSessionInfo()

Returns the same converted COMPLETE_SESSION_INFO_NEW as returned by GetSessionDataFromSdk().

Return: A reference to the COMPLETE_SESSION_INFO_NEW.

Overview of the SessionInterpretter_ToSDK class

```
class SessionIterpretter_ToSDK
{
public:

	SessionIterpretter_ToSDK();
	~SessionIterpretter_FromSDK();

	FULL_SDK_SESSION_INFO* FormatSessionDataForSdk(const
COMPLETE_SESSION_INFO_NEW& inSessionInfo);

	FULL_SDK_SESSION_INFO* GetSdkSessionInfo()

	void ClearAllData();

};
```

The SessionInterpretter_ToSDK class has a single purpose: convert a COMPLETE_SESSION_INFO_NEW into a FULL_SDK_SESSION_INFO. The SessionInterpretter_ToSDK allocates and manages the memory needed to create the FULL_SDK_SESSION_INFO. The FULL_SDK_SESSION_INFO gets freed when ClearAllData() is called. If ClearAllData() is not called, then it is freed when the SessionInterpretter_ToSDK dies.

SessionInterpretter_ToSDK class member functions:

FULL_SDK_SESSION_INFO* FormatSessionDataForSdk(const COMPLETE_SESSION_INFO_NEW& inSessionInfo);

COMPLETE_SESSION_INFO_NEW* inSessionInfo	[IN] the session info struct to convert.

This function is the real purpose of this class: to convert a COMPLETE_SESSION_INFO_NEW into a FULL_SDK_SESSION_INFO. The return value is a pointer to a FULL_SDK_SESSION_INFO, whose lifetime is controlled by the SessionInterpretter_ToSDK.

Return: A pointer to a FULL_SDK_SESSION_INFO.

FULL_SDK_SESSION_INFO* GetSdkSessionInfo()

Returns the same converted FULL_SDK_SESSION_INFO as returned by FormatSessionDataForSdk().

Return: A pointer to a FULL_SDK_SESSION_INFO.

---

void ClearAllData()

The lifetime of the converted FULL_SDK_SESSION_INFO is controlled by the SessionInterpretter_ToSDK. When the SessionInterpretter_ToSDK dies, the FULL_SDK_SESSION_INFO dies. But if you want to release the FULL_SDK_SESSION_INFO immediately, then call this function.

Overview of COMPLETE_SESSION_INFO_NEW

```
struct COMPLETE_SESSION_INFO_NEW
{
    std::pair<bool, SESSION_MAIN_INFO_NEW>                                 sessionMainInfo;
    std::pair<bool, VERSION_INFO_NEW>                                      versionInfo;
    std::pair<bool, std::wstring>                                          notesInfo;
    std::pair<bool, SESSION_STATE_INFO_NEW>                                stateInfo;
    std::pair<bool, SESSION_TEMPO_INFO_NEW>                                tempoInfo;
    std::pair<bool, SESSION_ALL_TRACKS_INFO_NEW>                           tracksInfo;
    std::pair<bool, std::vector<SESSION_ENTITY_INFO_NEW> >                 waveEntityInfo;
    std::pair<bool, std::vector<SESSION_ENTITY_INFO_NEW> >                 videoEntityInfo;
    std::pair<bool, std::vector<SESSION_GROUP_INFO_NEW> >                  groupInfo;
    std::pair<bool, SESSION_BLOCK_INFO_NEW>                                blockInfo;
    std::pair<bool, std::vector<SESSION_BLOCK_ENVELOPE_GROUP_INFO_NEW> >   blockEnvelopes;
    std::pair<bool, std::vector<SESSION_LOOP_INFO_NEW> >                   loopInfo;
    std::pair<bool, std::vector<SESSION_CUE_INFO_NEW> >                    cueInfo;
    std::pair<bool, SESSION_METRONOME_INFO_NEW>                            metronomeInfo;
    std::pair<bool, SESSION_MULTICHANNELENCODER_INFO_NEW>           multiChannelEncoderInfo;
};
```

The COMPLETE_SESSION_INFO_NEW struct is the top level container for all the session data. Its members consist of containers of data, each for a specific area of an Adobe® Audition® multitrack session. Some of these are required to be filled with valid data, but many are not. To designate which data members are valid, the std::pair is used, wrapping the data member with a bool, showing if it is valid or not. For example, if notesInfo.first == true, then notesInfo.second has valid data and should be used. If notesInfo.first == false, then notesInfo.second does not have valid data and can be ignored.

COMPLETE_SESSION_INFO_NEW members:

std::pair<bool, SESSION_MAIN_INFO_NEW> sessionMainInfo; Required

This data member contains the main session data. (see the description of SESSION_MAIN_INFO_NEW, below, for more details). It is required.

std::pair<bool, VERSION_INFO_NEW> versionInfo; Optional

This data member contains the Adobe® Audition® version information. (see the description of VERSION_INFO_NEW, below, for more details). It is not required.

std::pair<bool, std::wstring> notesInfo; Optional

This data member contains the session notes information. This is the text that is found in the Adobe® Audition® when you open the Advanced Session Properties dialog and click the 'Notes' tab. It is not required.

std::pair<bool, SESSION_STATE_INFO_NEW> stateInfo; Required

This data member contains the session state information. (see the description of SESSION_STATE_INFO_NEW, below, for more details). It is required.

std::pair<bool, SESSION_TEMPO_INFO_NEW> tempoInfo; Required

This data member contains the session tempo information. (see the description of SESSION_TEMPO_INFO_NEW, below, for more details). It is required.

std::pair<bool, SESSION_ALL_TRACKS_INFO_NEW> tracksInfo; Required

This data member contains the session track information. (see the description of SESSION_ALL_TRACKS_INFO_NEW, below, for more details). It is required.

std::pair<bool, std::vector<SESSION_ENTITY_INFO_NEW> > waveEntityInfo; Optional

This data member contains the information for the audio files used in the session. Each audio file will have a corresponding SESSION_ENTITY_INFO_NEW. (see the description of SESSION_ENTITY_INFO_NEW, below, for more details). It is not required (but it will be needed if your session has any audio files.)

std::pair<bool, std::vector<SESSION_ENTITY_INFO_NEW> > videoEntityInfo; Optional

This data member contains the information for the video file used in the session (if any). If used, it contains one item that represents the video file used in the session. (Adobe® Audition® sessions can have a single video file) (see the description of SESSION_ENTITY_INFO_NEW, below, for more details). It is not required.

std::pair<bool, std::vector<SESSION_GROUP_INFO_NEW> > groupInfo; Optional

This data member contains the information for grouped clips. (see the description of SESSION_GROUP_INFO_NEW, below, for more details). It is not required.

std::pair<bool, SESSION_BLOCK_INFO_NEW> blockInfo; Optional

This data member contains the information for clips (audio and video). (see the description of SESSION_BLOCK_INFO_NEW, below, for more details). It is not required.

std::pair<bool, std::vector<SESSION_BLOCK_ENVELOPE_GROUP_INFO_NEW> > blockEnvelopes; Optional

This data member contains the information for clip envelopes. (see the description of SESSION_BLOCK_ENVELOPE_GROUP_INFO_NEW, below, for more details). It is not required.

std::pair<bool, std::vector<SESSION_LOOP_INFO_NEW> > loopInfo; Optional

This data member contains the information for looped properties of clips. (see the description of SESSION_LOOP_INFO_NEW, below, for more details). It is not required.

std::pair<bool, std::vector<SESSION_CUE_INFO_NEW> > cueInfo; Optional

This data member contains the information for cues (markers). (see the description of SESSION_CUE_INFO_NEW, below, for more details). It is not required.

std::pair<bool, SESSION_METRONOME_INFO_NEW> metronomeInfo; Optional

This data member contains the data for the metronome settings. (see the description of SESSION_METRONOME_INFO_NEW, below, for more details). It is not required.

std::pair<bool, SESSION_MULTICHANNELENCODER_INFO_NEW> multiChannelEncoderInfo; Optional

This data member contains the multichannel encoder (surround encoder) settings. (see the description of SESSION_MULTICHANNELENCODER_INFO_NEW, below, for more details). It is not required.

