SRAPI Reference Manual

Joel D. Miller

This reference manual describes the SLAB Render API. For programmers new to SLAB, a good place to start is the "File List" link in the header above. This link will display the header files included in the SLAB User Release for writing SLAB applications and SLAB Render plug-ins. (Note: The header files SLABHost.h (CSLABHost interface) and SLABClient.h (CSLABClient interface) are not included in the SLAB User Release. This is because CSLABHost and CSLABClient are accessed through the CSLABAPI interface.) Next, see the "Class Hierarchy" link for a description of the SRAPI classes and structures.

The documentation generation tool *doxygen* was used to generate the reference manual from source code comments.

See also:

SLAB User Manual

Copyright 2001,2002 U.S. Government as represented by the Administrator of the National Aeronautics and Space Administration.

SRAPI Reference Manual Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- CRPlugIn
- CScene
- CSLABAPI
  - CSLABClient
  - CSLABHost
- Desc
- Polar
- Position
- Stats

SRAPI Reference Manual Compound List

Here are the classes, structs, unions and interfaces with brief descriptions:

CRPlugIn	SLAB Render plug-in base class
CScene	SLAB scenario and scene parameter interface
CSLABAPI	User's primary interface to SLAB
CSLABClient	Client interface to SLAB
CSLABHost	Host interface to SLAB
Desc	Plug-in description struct
Polar	Polar location structure
Position	Position (location and orientation) structure
Stats	SLAB statistics structure

SRAPI Reference Manual File List

Here is a list of all documented files with brief descriptions:

rplugin.h	CRPlugIn class interface
scene.h	CScene class interface
SLABAPI.h	Abstract base class interface for the CSLABAPI class
SLABAPIClient.h	CSLABClient object allocation support routines
SLABAPIHost.h	CSLABHost object allocation support routines
SLABClient.h	CSLABClient class interface
SLABDEFS.h	SLAB defines and types
SLABHost.h	CSLABHost class interface

SRAPI Reference Manual Compound Members

a | b | c | e | f | g | i | l | m | n | o | p | r | s | u | x | y | z

Here is a list of all documented class members with links to the classes they belong to:

- a -

- az : Polar

- b -

- Begin() : CRPlugIn

- c -

- Callback() : CSLABHost, CSLABAPI
- Close() : CSLABAPI

- e -

- el : Polar
- End() : CRPlugIn
- EnvMaterial() : CSLABAPI
- EnvPlanes() : CSLABAPI
- Error() : CSLABAPI
- ErrorClear() : CSLABAPI, CRPlugIn
- ErrorSet() : CRPlugIn
- ErrorStack() : CSLABAPI
- ErrorState() : CSLABAPI, CRPlugIn
- ErrorString() : CSLABAPI, CRPlugIn

- f -

- FIRPoints() : CSLABAPI

- g -

- GetClips() : CRPlugIn
- GetDefHRTF() : CSLABAPI
- GetDelayIn() : CScene
- GetEnvMaterial() : CScene
- GetEnvSndSpd() : CScene
- GetFrameSize() : CScene
- GetHRTFDir() : CSLABAPI
- GetInstDir() : CSLABAPI
- GetLogDir() : CSLABAPI
- GetLstPosition() : CSLABHost, CSLABAPI, CScene
- GetOutL() : CScene
- GetOutR() : CScene
- GetSampleRate() : CScene
- GetSmoothTime() : CScene
- GetSrcGain() : CSLABHost, CSLABAPI
- GetSrcID() : CSLABHost, CSLABAPI
- GetSrcLocation() : CSLABHost, CSLABAPI
- GetSrcNum() : CSLABHost, CSLABAPI, CScene
- GetSrcRadius() : CScene
- GetSrcRel() : CSLABHost, CSLABAPI
- GetSrcSpread() : CScene
- GetStats() : CSLABAPI
- GetVersion() : CSLABAPI
- GiAz() : CScene
- GiData() : CScene
- GiEl() : CScene
- GiEmpty() : CScene
- GiEnabled() : CScene
- GiFIRn() : CScene

- GiGain() : CScene
- GiHRTF() : CScene
- GiIn() : CScene
- GiRange() : CScene
- GiRefID() : CScene
- GiX() : CScene
- GiY() : CScene
- GiZ() : CScene

- i -

- Init() : CRPlugIn

- l -

- lClip : Stats
- LogEntry() : CSLABHost, CSLABAPI
- LogName() : CSLABHost, CSLABAPI
- LogTime() : CSLABHost, CSLABAPI
- LstHRTF() : CSLABAPI
- LstHRTFFree() : CSLABAPI
- LstHRTFLoad() : CSLABAPI
- LstPosition() : CSLABAPI
- LstSensorOffset() : CSLABAPI
- lUnderflow : Stats

- m -

- m_bError : CRPlugIn
- m_lClip : CRPlugIn
- m_pScene : CRPlugIn
- m_strError : CRPlugIn
- MessageBox() : CSLABClient, CSLABAPI

- n -

- nDateDay : Desc
- nDateMonth : Desc
- nDateYear : Desc
- NextImage() : CScene
- NextSource() : CScene
- nPIVersion : Desc
- nVerMajor : Desc
- nVerMinor : Desc
- nVerRelease : Desc

- o -

- Open() : CSLABAPI
- OutDS() : CSLABAPI
- OutMemory() : CSLABAPI

- p -

- PCount() : CSLABAPI
- PCountReset() : CSLABAPI
- pitch : Position
- Process() : CRPlugIn

- r -

- r : Polar
- Recalc() : CScene
- RenderChange() : CSLABAPI
- RenderEnum() : CSLABAPI
- RenderID() : CSLABAPI
- RenderStart() : CSLABAPI
- RenderStop() : CSLABAPI
- Reset() : CSLABAPI
- ResetImages() : CScene
- roll : Position