Overview of SESSION_MAIN_INFO_NEW

```
struct SESSION_MAIN_INFO_NEW
{
   int              iSampleRate;
   __int64          qiNumSamples;
   double           fMasterVolume;
   __int64          qiMainOffset;
   int              iRulerFormat;
   int              bPanningMode;
   double           fVolEnvRange;
   std::wstring     wszMaster;
   std::wstring     wszASIODriver;
   int              iTakeNumber;
   int              iNumberOfBlocks;
   bool             bRecovery;
   std::wstring     wszFilename;
};
```

The SESSION_MAIN_INFO_NEW struct describes the basic information about the session, like the sample rate and the length of the session.

SESSION_MAIN_INFO_NEW members:

---

int iSampleRate;

The sample rate of the session. (e.g. 44100 for a sample rate of 44.1 KHz)

__int64 qiNumSamples;

The session length. The number of samples in the session

---

double fMasterVolume;

This is mostly obsolete - set to 1.0

__int64 qiMainOffset;

The SMPTE Start Time Offset. (see the "SMPTE Start Time Offset" option in the General tab of the Advanced Session Properties dialog.)

---

int iRulerFormat;

The ruler format of the session. (use the kRulerFormat enum)

```
enum kRulerFormat
{
      kRulerFormat_HoursMinutesSeconds = 0,
      kRulerFormat_Samples = 1,
      kRulerFormat_Frames = 2,
      kRulerFormat_SMPTE2997Drop = 3,
      kRulerFormat_BarsAndBeats = 4,
      kRulerFormat_SMPTE2997 = 5
};
```

int bPanningMode;

The panning mode of the session. (set to 0 for "L/R Cut Logarithmic", 1 for "-3dB Center, Constant Power")

double fVolEnvRange;

The volume envelope range multiplier. For volume envelopes (always in 0 to 1.0 range), multiply by this factor for true volume

std::wstring wszMaster;

obsolete - leave set as "Master"

std::wstring wszASIODriver;

The name of the audio driver selected in the Multitrack View tab of the Audio Hardware Setup dialog.

int iTakeNumber;

Ignore. Set to 0

int iNumberOfBlocks;

The total number of audio and video blocks (doesn't count rewire or midi tracks)

bool bRecovery;

This is mainly used for internal purposes only. For standard use, bRecovery should be set to false and wszFilename can be empty. If bRecovery is true, that indicates that this session was saved via "auto-save for recovery", and then wszFilename will be used as the sessions filename, rather than the actual name of the file.

std::wstring wszFilename;

See notes for bRecovery, above.

Overview of VERSION_INFO_NEW

```
struct VERSION_INFO_NEW
{
   double version;
   double buildNumber;
   std::wstring szVersionStringInfo;
};
```

The VERSION_INFO_NEW struct describes the version of Adobe® Audition®.

VERSION_INFO_NEW members:

---

double version;

Adobe® Audition® version (e.g. 3.0)

double buildNumber;

Adobe® Audition® build number (use 0.0, as this is currently not used)

std::wstring szVersionStringInfo;

Adobe® Audition® version string (e.g. "Audition 3.0 [0.0]")

Overview of SESSION_STATE_INFO_NEW

```
struct SESSION_STATE_INFO_NEW
{
   // Horizontal view
   double fLeftSample;
   double fRightSample;

   // Vertical view
   double fLoTrack;
   double fHiTrack;
   double fNewLoTrack;

   // Selection
   __int64 qiLoSample;
   __int64 qiHiSample;
};
```

The SESSION_STATE_INFO_NEW struct describes the selection and view state of the multitrack session.

SESSION_STATE_INFO_NEW members:

double fLeftSample;

The sample at the left edge of the view.

double fRightSample;

The sample at the right edge of the view.

double fLoTrack;

Indicates the first pixel of track viewing area. If your session has 10 tracks and each track has a track height of 100 pixels, then the total track viewing area is 1000 pixles. If fLoTrack is 300 and fHiTrack is 700, then that indicates that the tracks in view will be tracks 4 thru 7

double fHiTrack;

Indicates the first pixel of track viewing area. (see additional notes for fLoTrack)

double fNewLoTrack;

This should be set to the same value as fLoTrack.

__int64 qiLoSample;

The first sample in the selection.

__int64 qiHiSample;

The last sample in the selection. (for no selection, set qiHiSample equal to qiLoSample – 1)

Overview of SESSION_TEMPO_INFO_NEW

```
struct SESSION_TEMPO_INFO_NEW
{
   double fBeatsPerMinute;
   int iBeatsPerBar;
   int iTicksPerQuarterNote;
   double fBeatOffsetMs;
   int iSessionBaseNote;
   int iNoteLen;
};
```

The SESSION_TEMPO_INFO_NEW struct describes the tempo information for the multitrack session. Most of the members have direct correlations to options found in the Tempo tab of the Advanced Session Properties dialog, or the Session Properties dockable tab.

SESSION_TEMPO_INFO_NEW members:

---

double fBeatsPerMinute;

Beats per Minute.

int iBeatsPerBar;

Top number of the time signature.

int iTicksPerQuarterNote;

Ticks per beat.

double fBeatOffsetMs;

See the "Song Start" option in the Tempo tab of the Advanced Session Properties dialog.

int iSessionBaseNote;

The base note or key for the session. Use the `kSessionBaseNote enum for this.`

```
enum kSessionBaseNote
{
      kSessionBaseNote_None          = 0,
      kSessionBaseNote_C             = 60,
      kSessionBaseNote_CSharp        = 61,
      kSessionBaseNote_D             = 62,
      kSessionBaseNote_DSharp        = 63,
      kSessionBaseNote_E             = 64,
      kSessionBaseNote_F             = 65,
      kSessionBaseNote_FSharp        = 66,
      kSessionBaseNote_G             = 67,
      kSessionBaseNote_GSharp        = 68,
      kSessionBaseNote_A             = 69,
      kSessionBaseNote_ASharp        = 70,
      kSessionBaseNote_B             = 71,
};
```

int iNoteLen;

Bottom number of the time signature.

Overview of SESSION_ALL_TRACKS_INFO_NEW

```
struct SESSION_ALL_TRACKS_INFO_NEW
{
   unsigned int   nextTrackID;

   SESSION_AUDIO_TRACK_FULL_INFO_NEW masterTrack;
   std::vector<SESSION_AUDIO_TRACK_FULL_INFO_NEW> audioTracks;
   std::vector<SESSION_COMMON_TRACK_FULL_INFO_NEW> nonAudioTracks;
};
```

The SESSION_ALL_TRACKS_INFO_NEW struct contains information about the tracks. There must be a master track. There can be either 0 or 1 video track. And there can be 0 or more audio tracks (including bus tracks, rewire tracks, and midi tracks).

SESSION_ALL_TRACKS_INFO_NEW members:

unsigned int nextTrackID;

Although the name of this is nextTrackID, it doesn't match up to a track's id. Instead, it matches up to it's order index (SESSION_AUDIO_TRACK_FULL_INFO_NEW::trackOrderIndex), which is an alternative form of track id. If a session has 5 tracks, the indexes might be 0, 1, 2, 3, and 4, and nextTrackID would be 5. In this case, nextTrackID is equal to the number of tracks. However, if a user deletes tracks 2 and 3, then the track indexes would be 0, 1, and 4, and nextTrackID would still be 5. To keep things simple, track indexes are not reused. So nextTrackID should be equal to the largest track index, plus 1.

SESSION_AUDIO_TRACK_FULL_INFO_NEW masterTrack;

master track. For more information on this data member, see the description for SESSION_AUDIO_TRACK_FULL_INFO_NEW, below.

std::vector<SESSION_AUDIO_TRACK_FULL_INFO_NEW> audioTracks;

audio tracks. For more information on this data member, see the description for SESSION_AUDIO_TRACK_FULL_INFO_NEW, below.

std::vector<SESSION_COMMON_TRACK_FULL_INFO_NEW> nonAudioTracks;

video tracks (there should be either 0 or 1). For more information on this data member, see the description for SESSION_COMMON_TRACK_FULL_INFO_NEW, below.

Overview of SESSION_AUDIO_TRACK_FULL_INFO_NEW

```
struct SESSION_AUDIO_TRACK_FULL_INFO_NEW
{
   unsigned int                     trackType;
   unsigned int                     trackOrderIndex;
   SESSION_AUDIO_TRACK_INFO_NEW     track;
};
```

The SESSION_AUDIO_TRACK_FULL_INFO_NEW struct contains information about an audio track. In has a SESSION_AUDIO_TRACK_INFO_NEW data item, which contains most of the track information, but then has some extra data as well.

SESSION_AUDIO_TRACK_FULL_INFO_NEW members:

---

unsigned int trackType;

Indicates what kind of track this is, using the following values:

Track_Audio = 0	// audio track
Track_Bus = 1	// audio track
Track_Master = 2	// audio track
Track_OBSOLETE = 3	// obsolete - do not use
Track_Video = 4	// non-audio track (only has common track info)
Track_MIDIHost = 5	// audio track

---

unsigned int trackOrderIndex;

determines how the tracks are ordered in the UI. The master track is always 0. Other tracks start their ordering at 1. (so the top track would have an order index of 1, the next track 2, etc…)

SESSION_AUDIO_TRACK_INFO_NEW track;

the rest of the track information. For more information, see SESSION_AUDIO_TRACK_INFO_NEW, below.

Overview of SESSION_COMMON_TRACK_FULL_INFO_NEW

```
struct SESSION_COMMON_TRACK_FULL_INFO_NEW
{
   unsigned int                     trackType;
   unsigned int                     trackOrderIndex;
   SESSION_COMMON_TRACK_INFO_NEW    track;
};
```

The SESSION_COMMON_TRACK_FULL_INFO_NEW struct contains information about a video track. In has a SESSION_COMMON_TRACK_INFO_NEW data item, which is also used by audio tracks, but then has some extra data as well.

SESSION_COMMON_TRACK_FULL_INFO_NEW members:

---

unsigned int trackType;

---

same as SESSION_AUDIO_TRACK_FULL_INFO_NEW, above (should always be Track_Video, since this should always represent a video track)

---

unsigned int trackOrderIndex;

---

same as SESSION_AUDIO_TRACK_FULL_INFO_NEW, above (should always be 1, since this should always represent a video track)

SESSION_COMMON_TRACK_INFO_NEW track;

---

the rest of the track information. For more information, see SESSION_COMMON_TRACK_INFO_NEW, below.

Overview of SESSION_COMMON_TRACK_INFO_NEW

```
struct SESSION_COMMON_TRACK_INFO_NEW
{
   unsigned long  id;
   unsigned long  dwFlags;
   std::wstring   szTitle;
   unsigned long  dwTrackHeight;
};
```

The SESSION_COMMON_TRACK_INFO_NEW struct contains information about a track. This struct is used by both video and audio tracks.

SESSION_COMMON_TRACK_INFO_NEW members:

---

unsigned long id;

Track id. Valid ids for audio tracks should start from 10000 (up to 239999). The valid id for the video track (there can be only one) is 490000

---

unsigned long dwFlags;

See track flags below

```
#define F_TRACK_MUTE           0x01         // Mute
#define F_TRACK_SOLO           0x02         // Solo
#define F_TRACK_RECORD         0x04         // Armed for record
#define F_TRACK_FROZEN         0x08         // Frozen
#define F_PRE_METER            0x10         // ignore (unused)
#define F_PRE_EFFECT           0x20         // ignore (unused)
#define F_TRACK_EQ_UI          0x40         // ignore (unused)
#define F_EFFECTS_RACK_UI      0x80         // ignore (unused)
```

---

std::wstring szTitle;

Track title

---

unsigned long dwTrackHeight;

Track height (in pixels)

Overview of SESSION_AUDIO_TRACK_INFO_NEW

```
struct SESSION_AUDIO_TRACK_INFO_NEW
{
   SESSION_COMMON_TRACK_INFO_NEW baseTrackInfo;

   unsigned int   trackType;
   unsigned int   busID;
   unsigned int   automationMode;
   std::wstring   inputDriver;
   unsigned int   inputPort;
   std::wstring   inputPortName;
   std::wstring   outputDriver;
   unsigned int   outputPort;
   std::wstring   outputPortName;
   bool           isPreMeter;
   bool           isPreEffect;
   bool           isFreeze;
   bool           isTrackEQUI;
   bool           isEffectsRackUI;

   SESSION_AUDIO_COMPONENT_NEW                        inputComponent;
   SESSION_AUDIO_COMPONENT_NEW                        eqComponent;
   SESSION_AUDIO_COMPONENT_NEW                        faderComponent;
   SESSION_AUDIO_COMPONENT_NEW                        muteComponent;
   SESSION_AUDIO_OUTPUT_COMPONENT_NEW                 outputComponent;
   SESSION_AUDIO_EFFECT_COMPONENT_GROUP_NEW           effectComponentGroup;
   std::vector<SESSION_AUDIO_SEND_COMPONENT_NEW>      sendComponents;
   std::vector<SESSION_AUDIO_EFFECT_COMPONENT_NEW>    effectComponents;

   std::vector<bool>    hasEffectInSlot;
   bool                 isShowingAutomationLanes;
   unsigned int         trackHeight;

   std::vector<SESSION_AUDIO_TRACK_AUTOMATION_LANE_INFO_NEW>      automationLanes;
   SESSION_AUDIO_EFFECT_COMPONENT_NEW                             midiHostEffect;
};
```

The SESSION_AUDIO_TRACK_INFO_NEW struct contains information about an audio track.

SESSION_AUDIO_TRACK_INFO_NEW members:

SESSION_COMMON_TRACK_INFO_NEW baseTrackInfo;

Basic track info. For more information, see SESSION_COMMON_TRACK_INFO_NEW, above.

unsigned int trackType;

Defines the track type. Use the following values:

TRACK_CONTENT = 0	// standard audio track with clips and such
TRACK_BUS = 1	// bus track
TRACK_MASTER = 2	// master track
TRACK_MIDIHOST = 3	// midi host track

unsigned int busID;

identifies the bus ID, if the track is a bus. If the track is not a bus, set this to 0. If the track is the master track (technically a bus), set this to 20000. If the track is a bus track, set this to a bus ID, starting at 20001.

unsigned int automationMode;

Automation mode. Use the following values:

AUTOMATION_OFF = 0
AUTOMATION_READ = 1
AUTOMATION_WRITE = 2
AUTOMATION_LATCH = 3
AUTOMATION_TOUCH = 4

std::wstring inputDriver;

The selected sound card. (should match up with SESSION_MAIN_INFO_NEW::wszASIODriver)

unsigned int inputPort;

The input port number used

std::wstring inputPortName;

The name of the input port

std::wstring outputDriver;

The selected sound card. (should match up with SESSION_MAIN_INFO_NEW::wszASIODriver)

unsigned int outputPort;

The busID of the target bus if outputting to a bus. The output port number if outputting to an audio device

std::wstring outputPortName;

The name of the output port

bool isPreMeter;

set to true if the track is armed for record (arming a track for record is done by setting a flag on SESSION_COMMON_TRACK_INFO_NEW::dwFlags)

bool isPreEffect;

determines whether the effects are pre-fader or post-fader

bool isFreeze;

is the track frozen? True, yes. False, no. (And if so, make sure the appropriate flag is set on SESSION_COMMON_TRACK_INFO_NEW::dwFlags)

bool isTrackEQUI;

not used - set to false

bool isEffectsRackUI;

not used - set to false

SESSION_AUDIO_COMPONENT_NEW inputComponent;