- s -

- ScriptAdd() : CSLABAPI
- ScriptBegin() : CSLABAPI
- ScriptRead() : CSLABAPI
- ScriptWrite() : CSLABAPI
- SetDefHRTF() : CSLABAPI
- SetHRTFDir() : CSLABAPI
- SetNotify() : CSLABAPI
- SiData() : CScene
- SmoothTime() : CSLABAPI
- SrcEnable() : CSLABAPI
- SrcFile() : CSLABAPI
- SrcFree() : CSLABAPI
- SrcGain() : CSLABAPI
- SrcGenImpulse() : CSLABAPI
- SrcGenNoise() : CSLABAPI
- SrcGenSine() : CSLABAPI
- SrcGenSquare() : CSLABAPI
- SrcLocate() : CSLABAPI
- SrcLocatePolar() : CSLABAPI
- SrcRadius() : CSLABAPI
- SrcRefEnable() : CSLABAPI
- SrcRefOffset() : CSLABAPI
- SrcSpread() : CSLABAPI
- SrcTrajEnable() : CSLABAPI
- SrcTrajLMod() : CSLABAPI
- SrcTrajLStart() : CSLABAPI
- SrcTrajPMod() : CSLABAPI
- SrcTrajPStart() : CSLABAPI
- SrcTrajRate() : CSLABAPI
- SrcTrajStart() : CSLABAPI
- SrcTrajStop() : CSLABAPI
- strAuthor : Desc

- strDesc : Desc
- strName : Desc

- u -

- Updated() : CScene

- x -

- x : Position

- y -

- y : Position
- yaw : Position

- z -

- z : Position

SRAPI Reference Manual generated Thu Dec 19 14:06:11 2002 by doxygen 1.2.17

SRAPI Reference Manual File Members

d | e | f | i | m | p | r | s

Here is a list of all documented file members with links to the documentation:

- d -

- dDeg2Rad : SLABDEFS.h
- DESC_VERSION : rplugin.h
- dGainMutedB : SLABDEFS.h
- dIn2M : SLABDEFS.h
- dINTERAURALin : SLABDEFS.h
- dINTERAURALm : SLABDEFS.h
- dM2In : SLABDEFS.h
- dPI : SLABDEFS.h
- dPId2 : SLABDEFS.h
- dPIx2 : SLABDEFS.h
- dRad2Deg : SLABDEFS.h
- dSENSOROFFSETin : SLABDEFS.h
- dSENSOROFFSETm : SLABDEFS.h
- dSLAB_FS : SLABDEFS.h
- dSoundSpeed : SLABDEFS.h

- e -

- ERROR_STRINGS : SLABDEFS.h

- f -

- fDeg2Rad : SLABDEFS.h
- fIn2M : SLABDEFS.h
- fINTERAURALin : SLABDEFS.h
- fINTERAURALm : SLABDEFS.h
- fM2In : SLABDEFS.h

- fPI : SLABDEFS.h
- fPId2 : SLABDEFS.h
- fPIx2 : SLABDEFS.h
- fRad2Deg : SLABDEFS.h
- fSENSOROFFSETin : SLABDEFS.h
- fSENSOROFFSETm : SLABDEFS.h

- i -

- IDRender : SLABDEFS.h
- IDSS : SLABDEFS.h

- m -

- MAT_CARPET : SLABDEFS.h
- MAT_CONCRETE : SLABDEFS.h
- MAT_GLASS : SLABDEFS.h
- MAT_GYPSUM : SLABDEFS.h
- MAT_NONE : SLABDEFS.h
- MAT_NUM : SLABDEFS.h
- MAT_PLASTER : SLABDEFS.h
- MAT_WOODAIR : SLABDEFS.h
- MATERIAL_STRINGS : SLABDEFS.h
- MaterialType : SLABDEFS.h

- p -

- PIESL : rplugin.h
- Planes : SLABDEFS.h
- POS_PCH : SLABDEFS.h
- POS_ROL : SLABDEFS.h
- POS_X : SLABDEFS.h
- POS_Y : SLABDEFS.h
- POS_YAW : SLABDEFS.h
- POS_Z : SLABDEFS.h
- PositionIndex : SLABDEFS.h

- r -

- RENDER_DIOTIC : SLABDEFS.h
- RENDER_DIOTIC_V : SLABDEFS.h
- RENDER_MAX : SLABDEFS.h
- RENDER_MONOTIC_L : SLABDEFS.h
- RENDER_MONOTIC_R : SLABDEFS.h
- RENDER_PLUGIN : SLABDEFS.h
- RENDER_SPATIAL : SLABDEFS.h
- RVB_ALL : SLABDEFS.h
- RVB_PATHS : SLABDEFS.h
- RVB_PLANES : SLABDEFS.h
- RVB_XN : SLABDEFS.h
- RVB_XP : SLABDEFS.h
- RVB_YN : SLABDEFS.h
- RVB_YP : SLABDEFS.h
- RVB_ZN : SLABDEFS.h
- RVB_ZP : SLABDEFS.h

- s -

- SLAB_FS : SLABDEFS.h
- SLABAPIClient() : SLABAPIClient.h
- SLABAPIClientError() : SLABAPIClient.h
- SLABAPIClientErrorStack() : SLABAPIClient.h
- SLABAPIClientErrorString() : SLABAPIClient.h
- SLABAPIHost() : SLABAPIHost.h
- SLABAPIHostError() : SLABAPIHost.h
- SLABAPIHostErrorStack() : SLABAPIHost.h
- SLABAPIHostErrorString() : SLABAPIHost.h
- SLABError : SLABDEFS.h
- SS_EnvPlanes : SLABDEFS.h
- SS_LstPosition : SLABDEFS.h
- SS_SrcEnable : SLABDEFS.h
- SS_SrcGain : SLABDEFS.h

- SS_SrcLocate : SLABDEFS.h
- SS_SrcLocatePolar : SLABDEFS.h
- SS_SrcRefEnable : SLABDEFS.h
- SS_SrcTrajLMod : SLABDEFS.h
- SS_SrcTrajLStart : SLABDEFS.h
- SS_SrcTrajPStart : SLABDEFS.h
- SS_SrcTrajStart : SLABDEFS.h
- SS_SrcTrajStop : SLABDEFS.h
- SS_Tail : SLABDEFS.h

Polar Struct Reference

List of all members.

Detailed Description

The Polar structure contains elements for defining a location using polar coordinates.

Public Attributes

double az
azimuth

double el
elevation

double r
range

CRPlugIn Class Reference

List of all members.

Detailed Description

The CRPlugIn class is the base class of SLAB Render plug-ins. Render plug-ins perform all audio rendering in SLAB. They are given access to SLAB's scenario parameters, scene information, input delay lines, and output stream.

See also:

CScene

Header: rplugin.h
Library: none

Error Functions

void ErrorClear ()
clears error state

char * ErrorString ()
returns error string

bool ErrorState ()
returns error state

bool ErrorSet (char *strError)
sets error string

Public Methods

Plug-In Functions

virtual bool Init (CScene *pScene, void **ppDesc)=0
performs one-time constructor initialization

virtual bool Process (void)=0
processes a frame of input samples

virtual bool Begin (void)
performs plugged-in initialization

virtual bool End (void)
frees plugged-in allocation

Query Functions
unsigned long GetClips (void)
returns the clipping statistic

Protected Attributes

Plug-In Attributes
CScene * m_pScene
scenario and scene parameter access

unsigned long m_lClip
clip statistic

Private Attributes

Error Attributes
char m_strError [PIESL]
for internal use only

bool m_bError
for internal use only

Member Function Documentation

bool CRPlugIn::ErrorSet(char * *strError*) `[inline, protected]`

The ErrorSet() function sets the plug-in error string, signaling an error. SLAB will catch the error, stop rendering, and report the error to the user. Plug-in developers should call this function when they need to signal an error condition.

Returns :

false = error, true = no error

See also:

Error Functions

Parameters:

strError error string (max length = PIESL)

void CRPlugIn::ErrorClear( ) `[inline]`

The ErrorClear() function clears the error state. It is called by SLAB when the SRAPI user clears the error. Plug-in developers should not call this function.

Returns :

none

See also:

Error Functions

char* CRPlugIn::ErrorString( ) `[inline]`

The ErrorString() function returns the error string. It is called by SLAB to retrieve the error string when a plug-in signals an error. Plug-in developers should not call this function.

Returns :

error string

See also:

Error Functions

bool CRPlugIn::ErrorState( ) `[inline]`

The ErrorState() function returns the current error state.

Returns :

false = error, true = no error

See also:

Error Functions

```
virtual bool CRPlugIn::Init( CScene *    pScene,
                             void **     ppDesc
                           ) [pure virtual]
```

The Init() function performs one-time constructor initialization. The plug-in developer must override this function to: (1) initialize the *m_pScene* member variable and (2) return a description struct. This function can be used in place of a constructor to set defaults based on default scenario parameters or to return error information.

All plug-ins are loaded when the CSLABHost and CSLABServer objects are allocated. After a plug-in is loaded, its Init function is called to initialize the plug-in. This is the only time the Init() function is called. Initialization that must occur every time a plug-in is plugged-in should be performed in Begin().

Returns :

true = success, false = failure

See also:

Plug-In Functions, CScene, Desc

Parameters:

pScene CScene pointer passed to Init

ppDesc description struct returned to SLAB

virtual bool CRPlugIn::Process(void) `[pure virtual]`

The Process() function processes a frame of input samples. The plug-in developer must override this function to implement their own renderer. Except for extremely simple renderers, the following code should be included at the beginning of Process:

```
if( m_pScene->Updated() )
{
  m_pScene->Recalc();
  if( !SceneToSigProc() )
    return false;
}
```

This code fragment checks if any scenario parameters have been updated. If so, the scene is recalculated (e.g. new listener-relative parameters are computed). The plug-in developer then calls their own function (SceneToSigProc()) to convert the scene parameters to their renderer's signal processing parameters (e.g. FIR coefficients).

The plug-in accesses the scenario and scene parameters, the input delay lines, and the output stream through the CScene object. This is discussed further in the CScene documentation.

Returns :

true = success, false = failure

See also:

Plug-In Functions, CScene

virtual bool CRPlugIn::Begin(void) `[inline, virtual]`

The Begin() function is called every time the plug-in is plugged-in. Override this function to initialize member variables that keep track of renderer state and to allocate image data (see CScene).

Returns :

true = success, false = failure

See also:

Plug-In Functions, Init, End()

virtual bool CRPlugIn::End(void) `[inline, virtual]`

The End() function is called every time the plug-in is plugged-out. Override this function to free data allocated during Begin().

Returns :

true = success, false = failure

See also:

Plug-In Functions, Begin()

unsigned long CRPlugIn::GetClips(void) `[inline]`

The GetClips() function returns the clipping statistic (see *m_lClip*).

Returns :

the number of clips

See also:

Query Functions, m_lClip

Member Data Documentation

char CRPlugIn::m_strError[PIESL] `[private]`

m_strError is for internal use only. This member variable should only be accessed through the Error Functions.

bool CRPlugIn::m_bError `[private]`

m_bError is for internal use only. This member variable should only be accessed through the Error Functions.

CScene* CRPlugIn::m_pScene `[protected]`

m_pScene provides access to SLAB's scenario and scene parameters. The plug-in developer must initialize this member variable in their Init() override.

unsigned long CRPlugIn::m_lClip `[protected]`

The use of this variable is up to the Render plug-in developer. Should you choose to use it, initialize it to 0 in your Begin() override and increment it as appropriate in your Process override. The SRAPI developer can get access to this information via the host-mode function GetStats().

CSLABHost Class Reference

Inheritance diagram for CSLABHost:

List of all members.

Detailed Description

The CSLABHost class provides host-mode access to the SLAB renderer. In host-mode, the SLAB renderer is executed locally on the user's machine. This mode is best suited for demos and SLAB application development. For complex environments, it can be very difficult to sustain low-latency audio output since SLAB competes for system resources with other applications. When SLAB failes to keep up with audio output, an underflow error occurs and SLAB stops rendering. Thus, when rendering a computationally expensive environment, host-mode should be avoided for critical applications (e.g. psychoacoustic experimentation).

Host-mode provides a few capabilities client/server-mode does not. There are several Get functions for querying scenario state information and a set of Log functions for recording SLAB status information and user log messages.

CSLABHost objects are allocated with the SLABAPIHost support function declared in SLABAPIHost.h.

Note:

The user interacts with the CSLABHost object through the CSLABAPI class interface.

See also:

CSLABAPI
SLABAPIHost.h

Header: SLABAPI.h
Library: slabh, slabs

Public Methods

Callback Function

void Callback (void(*pCallback)(void *pPrm), void *pParam, int nPeriod)

sets a user callback routine

Query Functions

Position * GetLstPosition ()

gets the current listener position

Position * GetSrcLocation (IDSRC idSrc)
gets the location of the requested source

Polar * GetSrcRel (IDSRC idSrc)
gets the listener-relative location of the requested source

double GetSrcGain (IDSRC idSrc)
gets the linear gain scalar of the requested source

int GetSrcNum ()
gets the number of allocated sources

IDSRC GetSrcID (int nSrc)
gets the IDSRC of the requested source number

Log Functions

void LogName (char *pLogName)
sets log filename

void LogEntry (char *pLogEntry)
logs a user message

void LogTime ()
logs the current system time

Member Function Documentation

void CSLABHost::Callback(void(* *pCallback*)(void *pPrm),
void * *pParam*,
int *nPeriod*
) `[inline, virtual]`

The Callback() function allows the user to specify a callback routine to be called from within the rendering frame update loop. This is primarily useful for scenarios requiring a high update rate (e.g. a custom trajectory) and/or sound source sample synchronization (e.g. non-real-time rendering). The period of the callback call will be normalized to a multiple of the frame size (32 samples).