Input component. For more information, see SESSION_AUDIO_COMPONENT_NEW, below.
(componentType should be set to kAudioComponentType_Input)

SESSION_AUDIO_COMPONENT_NEW eqComponent;

Eq component. For more information, see SESSION_AUDIO_COMPONENT_NEW, below. componentType should be set to kAudioComponentType_EQ

SESSION_AUDIO_COMPONENT_NEW faderComponent;

Fader component. For more information, see SESSION_AUDIO_COMPONENT_NEW, below.
componentType should be set to kAudioComponentType_Fader

SESSION_AUDIO_COMPONENT_NEW muteComponent;

Mute component. For more information, see SESSION_AUDIO_COMPONENT_NEW, below.
componentType should be set to kAudioComponentType_Mute

SESSION_AUDIO_OUTPUT_COMPONENT_NEW outputComponent;

Output component. For more information, see SESSION_AUDIO_OUTPUT_COMPONENT_NEW, below

SESSION_AUDIO_EFFECT_COMPONENT_GROUP_NEW effectComponentGroup;

Effect group component. For more information, see
SESSION_AUDIO_EFFECT_COMPONENT_GROUP_NEW, below

std::vector<SESSION_AUDIO_SEND_COMPONENT_NEW> sendComponents;

Send components. For more information, see SESSION_AUDIO_SEND_COMPONENT_NEW, below

std::vector<SESSION_AUDIO_EFFECT_COMPONENT_NEW> effectComponents;

Effect components. For more information, see SESSION_AUDIO_EFFECT_COMPONENT_NEW, below

std::vector<bool> hasEffectInSlot;

This determines what slots the effects (in effectComponents) use. This is expected to be a vector of 16 elements, all should be false except for where an effect exists. So if only the 3rd item in this vector is true, then this implies a single effect and it's placement is in the 3rd slot.

bool isShowingAutomationLanes;

not used - set to false

unsigned int trackHeight;

track height (in pixels)

std::vector<SESSION_AUDIO_TRACK_AUTOMATION_LANE_INFO_NEW> automationLanes;

see SESSION_AUDIO_TRACK_AUTOMATION_LANE_INFO_NEW, below

SESSION_AUDIO_EFFECT_COMPONENT_NEW midiHostEffect;

The midi host component (if trackType == TRACK_MIDIHOST. Otherwise, this is ignored). For more information, see SESSION_AUDIO_EFFECT_COMPONENT_NEW, below

Overview of SESSION_AUDIO_COMPONENT_NEW

```
struct SESSION_AUDIO_COMPONENT_NEW
{
   unsigned int componentType;
   __int64 initialDelay;
   bool isBypass;
   bool isSerial;
   std::vector<SESSION_AUDIO_PARAM_NEW> audioParams;
};
```

The SESSION_AUDIO_COMPONENT_NEW struct contains information about an audio component.

SESSION_AUDIO_COMPONENT_NEW members:

---

unsigned int componentType;

---

use the kComponentType enum, below

```
enum kComponentType
{
      kAudioComponentType_Input          = 0,    // input gain
      kAudioComponentType_Fader          = 1,    // volume
      kAudioComponentType_Meter          = 2,    // meter
      kAudioComponentType_Send           = 3,    // send
      kAudioComponentType_Effect         = 4,    // insert effects (requires a guidString to identify the effect)
      kAudioComponentType_EQ             = 5,    // track eq
      kAudioComponentType_Mute           = 6,    // mute
      kAudioComponentType_Output         = 7,    // pan
      kAudioComponentType_EffectGroup    = 8     // insert effects group
};
```

__int64 initialDelay;

---

set to 0

---

bool isBypass;

---

true if turned off, bypassed, disabled. false if this component is on / enabled.

---

bool isSerial;

---

set to true

---

std::vector<SESSION_AUDIO_PARAM_NEW> audioParams;

---

automation data for this component's parameters. For more information, see the SESSION_AUDIO_PARAM_NEW description, below.

Overview of SESSION_AUDIO_OUTPUT_COMPONENT_NEW

```
struct SESSION_AUDIO_OUTPUT_COMPONENT_NEW
{
   unsigned int componentType;
   __int64 initialDelay;
   bool isBypass;
   std::vector<SESSION_AUDIO_PARAM_NEW> audioParams;
   unsigned int pannerMode;
   bool isSumDown;
   unsigned int outputChannels;
   int busID;
};
```

The SESSION_AUDIO_OUTPUT_COMPONENT_NEW struct contains information about an audio output component.

SESSION_AUDIO_OUTPUT_COMPONENT_NEW members:

---

unsigned int componentType;

---

same as SESSION_AUDIO_COMPONENT_NEW::componentType. Set to kAudioComponentType_Output.

__int64 initialDelay;

---

set to 0

---

bool isBypass;

---

set to false

---

std::vector<SESSION_AUDIO_PARAM_NEW> audioParams;

---

automation data for this component's parameters. For more information, see the SESSION_AUDIO_PARAM_NEW description, below.

---

unsigned int pannerMode;

---

should always be set to 1 if the track output device is stereo, 0 if the track output device is mono

---

bool isSumDown;

---

true if the track output is summed to mono. false if not summed to mono.

---

unsigned int outputChannels;

---

should either be set to 2 (if outputting to a stereo device) or 1 (if outputting to a mono device)

---

int busID;

---

set to 0 (if this track outputs to a bus, then the busID is set in the SESSION_AUDIO_TRACK_INFO_NEW::outputPort field)

Overview of SESSION_AUDIO_EFFECT_COMPONENT_GROUP_NEW

```
struct SESSION_AUDIO_EFFECT_COMPONENT_GROUP_NEW
{
   unsigned int componentType;
   __int64 initialDelay;
   bool isBypass;
   bool isSerial;
   std::vector<SESSION_AUDIO_PARAM_NEW> audioParams;
   SESSION_AUDIO_COMPONENT_NEW inputGain;
   SESSION_AUDIO_COMPONENT_NEW outputGain;
};
```

The SESSION_AUDIO_EFFECT_COMPONENT_GROUP_NEW struct contains information about an audio output component.

SESSION_AUDIO_EFFECT_COMPONENT_GROUP_NEW members:

---

unsigned int componentType;

---

same as SESSION_AUDIO_COMPONENT_NEW::componentType. Set to kAudioComponentType_EffectGroup

---

__int64 initialDelay;

---

set to 0

---

bool isBypass;

---

true if turned off, bypassed, disabled. false if this component is on / enabled.

---

bool isSerial;

---

set to true

---

std::vector<SESSION_AUDIO_PARAM_NEW> audioParams;

---

automation data for this component's parameters. For more information, see the SESSION_AUDIO_PARAM_NEW description, below.

SESSION_AUDIO_COMPONENT_NEW inputGain;

Input gain audio component. For more information, see the SESSION_AUDIO_COMPONENT_NEW description, above. (this uses a component type of 0)

SESSION_AUDIO_COMPONENT_NEW outputGain;

Output gain audio component. For more information, see the SESSION_AUDIO_COMPONENT_NEW description, above. (this uses a component type of 0)

Overview of SESSION_AUDIO_SEND_COMPONENT_NEW

```
struct SESSION_AUDIO_SEND_COMPONENT_NEW
{
   unsigned int componentType;
   __int64 initialDelay;
   bool isBypass;
   std::vector<SESSION_AUDIO_PARAM_NEW> audioParams;
   unsigned int pannerMode;
   bool isSumDown;
   unsigned int numOutputChannels;
   int busID;
   bool isPre;
};
```

The SESSION_AUDIO_SEND_COMPONENT_NEW struct contains information about an audio send component.