Note:

The callback and scripting mechanisms are mutually exclusive!

Returns :

none

See also:

Host Functions

Parameters:

pCallback address of user callback routine

pParam address of user data
nPeriod period of callback call (samples)

Reimplemented from CSLABAPI.

Position* CSLABHost::GetLstPosition( ) `[inline, virtual]`

The GetLstPosition() function gets the current listener position.

Returns :

Position structure containing listener position

See also:

Query Functions
Position structure in SLABDEFS.h

Reimplemented from CSLABAPI.

Position* CSLABHost::GetSrcLocation(IDSRC *idSrc*) `[inline, virtual]`

The GetSrcLocation() function gets the location of the requested source.

Returns :

Position structure containing source location

See also:

Query Functions
Position structure in SLABDEFS.h

Parameters:

idSrc source ID

Reimplemented from CSLABAPI.

Polar* CSLABHost::GetSrcRel(IDSRC *idSrc*) `[inline, virtual]`

The GetSrcRel() function gets the listener-relative azimuth and elevation of the requested source.

Returns :

Polar structure containing listener-relative polar source location (the r field is not used)

See also:

Query Functions
Polar structure in SLABDEFS.h

Parameters:

idSrc source ID

Reimplemented from CSLABAPI.

double CSLABHost::GetSrcGain(IDSRC *idSrc*) `[inline, virtual]`

The GetSrcGain() function gets the linear gain scalar of the requested source.

Returns :

linear gain scalar

See also:

Query Functions

Parameters:

idSrc source ID

Reimplemented from CSLABAPI.

int CSLABHost::GetSrcNum( ) `[inline, virtual]`

The GetSrcNum() function gets the number of allocated sources.

Returns :

number of allocated sources

See also:

Query Functions

Reimplemented from CSLABAPI.

IDSRC CSLABHost::GetSrcID(int *nSrc*) `[inline, virtual]`

The GetSrcID() function gets the IDSRC of the requested source number. The source number refers to the order in which the sources were allocated (0 = first source allocated, 1 = second, and so on).

Returns :

IDSRC corresponding to *nSrc*

See also:

Query Functions

Parameters:

nSrc source number

Reimplemented from CSLABAPI.

void CSLABHost::LogName(char * *pLogName*) `[virtual]`

The LogFile() function specifies the file to use as a SLAB log file. If this function is not called, no log file is created. By default, the log file will log error information, but it can also log user log messages via LogEntry and LogTime. If the file already exists, new log entries are appended to the end of the file.

Returns :

none

See also:

Log Functions

Parameters:

pLogName name of SLAB log file

Reimplemented from CSLABAPI.

void CSLABHost::LogEntry(char * *pLogEntry*) `[inline, virtual]`

The LogEntry() function logs a user message in the SLAB log file.

Returns :

none

Remarks:

The log file is opened and closed for each log entry.

See also:

Log Functions

Parameters:

pLogEntry user log message

Reimplemented from CSLABAPI.

void CSLABHost::LogTime( ) `[inline, virtual]`

The LogTime() function logs the current system time in the SLAB log file.

Returns :

none

Remarks:

The log file is opened and closed for each log entry.

See also:

Log Functions

Reimplemented from CSLABAPI.

CSLABAPI Class Reference

Inheritance diagram for CSLABAPI:

List of all members.

Detailed Description

The SLAB API is encapsulated in the CSLABAPI class. The CSLABAPI class is the abstract base class for the CSLABHost and CSLABClient classes. This abstraction allows SLAB to run locally or remotely using the same interface. All host and client/server implementation details are hidden in the CSLABHost and CSLABClient classes. The host interface provides the following features the client does not: callback capabilities, scenario information querying, and logging (see CSLABHost).

Between RenderStart and RenderStop, SLAB is *processing*. Functions that cannot be called while processing are termed *non-process-time* functions. If these functions are called while processing, an error results. See the Remarks section of the function documentation to determine if a function is a non-process-time function. If a function is not a non-process-time function, it can be called at any time.

In client-server mode, functions that block waiting for a response from the server are termed *blocking* functions. Funtions that do not block are termed *non-blocking* functions.

(non-)process-time and (non-)blocking issues are discussed at greater length in the Error Handling section of the SLAB User Manual.

Allocation Example:

```
CSLABAPI*  m_pSLAB;

if( bHostMode )
  m_pSLAB = SLABAPIHost();
else
  m_pSLAB = SLABAPIClient( "10.0.0.1" );
```

(From this point on, the host-mode and client/server-mode interfaces are identical.)

Header: SLABAPI.h
Library: slabh, slabc, slabs

Public Methods

Render Functions

virtual SLABError RenderStart (int nID)=0
starts rendering

virtual SLABError RenderChange (int nID)=0
changes the rendered plug-in

virtual SLABError RenderStop ()=0

stops rendering

virtual Desc ** RenderEnum ()=0
enumerates render plug-ins

virtual int RenderID (char *pName)=0
returns the render ID corresponding to a renderer's name

Source Functions

virtual IDSRC SrcFile (char *pFilename, bool bLoop=true)=0
allocates a wave file sound source

virtual IDSRC SrcGenNoise (int nAmp)=0
allocates a Gaussian noise sound source

virtual IDSRC SrcGenSine (int nFreq, int nAmp)=0
allocates a sine wave sound source

virtual IDSRC SrcGenSquare (int nFreq, int nAmp)=0
allocates a square wave sound source

virtual IDSRC SrcGenImpulse (int nPeriod, int nAmp)=0
allocates an impulse train sound source

virtual SLABError SrcFree ()=0
frees all allocated sound sources

virtual SLABError SrcSpread (double dSrcSpread)=0
sets the amount of spherical spreading loss

virtual SLABError SrcRadius (double dSrcRadius)=0
sets the radius of all sound sources

virtual SLABError SrcLocate (IDSRC idSrc, double x, double y, double z)=0
sets the location of a sound source

virtual SLABError SrcLocatePolar (IDSRC idSrc, double az, double el, double r)=0
sets the location of a sound source relative to the origin

virtual SLABError SrcGain (IDSRC idSrc, double dDB)=0
sets the gain of a sound source

virtual SLABError SrcEnable (IDSRC idSrc, bool bEnable)=0
enables or disables a sound source

virtual SLABError SrcRefEnable (IDSRC idSrc, int nRef, bool bEnable)=0
enables or disables a sound source's reflection

virtual SLABError SrcRefOffset (IDSRC idSrc, int nRef, bool bRefOffset, double x=0.0, double y=0.0, double z=0.0)=0
enables or disables reflection offsets

Source Trajectory Functions

virtual SLABError SrcTrajEnable (bool bEnable)=0
enables or disables all source trajectories

virtual SLABError SrcTrajRate (double dUpdateRate)=0
sets the rate at which trajectories are updated

virtual SLABError SrcTrajLStart (IDSRC idSrc, double dX, double dY, double dZ, double dRateX, double dRateY, double dRateZ)=0
starts a linear trajectory

virtual SLABError SrcTrajLMod (IDSRC idSrc, double dRateX, double dRateY, double dRateZ)=0
modifies a linear trajectory

virtual SLABError SrcTrajPStart (IDSRC idSrc, double dAz, double dEl, double dR, double dRateAz, double dRateEl, double dRateR)=0
starts a polar trajectory

virtual SLABError SrcTrajPMod (IDSRC idSrc, double dRateAz, double dRateEl, double dRateR)=0
modifies a polar trajectory

virtual SLABError SrcTrajStart (IDSRC idSrc)=0
starts a previously stopped trajectory

virtual SLABError SrcTrajStop (IDSRC idSrc)=0
stops a source trajectory

Environment Functions

virtual SLABError EnvPlanes (double xp, double xn, double yp, double yn, double zp, double zn)=0
sets the locations of the planes

virtual SLABError EnvMaterial (int nPlane, MaterialType matType)=0
sets the plane material for a given plane

Listener Functions

virtual IDHRTF LstHRTFLoad (const char *pName)=0
loads an HRTF database

virtual SLABError LstHRTFFree (IDHRTF idHRTF)=0
frees a loaded HRTF database

virtual SLABError LstHRTF (IDHRTF idHRTF, IDSRC idSrc=NULL)=0
selects an HRTF database

virtual SLABError LstPosition (double x, double y, double z, double yaw, double pitch, double roll)=0
positions the listener

virtual SLABError LstSensorOffset (double x, double y, double z)=0
sets tracker sensor offsets

Scripting Functions

virtual SLABError ScriptBegin (double dUpdateRate)=0
enables script processing

virtual SLABError ScriptAdd (IDSS idSS, double dTime,...)=0
adds a script command

virtual SLABError ScriptRead (char *pFilename, IDSRC *pIDSrc, int nLen)=0
reads a SLAB script

virtual SLABError ScriptWrite (char *pFilename, IDSRC *pIDSrc, int nLen)=0
writes a SLAB script

Output Functions

virtual SLABError OutDS (int nOut, int nWrite)=0

sets the sound output destination to DirectSound

virtual SLABError OutMemory (char *pFilename)=0
sets the sound output destination to memory

Signal Processing Functions

virtual SLABError SmoothTime (double dSmoothTime)=0
sets the time constant of the parameter smoothing filter

virtual SLABError FIRPoints (int nFIRPoints)=0
sets the number of FIR points for HRIR filtering

Miscellaneous Functions

virtual SLABError SetNotify (CWnd *pWnd, WPARAM wParam)=0
sets the notification window and notification message

virtual void Reset ()=0
resets the state of SLAB

virtual Stats * GetStats ()=0
gets SLAB statistics

Error Functions

virtual char * ErrorString ()=0
returns the current SLAB error string

virtual char * ErrorStack ()=0
returns the SLAB error strack

virtual void ErrorClear ()
clears the current SLAB error

bool ErrorState ()
returns the SLAB error state

SLABError Error ()
returns the current SLAB error code

Host Functions

virtual void Callback (void(*pCallback)(void *pPrm), void *pParam, int nPeriod)
implemented in CSLABHost

virtual Position * GetLstPosition ()
implemented in CSLABHost

virtual IDSRC GetSrcID (int nSrc)
implemented in CSLABHost

virtual Position * GetSrcLocation (IDSRC idSrc)
implemented in CSLABHost

virtual Polar * GetSrcRel (IDSRC idSrc)
implemented in CSLABHost

virtual double GetSrcGain (IDSRC idSrc)
implemented in CSLABHost

virtual int GetSrcNum ()

implemented in CSLABHost

virtual void LogName (char *pLogName)
implemented in CSLABHost

virtual void LogEntry (char *pLogEntry)
implemented in CSLABHost

virtual void LogTime ()
implemented in CSLABHost

Client Functions

virtual void MessageBox (bool bEnable)
implemented in CSLABClient

Server Functions

virtual SLABError Open (void(*pFuncLog)(char *strLogEntry))
implemented in CSLABServer

virtual void Close ()
implemented in CSLABServer

virtual unsigned long PCount ()
implemented in CSLABServer

virtual void PCountReset ()
implemented in CSLABServer

Static Public Methods

Registry Functions

bool GetVersion (char *pStr, int nSize)
gets the SLAB version number

bool GetInstDir (char *pStr, int nSize)
gets the SLAB installation directory

bool GetLogDir (char *pStr, int nSize)
gets the SLAB log directory

bool GetHRTFDir (char *pStr, int nSize)
gets the SLAB HRTF directory

bool GetDefHRTF (char *pStr, int nSize)
gets the SLAB default HRTF database

bool SetHRTFDir (const char *pStr)
sets the SLAB HRTF directory

bool SetDefHRTF (const char *pStr)
sets the SLAB default HRTF database

Member Function Documentation

```
bool CSLABAPI::GetVersion( char *   pStr,
                           int      nSize
                         ) [static]
```

The GetVersion() function retrieves the SLAB version number stored in the registry. The version number is formatted as follows: major.minor.release.

Returns :

true = success, false = failure

Remarks:

This function exists for the host, client, and server, but it only checks the registry of the local machine. Thus, in client/server mode, it retrieves the registry information of the client, not the server.