SESSION_AUDIO_SEND_COMPONENT_NEW members:

unsigned int componentType;

same as SESSION_AUDIO_COMPONENT_NEW::componentType. Set to kAudioComponentType_Send

__int64 initialDelay;

set to 0

bool isBypass;

true if turned off, bypassed, disabled. false if this component is on / enabled.

std::vector<SESSION_AUDIO_PARAM_NEW> audioParams;

automation data for this component's parameters. For more information, see the SESSION_AUDIO_PARAM_NEW description, below.

unsigned int pannerMode;

set to 1

bool isSumDown;

set to false

unsigned int numOutputChannels;

set to 2

int busID;

set to 0 if this send is unused. If this send is used, this should match up with SESSION_AUDIO_TRACK_INFO_NEW::busID for the buss that is used

bool isPre;

true if the send is pre-fader, false if the send is post-fader

Overview of SESSION_AUDIO_EFFECT_COMPONENT_NEW

```
struct SESSION_AUDIO_EFFECT_COMPONENT_NEW
{
   unsigned int componentType;
   __int64 initialDelay;
   bool isBypass;
   std::vector<SESSION_AUDIO_PARAM_NEW> audioParams;
   GUID guid;
   std::wstring name;
   std::wstring groupName;
   std::wstring effectParamsString;
};
```

The SESSION_AUDIO_EFFECT_COMPONENT_NEW struct contains information about an audio effect component (used for VST effects applied to a track)

SESSION_AUDIO_EFFECT_COMPONENT_NEW members:

unsigned int componentType;

same as SESSION_AUDIO_COMPONENT_NEW::componentType. Set to kAudioComponentType_Effect

__int64 initialDelay;

set to 0

bool isBypass;

true if turned off, bypassed, disabled. false if this component is on / enabled.

std::vector<SESSION_AUDIO_PARAM_NEW> audioParams;

automation data for this component's parameters. For more information, see the SESSION_AUDIO_PARAM_NEW description, below.

GUID guid;

The GUID of the vst effect.

std::wstring name;

used for reading in older .ses sessions (also used as a backup if no match is found using the GUID)

std::wstring groupName;

used for reading in older .ses sessions (also used as a backup if no match is found using the GUID)

std::wstring effectParamsString;

A string representation of the binary data chunk for the effect parameters

Overview of SESSION_AUDIO_TRACK_AUTOMATION_LANE_INFO_NEW

```
struct SESSION_AUDIO_TRACK_AUTOMATION_LANE_INFO_NEW
{
   unsigned int   audioComponentType;
   std::wstring   guidString;
   std::wstring   componentName;
   unsigned int   parameterIndex;
   unsigned int   laneUIHeight;
};
```

The SESSION_AUDIO_TRACK_AUTOMATION_LANE_INFO_NEW struct contains information about an automation lane.

SESSION_AUDIO_TRACK_AUTOMATION_LANE_INFO_NEW members:

---

unsigned int audioComponentType;

---

same as SESSION_AUDIO_COMPONENT_NEW::componentType.

---

std::wstring guidString;

---

if audioComponentType is 4, then this should be the effect's guid. If the audioComponentType is not 4, then this should be set to "no guid". (Note: VST effects don’t really have a GUID, instead they have a 4 byte identifier. Adobe® Audition® uses a GUID to reference the VST – it creates a GUID using the 4 byte identifier. Simply take the following GUID{EA93BBBE-0B8F-47D6-AC6D-B67B45687047}, and replace the last 4 bytes (8 digits) with the bytes from the VST’s unique identifier.)

---

std::wstring componentName;

---

only really needed with kComponentType_Send. ("Send 1 Output:", "Send 2 Output:", etc...)

---

unsigned int parameterIndex;

---

the parameter of the component

---

unsigned int laneUIHeight;

---

laneUIHeight (in pixels)

Overview of SESSION_AUDIO_PARAM_NEW

```
struct SESSION_AUDIO_PARAM_NEW
{
   bool                 bIsAutomatable;
   double               parameterValueForRendering;
   bool                 bSafeDuringWrite;
   SESSION_ENV_INFO_NEW envelope;
};
```

The SESSION_AUDIO_PARAM_NEW struct contains information about an automatable audio parameter.

SESSION_AUDIO_PARAM_NEW members:

bool bIsAutomatable;

true if the parameter is automatable, false if not

double parameterValueForRendering;

the current value (should match up to the value in the envelope at sample position SESSION_STATE_INFO_NEW::qiLowSample)

bool bSafeDuringWrite;

true if the option for "safe during write" is set for this parameter. otherwise, false.

SESSION_ENV_INFO_NEW envelope;

the parameter automation envelope. For more information, see the description of SESSION_ENV_INFO_NEW, below.

Overview of SESSION_ENV_INFO_NEW

```
struct SESSION_ENV_INFO_NEW
{
   unsigned int uiType;
   bool bSplined;
   std::wstring   szName;
   unsigned long  dwFlags;
   unsigned long  LineColor;
   __int64  dwSize;
   bool     bReadOnly;
   std::vector<SESSION_ENVELOPE_POINT_INFO_NEW> points;
};
```

The SESSION_ENV_INFO_NEW struct contains information about an automation envelope.

SESSION_ENV_INFO_NEW members:

---

unsigned int uiType;

typically set to 0. Set to 1 for volume faders, and 2 for panning.

bool bSplined;

unused. set to false

std::wstring szName;

name of the parameter that owns this envelope

unsigned long dwFlags;

unused. set to 0

unsigned long LineColor;

an RGB value in the COLORREF format

__int64 dwSize;

How many samples long is the envelope

bool bReadOnly;

This should match the SESSION_AUDIO_PARAM_NEW::bSafeDuringWrite value. (i.e. if safe during write, then also read only)

std::vector<SESSION_ENVELOPE_POINT_INFO_NEW> points;

This contains all the envelope points in the envelope. For more information, see the description of SESSION_ENVELOPE_POINT_INFO_NEW, below.

Overview of SESSION_ENVELOPE_POINT_INFO_NEW

```
struct SESSION_ENVELOPE_POINT_INFO_NEW
{
   __int64 dwAt;
   double value;
};
```

The SESSION_ENVELOPE_POINT_INFO_NEW struct contains information about an envelope.

SESSION_ENVELOPE_POINT_INFO_NEW members:

__int64 dwAt;

---

sample position of this point

---

double value;

---

normalized value (from 0.0 to 1.0 - from the UI perspective, 0.0 would show the point at the bottom of the block/track, 1.0 would show the point at the top of the block/track)

Overview of SESSION_ENTITY_INFO_NEW

```
struct SESSION_ENTITY_INFO_NEW
{
   unsigned long dwWaveformID;
   unsigned long dwFileFormat;
   int iCodecDelay;
   __int64 dwTrueLength;
   unsigned int iEntityType;
   std::wstring wszFilename;
};
```

The SESSION_ENTITY_INFO_NEW struct describes an entity, either an audio file or video file.

SESSION_ENTITY_INFO_NEW members:

---

unsigned long dwWaveformID;

unique identifier for this entity. It’s compared against SESSION_COMMON_BLOCK_INFO_NEW::dwEntityId to determine which entities are used for each block.

unsigned long dwFileFormat;

specifies a specific file format, but you should simply set this to -1

int iCodecDelay;

obsolete - set to -1

__int64 dwTrueLength;

if an audio entity, set this to the length of the entity, in samples. If video, set to -1

unsigned int iEntityType;

set to 1 if audio file, 3 if video file.

std::wstring wszFilename;

filename - can be either: 1) full path, or 2) just filename, if the entity is in the same folder as the session file.

Overview of SESSION_GROUP_INFO_NEW

```
struct SESSION_GROUP_INFO_NEW
{
   int iIndex;
   int iColor;
};
```

The SESSION_GROUP_INFO_NEW struct describes a clip group. There can be up to 63 clip groups in a session.

SESSION_GROUP_INFO_NEW members:

---

int iIndex;

The group index id. Valid indexes are from 1 to 63. This matches up with SESSION_COMMON_BLOCK_INFO_NEW::iParentGroup

---

int iColor;

The color value found in the Group Color dialog (Clip menu > Group Color). default is 186.