See also:

Registry Functions

Parameters:

pStr buffer to receive null-terminated version string
nSize size of *pStr* (bytes)

```
bool CSLABAPI::GetInstDir( char *   pStr,
                           int      nSize
                         ) [static]
```

The GetInstDir() function retrieves the SLAB installation directory stored in the registry.

Returns :

true = success, false = failure

Remarks:

This function exists for the host, client, and server, but it only checks the registry of the local machine. Thus, in client/server mode, it retrieves the registry information of the client, not the server.

See also:

Registry Functions

Parameters:

pStr buffer to receive null-terminated installation directory string
nSize size of *pStr* (bytes)

```
bool CSLABAPI::GetLogDir( char *   pStr,
                          int      nSize
                        ) [static]
```

The GetLogDir() function retrieves the SLAB log directory stored in the registry.

Returns :

true = success, false = failure

Remarks:

This function exists for the host, client, and server, but it only checks the registry of the local machine. Thus, in client/server mode, it retrieves the registry information of the client, not the server.

See also:

Registry Functions

Parameters:

pStr buffer to receive null-terminated log directory string
nSize size of *pStr* (bytes)

```
bool CSLABAPI::GetHRTFDir( char *   pStr,
                           int      nSize
                         ) [static]
```

The GetHRTFDir() function retrieves the SLAB HRTF directory stored in the registry.

Returns :

true = success, false = failure

Remarks:

This function exists for the host, client, and server, but it only checks the registry of the local machine. Thus, in client/server mode, it retrieves the registry information of the client, not the server.

See also:

SetHRTFDir(), Registry Functions

Parameters:

pStr buffer to receive null-terminated HRTF directory string
nSize size of *pStr* (bytes)

```
bool CSLABAPI::GetDefHRTF( char *   pStr,
                           int      nSize
                         ) [static]
```

The GetDefHRTF() function retrieves the SLAB default HRTF database stored in the registry.

Returns :

true = success, false = failure

Remarks:

This function exists for the host, client, and server, but it only checks the registry of the local machine. Thus, in client/server mode, it retrieves the registry information of the client, not the server.

See also:

SetDefHRTF(), Registry Functions

Parameters:

pStr buffer to receive null-terminated default HRTF database string
nSize size of *pStr* (bytes)

bool CSLABAPI::SetHRTFDir(const char * *pStr*) `[static]`

The SetHRTFDir() function sets the SLAB HRTF directory stored in the registry.

Returns :

true = success, false = failure

Remarks:

This function exists for the host, client, and server, but it only sets the registry of the local machine. Thus, in client/server mode, it sets the registry key of the client, not the server.

See also:

GetHRTFDir(), Registry Functions

Parameters:

pStr null-terminated default HRTF directory string

bool CSLABAPI::SetDefHRTF(const char * *pStr*) `[static]`

The SetDefHRTF() function sets the SLAB default HRTF database stored in the registry.

Returns :

true = success, false = failure

Remarks:

This function exists for the host, client, and server, but it only sets the registry of the local machine. Thus, in client/server mode, it sets the registry key of the client, not the server.

See also:

GetDefHRTF(), Registry Functions

Parameters:

pStr null-terminated default HRTF database string

virtual SLABError CSLABAPI::RenderStart(int *nID*) `[pure virtual]`

The RenderStart() function initiates rendering. *nID* specifies either a SLAB render ID (RENDER_SPATIAL, RENDER_DIOTIC, RENDER_MONOTIC_L, RENDER_MONOTIC_R) or an ID returned by RenderEnum(). Renderers are encapsulated in SLAB render plug-ins and can be developed by the user.

The pre-defined SLAB Render plug-in IDs:

RENDER_SPATIAL: The SLAB Spatial renderer is the default renderer and renders all of the scenario parameters specified by the SLAB Render API.

RENDER_DIOTIC_V: DioticV supports diotic, left-monotic, and right-monotic auditory display. The display type for each source is determined from the source's Y-axis location: + = left-monotic, 0.0 = diotic, and - = right-monotic. DioticV ignores the Listener, Environment, and DSP API parameters. It also ignores the following Source parameters: SrcSpread(), SrcRadius(), and SrcRefEnable().

RENDER_DIOTIC: Diotic renders a diotic auditory display. All source are panned equally to the listener's left and right ears. Diotic renders the same set of scenario parameters as DioticV.

RENDER_MONOTIC_L: Left-Monotic renders a left-monotic auditory display. All source are panned to the listener's left ear. Left-Monotic renders the same set of scenario parameters as DioticV.

RENDER_MONOTIC_R: Right-Monotic supports right-monotic auditory display. All source are panned to the listener's right ear. Right-Monotic renders the same set of scenario parameters as DioticV.

Returns :

SLABError

Remarks:

blocking function, non-process-time function

See also:

Render Functions

Parameters:

nID specifies the type of rendering to perform

virtual SLABError CSLABAPI::RenderChange(int *nID*) `[pure virtual]`

The RenderChange() function changes the Render plug-in. *nID* specifies either a SLAB render ID (RENDER_SPATIAL, RENDER_DIOTIC, RENDER_MONOTIC_L, RENDER_MONOTIC_R) or an ID returned by RenderEnum(). Renderers are encapsulated in SLAB render plug-ins and can be developed rendered by the user.

Returns :

SLABError

Remarks:

non-blocking function, process-time function

If RenderChange() is called before processing begun with RenderStart(), RenderChange() does nothing.

See also:

Render Functions

Parameters:

nID specifies the type of rendering to perform

virtual SLABError CSLABAPI::RenderStop( ) `[pure virtual]`

If the SLAB sound output type was set to memory with OutMemory(), the sound data in memory is written to a file during RenderStop().

See also:

Render Functions

virtual Desc** CSLABAPI::RenderEnum( ) `[pure virtual]`

The RenderEnum() function enumerates render plug-ins by returning an array of plug-in description struct pointers. When CSLABHost or CSLABServer are allocated, all plug-ins found in the <install dir>\bin\rplugin directory are loaded into memory and all of the plug-in descriptions are placed in an array. The user can iterate through this array to query information about the available renderers. The description list is terminated by a NULL *Desc** array entry. Do NOT free the description structs.

Returns :

array of *Desc* pointers corresponding to the available renderers

Remarks:

blocking function

See also:

Render Functions

virtual int CSLABAPI::RenderID(char * *pName*) `[pure virtual]`

The RenderID() function returns the render ID corresponding to a render plug-in's name. The name is case-sensitive.