Overview of SESSION_BLOCK_INFO_NEW

```
struct SESSION_BLOCK_INFO_NEW
{
   std::vector<SESSION_COMMON_BLOCK_INFO_NEW> commonBlockInfo;
   std::vector<SESSION_WAVE_BLOCK_INFO_NEW> waveBlockInfo;
   std::vector<SESSION_VIDEO_BLOCK_INFO_NEW> videoBlockInfo;
};
```

The SESSION_BLOCK_INFO_NEW struct contains information about audio and video blocks (clips). Each block will have specific block info (wave or video) data, and also have common block info. The specific data structures link to the common block structure by the dwBlockID data items in each structure. So if there are 5 audio block data structures and 1 video block data structure, then there should be 6 common block structures.

SESSION_BLOCK_INFO_NEW members:

std::vector<SESSION_COMMON_BLOCK_INFO_NEW> commonBlockInfo;

The common block (clip) information. Every item in this list should have a corresponding SESSION_WAVE_BLOCK_INFO_NEW or SESSION_VIDEO_BLOCK_INFO_NEW.

std::vector<SESSION_WAVE_BLOCK_INFO_NEW> waveBlockInfo;

Audio-specific block information. Every item in this list should have a corresponding SESSION_COMMON_BLOCK_INFO_NEW.

std::vector<SESSION_VIDEO_BLOCK_INFO_NEW> videoBlockInfo;

Video-specific block information. Every item in this list should have a corresponding SESSION_COMMON_BLOCK_INFO_NEW.

Overview of SESSION_COMMON_BLOCK_INFO_NEW

```
struct SESSION_COMMON_BLOCK_INFO_NEW
{
   unsigned long dwBlockID;
   unsigned long dwEntityID;
   double fLeftVol;
   double fRightVol;
   __int64 dwOffset;
   __int64 dwSize;
   __int64 dwFileIndent;
   unsigned long dwFlags;
   int iTrack;
   int iParentGroup;
   int iPunchGeneration;
   int iPreviousPunch;
   int iNextPunch;
   int iHue;
   int iParentBlock;
};
```

The SESSION_COMMON_BLOCK_INFO_NEW struct contains information that is applicable to every block (clip) in a session. This information is used in combination with the corresponding SESSION_WAVE_BLOCK_INFO_NEW or SESSION_VIDEO_BLOCK_INFO_NEW information to provide the full information on a wave block or video block.

SESSION_COMMON_BLOCK_INFO_NEW members:

---

unsigned long dwBlockID;

block id - this matches up with SESSION_WAVE_BLOCK_INFO_NEW::dwBlockID or SESSION_VIDEO_BLOCK_INFO_NEW::dwBlockID

unsigned long dwEntityID;

entity id - this matches up with SESSION_ENTITY_INFO_NEW::dwWaveformID

double fLeftVol;

volume multiplier (for normal volume, set to 1. for +6 dB, set to 2. for -6 dB, set to 0.5. etc...) fLeftVol and fRightVol should be identical.

double fRightVol;

volume multiplier (for normal volume, set to 1. for +6 dB, set to 2. for -6 dB, set to 0.5. etc...) fLeftVol and fRightVol should be identical.

__int64 dwOffset;

where the block starts (number of samples from beginning of session)

__int64 dwSize;

the size of the block (in samples)

__int64 dwFileIndent;

where in the entity does this block start (offset in samples)

unsigned long dwFlags;

see block flags, below

```
#define F_BLOCK_OPAQUE          0x0001   // If set, any blocks hidden by this block overlapping are not played
#define F_BLOCK_LOCKED          0x0002   // Locked in time
#define F_BLOCK_MUTE            0x0004   // Muted
#define F_BLOCK_NORECORD        0x0008   // if track marked for record, a NORECORD will NOT be recorded into
#define F_BLOCK_RECORD          0x0010   // if track marked for playback, a RECORD will record into waveblock anyways
#define F_BLOCK_PUNCHIN         0x0020   // if recording into this track, we must first save current, and create new blank
#define F_BLOCK_TOUCHED         0x0040   // used internally - do not use
#define F_BLOCK_UNUSED          0x0080   // used internally - do not use
#define F_BLOCK_JUSTRECORDED    0x0100   // used internally - do not use
#define F_BLOCK_READINADJUST    0x0200   // used internally - do not use
#define F_BLOCK_GROWABLE        0x0400   // if set, block grows automatically while being recorded in to
#define F_BLOCK_HIDDEN          0x0800   // set when a block is part of a punch-in list but is not the currently viewed block
#define F_BLOCK_TEMPOLOCK       0x1000   // If set, and tempo changes, block moves to match
#define F_BLOCK_TEMPOLENLOCK    0x2000   // If set, and tempo changes, block length changes to match
```

int iTrack;

track id - this matches up with SESSION_COMMON_TRACK_INFO_NEW::id

int iParentGroup;

group id - this can be from 1 to 63 if using a group. Set to 0 if not using a group.

int iPunchGeneration;

set to 0

int iPreviousPunch;

set to 0

int iNextPunch;

set to 0

int iHue;

the color value found in the Clip Color dialog (Clip menu > Clip Color). audio block default is 102. video block default is 140.

int iParentBlock;

set to 0

Overview of SESSION_WAVE_BLOCK_INFO_NEW

```
struct SESSION_WAVE_BLOCK_INFO_NEW
{
   unsigned long dwBlockID;
   __int64 iFadeInOffset;
   __int64 iFadeOutOffset;
   int iFadeInShapeVal;
   int iFadeOutShapeVal;
   int iFadeInType;
   int iFadeOutType;
   bool hasCrossfadeLeft;
   bool hasCrossfadeRight;
   int iCrossfadeLeftLinkType;
   int iCrossfadeRightLinkType;
};
```

The SESSION_WAVE_BLOCK_INFO_NEW struct contains information that is applicable to every audio block (clip) in a session. This information is used in combination with the corresponding SESSION_COMMON_BLOCK_INFO_NEW to provide the full information on a wave block.

SESSION_WAVE_BLOCK_INFO_NEW members:

---

unsigned long dwBlockID;

---

id - this matches up with SESSION_COMMON_BLOCK_INFO_NEW::dwBlockID

__int64 iFadeInOffset;

---

length (in samples) of the on-clip fade in. (set to 0 for no fade in)

__int64 iFadeOutOffset;

---

length (in samples) of the on-clip fade out. (set to 0 for no fade out)

---

int iFadeInShapeVal;

---

shape of fade in (range: -100 to 100)

---

int iFadeOutShapeVal;

---

shape of fade out (range: -100 to 100)

---

int iFadeInType;

---

set to 0 for a linear/log fade, 1 for a cosine fade

---

int iFadeOutType;

---

set to 0 for a linear/log fade, 1 for a cosine fade

---

bool hasCrossfadeLeft;

---

true if has a crossfade with a different block on the left side

---

bool hasCrossfadeRight;

---

true if has a crossfade with a different block on the right side

---

int iCrossfadeLeftLinkType;

---

0 = not linked, 1 = linked constant gain, 2 = linked symmetrical

---

int iCrossfadeRightLinkType;

---

0 = not linked, 1 = linked constant gain, 2 = linked symmetrical

Overview of SESSION_VIDEO_BLOCK_INFO_NEW

```
struct SESSION_VIDEO_BLOCK_INFO_NEW
{
   unsigned long dwBlockID;
   unsigned short dwFlags;
   std::wstring wszAudioFileName;
};
```

The SESSION_VIDEO_BLOCK_INFO_NEW struct contains information that is applicable to every video block (clip) in a session. This information is used in combination with the corresponding SESSION_COMMON_BLOCK_INFO_NEW to provide the full information on a video block.