Returns :

integer ID >= 0 if name found, -1 if name not found

Remarks:

blocking function

See also:

Render Functions

Parameters:

pName case-sensitive name of render plug-in

virtual IDSRC CSLABAPI::SrcFile(char * *pFilename*,
bool *bLoop* = true
) `[pure virtual]`

The SrcFile() function allocates a wave file sound source. Only the first channel in the wave file is used. If one-shot playback is selected, the user is notified of playback completion via a notification message (see SetNotify()).

Returns :

IDSRC source ID. Error information can be retrieved by calling Error().

Remarks:

blocking function, non-process-time function

See also:

Source Functions

Parameters:

pFilename file to play
bLoop if true, loop file when playing

virtual IDSRC CSLABAPI::SrcGenNoise(int *nAmp*) `[pure virtual]`

The SrcGenNoise() function allocates a Gaussian noise sound source. The noise samples are generated using a variant of the linear congruential method.

Returns :

IDSRC source ID. Error information can be retrieved by calling Error().

Remarks:

blocking function, non-process-time function

See also:

Source Functions

Parameters:

nAmp max amplitude (0-32767)

```
virtual IDSRC CSLABAPI::SrcGenSine( int    nFreq,
                                    int    nAmp
                                  ) [pure virtual]
```

The SrcGenSine() function allocates a sine wave sound source.

Returns :

IDSRC source ID. Error information can be retrieved by calling Error().

Remarks:

blocking function, non-process-time function

See also:

Source Functions

Parameters:

nFreq frequency (Hertz)
nAmp max amplitude (0-32767)

```
virtual IDSRC CSLABAPI::SrcGenSquare( int    nFreq,
                                      int    nAmp
                                    ) [pure virtual]
```

The SrcGenSquare() function allocates a square wave sound source.

Returns :

IDSRC source ID. Error information can be retrieved by calling Error().

Remarks:

blocking function, non-process-time function

See also:

Source Functions

Parameters:

nFreq frequency (Hertz)
nAmp max amplitude (0-32767)

```
virtual IDSRC CSLABAPI::SrcGenImpulse( int    nPeriod,
                                       int    nAmp
                                     ) [pure virtual]
```

The SrcGenImpulse() function allocates an impulse train sound source.

Returns :

IDSRC source ID. Error information can be retrieved by calling Error().

Remarks:

blocking function, non-process-time function

See also:

Source Functions

Parameters:

nPeriod period (samples)
nAmp amplitude (0-32767)

virtual SLABError CSLABAPI::SrcFree( ) `[pure virtual]`

The SrcFree() function frees all allocated sound sources. Calling this function while processing results in an error. SrcFree() should be called before changing the set of allocated sound sources.

Returns :

SLABError

Remarks:

blocking function, non-process-time function

See also:

Source Functions

virtual SLABError CSLABAPI::SrcSpread(double *dSrcSpread*) `[pure virtual]`

The SrcSpread() function sets the amount of spherical spreading loss for all sound sources. A SLAB sound source is modeled as a cylindrical piston (e.g. a speaker cone) in order to determine its spreading loss. The source spread exponent affects the rate at which the spreading gain decreases as the source-listener range increases. The default value is 1.0. Although a spread exponent other than 1.0 does not have a physical interpretation, a value higher than 1.0 can be used to exaggerate the spherical spreading effect. A spread exponent of 0.0 disables spherical spreading loss.

Spherical spreading loss is defined by the following formula:

GainScalar = (1 + (Range / SrcRadius)^2)^(-SrcSpread/2)
By setting SrcRadius = dINTERAURALm/2 and SrcSpread = 1, this equation can be made to approximate 1/Range point source behavior. If the radius of the head is set as the 0db reference, the 1/Range formula becomes:

[a] (dINTERAURALm/2)/Range

With the substitutions mentioned above, the SLAB formula becomes:

[b] (dINTERAURALm/2) / ((dINTERAURALm/2)^2 + Range^2)^(1/2)

When Range >> (dINTERAURALm/2), [b] approximates [a].

Returns :

SLABError

Remarks:

non-blocking function

See also:

SrcRadius(), Source Functions, spread.m slabtool

Parameters:

dSrcSpread source spread exponent (see formula)

virtual SLABError CSLABAPI::SrcRadius(double *dSrcRadius*) `[pure virtual]`

The SrcRadius() function sets the source radius of all sound sources. The source radius parameter affects the spherical spreading loss model which controls the rate at which source gain decreases as the source-listener range increases. See SrcSpread() for a more thorough discussion of spherical spreading loss.

The default value of source radius is 10cm.

Returns :

SLABError

Remarks:

non-blocking function

See also:

SrcSpread(), Source Functions, spread.m slabtool

Parameters:

dSrcRadius radius of sound source (meters)

virtual SLABError CSLABAPI::SrcLocate(IDSRC *idSrc*,
double *x*,
double *y*,
double *z*
) `[pure virtual]`

The SrcLocate() function locates a sound source in world coordinates.

Returns :

SLABError

Remarks:

non-blocking function

Due to sound sample delay line length limitations, the world should not be considered infinite. Since SLAB uses a delay line of 500ms, the coordinate values should not exceed +/- 45 meters.

See also:

Source Functions

Parameters:

idSrc sound source ID
x x-axis coordinate (meters)
y y-axis coordinate (meters)
z z-axis coordinate (meters)

virtual SLABError CSLABAPI::SrcLocatePolar(IDSRC *idSrc*,
double *az*,
double *el*,
double *r*
) `[pure virtual]`

The SrcLocatePolar() function locates a sound source in origin-relative polar world coordinates.

Returns :

SLABError

Remarks:

non-blocking function

Due to sound sample delay line length limitations, the world should not be considered infinite. Since SLAB uses a delay line of 500ms, the range coordinate value should not exceed 45 meters.

See also:

Source Functions

Parameters:

idSrc	sound source ID
az	azimuth coordinate (radians)
el	elevation coordinate (radians)
r	range coordinate (meters)

```
virtual SLABError CSLABAPI::SrcGain( IDSRC    idSrc,
                                     double   dDB
                                   ) [pure virtual]
```

The SrcGain() function specifies the gain applied to sound source samples as they are pulled from the input delay line. This gain value is analogous to a source volume control.

Returns :

SLABError

Remarks:

non-blocking function

Source gain is one of two API gain parameters when using SLAB signal generation. The source generator gains specified by the SrcGen functions are applied before the input delay line. Source gain is applied after the input delay line and before the spatialization FIR filter.