SESSION_VIDEO_BLOCK_INFO_NEW members:

---

unsigned long dwBlockID;

---

id - this matches up with SESSION_COMMON_BLOCK_INFO_NEW::dwBlockID

---

unsigned short dwFlags;

---

set to 1 if wszAudioFileName specifies an audio file that was extracted from this video

---

std::wstring wszAudioFileName;

---

this should be the filename of the audio file that was extracted from the video file. If the audio was extracted from the video, but not specifically saved, simply append "Audio for " to the beginning of the video filename for use here. (video file is "MyVideo.avi", so this would be set to "Audio for MyVideo.avi")

Overview of SESSION_BLOCK_ENVELOPE_GROUP_INFO_NEW

```
struct SESSION_BLOCK_ENVELOPE_GROUP_INFO_NEW
{
   unsigned long dwID;
   std::vector<SESSION_BLOCK_ENVELOPE_INFO_NEW> envelopes;
   __int64 rightMostSampleInEnvelope;
};
```

The SESSION_ BLOCK_ENVELOPE_GROUP_INFO_NEW struct contains information about block envelopes. Every block in the session is expected to have an envelope group. For video blocks, the 'envelopes' member should be empty. For audio blocks, the 'envelopes' member should contain 1 envelope for volume and 1 envelope for pan.

SESSION_ BLOCK_ENVELOPE_GROUP_INFO_NEW members:

---

unsigned long dwID;

For block envelopes - Identifies the block(clip) that these envelopes are for - id matches up with SESSION_COMMON_BLOCK_INFO_NEW::dwBlockID. This struct can also be used to describe envelopes in the surround encoder. For track envelops in the surround encoder (multichannel encoder), this id should match up to SESSION_MULTICHANNELENCODER_TRACK_INFO_NEW::iTrack

---

std::vector<SESSION_BLOCK_ENVELOPE_INFO_NEW> envelopes;

envelope info. For more information, see SESSION_BLOCK_ENVELOPE_INFO_NEW, below.

__int64 rightMostSampleInEnvelope;

This should be set to the largest envelops::points::dwAt value. (pretty much the same as the length of the longest envelope)

Overview of SESSION_BLOCK_ENVELOPE_INFO_NEW

```
struct SESSION_BLOCK_ENVELOPE_INFO_NEW
{
   unsigned int uiType;
   bool bSplined;
   std::vector<SESSION_ENVELOPE_POINT_INFO_NEW> points;
};
```

The SESSION_ BLOCK_ENVELOPE_INFO_NEW struct contains information about an envelope.

SESSION_ BLOCK_ENVELOPE_INFO_NEW members:

---

unsigned int uiType;

This defines the type of envelope. See below for types.

```
#define F_ET_VOL   0x01    // Volume (or Front/Back for surround encoder envelopes - (1.0 = front, 0.0 =
                              back))
#define F_ET_PAN   0x02    // Pan (or Left/Right for surround encoder envelopes - (1.0 = left, 0.0 = right))
#define F_ET_DRY   0x04    // Wet/Dry (obsolete - Audition will create this envelope type for backwards
                              compatibility, but it can be ignored)
```

---

bool bSplined;

Whether or not the envelope is splined

---

std::vector<SESSION_ENVELOPE_POINT_INFO_NEW> points;

This contains all the envelope points in the envelope. For more information, see the description of SESSION_ENVELOPE_POINT_INFO_NEW, above.

Overview of SESSION_LOOP_INFO_NEW

```
struct SESSION_LOOP_INFO_NEW
{
   unsigned long dwID;
   double fNumBeats;
   double fNativeBpm;
   int iBaseNote;
   bool bLoopSimple;
   bool bLoopInSamples;
   unsigned long dwLoopSamples;
   bool bLoopInBeats;
   double fLoopEveryNBeats;
   bool bFollowSessionTempo;
   double fLoopTempo;
   bool bTempoMatchFixed;
   bool bTempoMatchStretch;
   bool bTempoMatchResample;
   bool bTempoMatchBeatSplice;
   int iBeatSpliceRate;
   bool bCrossfade;
   double fSessionTempoUsed;
   int iStretchQuality;
   double fStretchSplicesPerBeat;
   double fStretchOverlapping;
   bool bBeatSpliceAutoFind;
   double fBeatSpliceDb;
   double fBeatSpliceMs;
   unsigned long dwOrigFileIndent;
   double fTranspose;
   bool bLoopOnce;
};
```

The SESSION_LOOP_INFO_NEW struct contains information that corresponds to settings found in the Loop Properties dialog (Clip menu > Loop Properties).

SESSION_LOOP_INFO_NEW members:

---

unsigned long dwID;

Block ID for the block (clip) that is associated with this loop information. This id matches up to SESSION_COMMON_BLOCK_INFO_NEW::dwBlockID

---

double fNumBeats;

Corresponds to the "Number of Beats" field in the Loop Properties dialog

---

double fNativeBpm;

Corresponds to the "Tempo" field in the Loop Properties dialog

---

int iBaseNote;

Corresponds to the "Key" field in the Loop Properties dialog. Use the kSessionBaseNote enum (more info on the kSessionBaseNote enum can be found with the description of SESSION_TEMPO_INFO_NEW)

bool bLoopSimple;

Corresponds to the "Simple Looping" option in the Loop Properties dialog (both bLoopInSamples and bLoopInBeats should both be false)

bool bLoopInSamples;

Corresponds to the "Repeat every N seconds" option in the Loop Properties dialog (both bLoopSimple and bLoopInBeats should be false)

unsigned long dwLoopSamples;

How often to repeat (in samples) for the "Repeat every N seconds" option in the Loop Properties dialog

bool bLoopInBeats;

Corresponds to the "Repeat every N beats" option in the Loop Properties dialog (both bLoopInSamples and bLoopSimple should be false)

double fLoopEveryNBeats;

Corresponds to the "seconds" field for the "Repeat every N beats" option in the Loop Properties dialog

bool bFollowSessionTempo;

Corresponds to the "Follow Session Tempo" field in the Loop Properties dialog

double fLoopTempo;

Corresponds to the "beats" field for the "Repeat every N beats" option in the Loop Properties dialog

bool bTempoMatchFixed;

Corresponds to the "Fixed Length" setting for the "Tempo Matching" option in the Loop Properties dialog (if set to true, bTempoMatchStretch, bTempoMatchResample, and bTempoMatchBeatSplice should be false)

bool bTempoMatchStretch;

Corresponds to the "Time-Scale Stretch" setting for the "Tempo Matching" option in the Loop Properties dialog (if set to true, bTempoMatchFixed, bTempoMatchResample, and bTempoMatchBeatSplice should be false) (*see note below about "Hybrid" option)

bool bTempoMatchResample;

Corresponds to the "Resample" setting for the "Tempo Matching" option in the Loop Properties dialog (if set to true, bTempoMatchFixed, bTempoMatchStretch, and bTempoMatchBeatSplice should be false)

bool bTempoMatchBeatSplice;

Corresponds to the "Beat Splice" setting for the "Tempo Matching" option in the Loop Properties dialog (if set to true, bTempoMatchFixed, bTempoMatchStretch, and bTempoMatchResample should be false) (*see note below about "Hybrid" option)

Hybrid: If bTempoMatchBeatSplice and bTempoMatchStretch are both true, then Audition uses the "Hybrid" setting for Tempo Matching

int iBeatSpliceRate;

if bTempoMatchFixed or bTempoMatchStretch, set to 4. if bTempoMatchResample, set to 1. if bTempoMatchBeatSplice, set to 2. if both bTempoMatchStretch and bTempoMatchBeatSplice (hybrid), set to 4.

bool bCrossfade;

obsolete. Set to false.

double fSessionTempoUsed;

This should be set to the same as SESSION_TEMPO_INFO_NEW::fBeatsPerMinute

int iStretchQuality;

Corresponds to the "Quality" setting for the "Tempo Matching" option (available when "Time-Scale Stretch" or "Resample" is selected)

double fStretchSplicesPerBeat;

Corresponds to the "Solo Voice / Instrument" field for the "Time-Scale Stretch" setting for the "Tempo Matching" option in the Loop Properties dialog. (if non-zero, then true (checked), if zero, then false (unchecked)

double fStretchOverlapping;

Corresponds to the "Preserve Formant" field for the "Time-Scale Stretch" setting for the "Tempo Matching" option in the Loop Properties dialog. (if non-zero, then true (checked), if zero, then false (unchecked)

bool bBeatSpliceAutoFind;

Corresponds to the "Auto-find beats" option of the "Beat Splice" setting for the "Tempo Matching" option in the Loop Properties dialog. If true, "Auto-find beats" is selected. If false, "Use file's beat markers" is selected.

double fBeatSpliceDb;

Corresponds to the "dB" field for the "Auto-find beats" option of the "Beat Splice" setting for the "Tempo Matching" option in the Loop Properties dialog

double fBeatSpliceMs;

Corresponds to the "ms" field for the "Auto-find beats" option of the "Beat Splice" setting for the "Tempo Matching" option in the Loop Properties dialog

unsigned long dwOrigFileIndent;

If bLoopOnce is false, then unused: set to 0. If bLoopOnce is true, then this setting corresponds to the clip indent (in samples) of the original, unstretched source. (SESSION_COMMON_BLOCK_INFO_NEW::dwFileIndent will contain the file indent, relative to the new stretched audio)

double fTranspose;

Corresponds to the "Transpose" field in the Loop Properties dialog.

bool bLoopOnce;

If set, instead of being a looped-clip, the clip will be a time-stretch clip. Most other settings still apply. dwLoopSamples is used to determine the time stretch. (if clip was originally 5000 samples, and stretched 110%, then dwLoopSamples would be set to 5500)

Overview of SESSION_CUE_INFO_NEW

```
struct SESSION_CUE_INFO_NEW
{
   unsigned long dwType;
   __int64 qiOffset;
   __int64 qiLength;
   std::wstring wszName;
   std::wstring wszDesc;
};
```

The SESSION_CUE_INFO_NEW struct contains information about any cues (now called markers) in the session.

SESSION_CUE_INFO_NEW members:

---

unsigned long dwType;

The marker type. (see type defines, below)

```
#define CUECCTYPE(a, b, c, d) (((DWORD)d<<24) | ((DWORD)c<<16) | ((DWORD)b<<8) | ((DWORD)a))

#define CUECC_RGN     CUECCTYPE('r', 'g', 'n', ' ')          // Standard cue/marker
#define CUECC_TRAK    CUECCTYPE('t', 'r', 'a', 'k')          // CD track marker
#define CUECC_INDX    CUECCTYPE('i', 'n', 'd', 'x')          // CD index marker
```

__int64 qiOffset;

The marker position (in samples)

__int64 qiLength;

The marker length (in samples)

---

std::wstring wszName;

The marker name

---

std::wstring wszDesc;

The marker description

Overview of SESSION_METRONOME_INFO_NEW

```
struct SESSION_METRONOME_INFO_NEW
{
   int              iSampleRate;
   double           fBPM;
   double           fOffsetMs;
   double           fVolume
   int              iTickSpacing;
   int              iOffset;
   int              iSet;
   bool             bEnable;
   std::wstring     szDefaultTime;
   std::wstring     szPattern;
   unsigned long    dwOutputPort;
   int              iNoteLen;
};
```

The SESSION_METRONOME_INFO_NEW struct contains information about the metronome settings. Much of this data is duplicated in the SESSION_TEMPO_INFO_NEW. Other settings match up to the metronome info in the Advanced Session Properties dialog > Metronome tab.

SESSION_METRONOME_INFO_NEW members:

int iSampleRate;

This should match the session sample rate (SESSION_MAIN_INFO_NEW::iSampleRate)

double fBPM;

This should match SESSION_TEMPO_INFO_NEW::fBeatsPerMinutes

double fOffsetMs;

This should match SESSION_TEMPO_INFO_NEW::fBeatOffsetMs

double fVolume;

This matches up with the volume settings in the Advanced Session Properties dialog > Metronome tab.

int iTickSpacing;

The number of samples per tick (usually sampleRate / 2)

int iOffset;

The same measurement as fOffsetMs above, but in samples. (so if fOffsetMs is 100 (ms) and the sample rate is 44100, this should be 4410 (samples))

int iSet;

0-based index of the sound set dropdown list in the Advanced Session Properties dialog > Metronome tab.

bool bEnable;

Is the metronome turned on? True, yes. False, no.

std::wstring szDefaultTime;

The name of the metronome time signature (in the Advanced Session Properties dialog > Metronome tab.)

std::wstring szPattern;

The pattern used for the metronome (matches with the Pattern field in the Advanced Session Properties dialog > Metronome tab.)

unsigned long dwOutputPort;

Output port to use on the current audio device

int iNoteLen;

This should match SESSION_TEMPO_INFO_NEW::iNoteLen

Overview of SESSION_MULTICHANNELENCODER_INFO_NEW

```
struct SESSION_MULTICHANNELENCODER_INFO_NEW
{
   std::vector<SESSION_MULTICHANNELENCODER_TRACK_INFO_NEW> tracks;
   std::wstring sessionName;
   double fMasterVolume;
};
```

The SESSION_MULTICHANNELENCODER_INFO_NEW struct contains information about the multichannel encoder (now called “surround encoder”).

SESSION_MULTICHANNELENCODER_INFO_NEW members:

std::vector<SESSION_MULTICHANNELENCODER_TRACK_INFO_NEW> tracks;

The list of tracks. There should be an item for every audio track, (including bus tracks, midi tracks, and rewire tracks), and the master track. For more details see the description of SESSION_MULTICHANNELENCODER_TRACK_INFO_NEW, below.

std::wstring sessionName;

unused - just leave blank

double fMasterVolume;

Matches with "Master Level" in the Surround Encoder dialog. (for 0dB, set to 100.0, for +6dB, set to 200.0, for -infdB, set to 0.0)

Overview of SESSION_MULTICHANNELENCODER_TRACK_INFO_NEW

```
struct SESSION_MULTICHANNELENCODER_TRACK_INFO_NEW
{
   bool bUseFlatLines;
   double fFlatFade;
   double fFlatPan;
   int iSubLevel;
   int iCenterLevel;
   int iTrackVolume;
   unsigned long dwActiveSpeakers;
   int iPanningMode;
   bool bFixedRouting;
   int iTrack;
   SESSION_BLOCK_ENVELOPE_GROUP_INFO_NEW envelopeGroup;
};
```

The SESSION_MULTICHANNELENCODER_TRACK_INFO_NEW struct contains information for a single track in the multichannel encoder (surround encoder).

SESSION_MULTICHANNELENCODER_TRACK_INFO_NEW members:

---

bool bUseFlatLines;

inversely matches "Pan Envelopes" setting in the Surround Encoder dialog (set to false if "Pan Envelopes" should be checked)

double fFlatFade;

if bUseFlatLines is true, this is used to determine front / back relationship (1.0 = front, 0.0 = back)

int iSubLevel;

sub level (0 to 100)

int iCenterLevel;

center level (0 to 100)

int iTrackVolume;

overall track volume (0 to 100)

unsigned long dwActiveSpeakers;

bit field of active speakers (see speakers below)

```
#define SPEAKER_FRONT_LEFT            0x1
#define SPEAKER_FRONT_RIGHT           0x2
#define SPEAKER_FRONT_CENTER          0x4
#define SPEAKER_LOW_FREQUENCY         0x8
#define SPEAKER_BACK_LEFT             0x10
#define SPEAKER_BACK_RIGHT            0x20
```

int iPanningMode;

special panning mode (0 for stereo, 1 for summed-to-mono)

bool bFixedRouting;

if true, all output goes to active speakers (otherwise use panning envelopes)

int iTrack;

0-based index of track, as visible in the multitrack ui (master track is always the last track, so it has the highest index number).   (not to be confused with SESSION_AUDIO_TRACK_FULL_INFO_NEW::trackOrderIndex, which uses a 1-based index, with the master track being 0)

SESSION_BLOCK_ENVELOPE_GROUP_INFO_NEW envelopeGroup;

envelopes for the track - each track should have two envelops in this group, one for front/back panning, one for left/right panning. For more details, see the description of SESSION_BLOCK_ENVELOPE_GROUP_INFO_NEW, above.