See also:

Source Functions

Parameters:

idSrc	sound source ID
dDB	exponential gain (dB)

```
virtual SLABError CSLABAPI::SrcEnable( IDSRC    idSrc,
                                       bool     bEnable
                                     ) [pure virtual]
```

The SrcEnable() function enables or disables a sound source. SrcEnable() is similar to a mixer's mute button. When a source is disabled, it is no longer rendered, thus decreasing CPU load.

Returns :

SLABError

Remarks:

non-blocking function

See also:

SrcRefEnable(), Source Functions

Parameters:

idSrc	sound source ID
bEnable	if true, the source is enabled, if false, the source is disabled

```
virtual SLABError CSLABAPI::SrcRefEnable( IDSRC   idSrc,
                                          int     nRef,
                                          bool    bEnable
                                         ) [pure virtual]
```

The SrcRefEnable() function enables or disables a sound source's reflection. When a reflection is disabled, it is no longer processed, thus decreasing CPU load.

Returns :

SLABError

Remarks:

non-blocking function

See also:

SrcEnable(), Source Functions

Parameters:

idSrc	sound source ID
nRef	reflection identifier (see SLABDEFS.h RVB_*)
bEnable	if true, the reflection is enabled, if false, the reflection is disabled

```
virtual SLABError CSLABAPI::SrcRefOffset( IDSRC   idSrc,
                                          int     nRef,
                                          bool    bRefOffset,
                                          double  x = 0.0,
                                          double  y = 0.0,
                                          double  z = 0.0
                                         ) [pure virtual]
```

The SrcRefOffset() function enables or disables reflection offsets. The offsets will be added to the computed reflection image location.

Returns :

SLABError

Remarks:

non-blocking function

See also:

SrcRefEnable(), Source Functions

Parameters:

idSrc	sound source ID
nRef	reflection identifier (see SLABDEFS.h RVB_*)
bRefOffset	if true, reflection offsets are enabled, if false, reflection offsets are disabled
x	x offset amount
y	y offset amount
z	z offset amount

virtual SLABError CSLABAPI::SrcTrajEnable(bool *bEnable*) `[pure virtual]`

The SrcTrajEnable() function enables or disables all source trajectories. Trajectories are disabled by default.

Returns :

SLABError

Remarks:

non-blocking function

See also:

Source Trajectory Functions

Parameters:

bEnable	if true, enable all trajectories, if false, disable all trajectories

virtual SLABError CSLABAPI::SrcTrajRate(double *dUpdateRate*) `[pure virtual]`

The SrcTrajRate() function sets the rate at which trajectories are updated. This will be normalized to a multiple of the frame size (32 samples) (e.g. a specified update rate of 120Hz yields an actual update rate of 125.28Hz). The default update rate is 120Hz.

Returns :

SLABError

Remarks:

non-blocking function

See also:

Source Trajectory Functions

Parameters:

dUpdateRate	trajectory update rate (Hz)

```
virtual SLABError CSLABAPI::SrcTrajLStart( IDSRC    idSrc,
                                           double   dX,
                                           double   dY,
                                           double   dZ,
                                           double   dRateX,
                                           double   dRateY,
                                           double   dRateZ
                                          ) [pure virtual]
```

The SrcTrajLStart() function starts and defines a linear trajectory for a specified source.

Warning:

Be sure to stop the trajectory before it causes a delay line overflow (see SrcLocate())!

Returns :

SLABError

Remarks:

non-blocking function

See also:

Source Trajectory Functions

Parameters:

idSrc	sound source ID
dX	initial x-axis value (meters)
dY	initial y-axis value (meters)
dZ	initial z-axis value (meters)
dRateX	x-axis rate (meters/s)
dRateY	y-axis rate (meters/s)
dRateZ	z-axis rate (meters/s)

```
virtual SLABError CSLABAPI::SrcTrajLMod( IDSRC    idSrc,
                                         double   dRateX,
                                         double   dRateY,
                                         double   dRateZ
                                        ) [pure virtual]
```

The SrcTrajLStart() function modifies the rates of a previously defined linear trajectory.

Returns :

SLABError

Remarks:

non-blocking function

See also:

Source Trajectory Functions

Parameters:

idSrc	sound source ID
dRateX	new x-axis rate (meters/s)
dRateY	new y-axis rate (meters/s)
dRateZ	new z-axis rate (meters/s)

```
virtual SLABError CSLABAPI::SrcTrajPStart( IDSRC    idSrc,
                                           double   dAz,
                                           double   dEl,
                                           double   dR,
                                           double   dRateAz,
                                           double   dRateEl,
                                           double   dRateR
                                          ) [pure virtual]
```

The SrcTrajPStart() function starts and defines a polar trajectory for a specified source.

Warning:

Be sure to stop the trajectory before it causes a delay line overflow (see SrcLocatePolar())!

Returns :

SLABError

Remarks:

non-blocking function

See also:

Source Trajectory Functions

Parameters:

idSrc	sound source ID
dAz	initial azimuth value (radians)
dEl	initial elevation value (radians)
dR	initial range value (meters)
dRateAz	azimuth rate (radians/s)
dRateEl	elevation rate (radians/s)
dRateR	range rate (meters/s)

```
virtual SLABError CSLABAPI::SrcTrajPMod( IDSRC    idSrc,
                                         double   dRateAz,
                                         double   dRateEl,
                                         double   dRateR
                                        ) [pure virtual]
```

The SrcTrajPMod() function modifies the rates of a previously defined polar trajectory.

Returns :

SLABError

Remarks:

non-blocking function

See also:

Source Trajectory Functions

Parameters:

idSrc	sound source ID
dRateAz	new azimuth rate (radians/s)
dRateEl	new elevation rate (radians/s)
dRateR	new range rate (meters/s)

```
virtual SLABError CSLABAPI::SrcTrajStart( IDSRC   idSrc) [pure virtual]
```

The SrcTrajStart() function starts a previously stopped trajectory for a specified source.

Returns :

SLABError

Remarks:

non-blocking function

See also:

Source Trajectory Functions

Parameters:

idSrc sound source ID

virtual SLABError CSLABAPI::SrcTrajStop(IDSRC *idSrc*) `[pure virtual]`

The SrcTrajStop() function stops a source trajectory.

Returns :

SLABError

Remarks:

non-blocking function

See also:

Source Trajectory Functions

Parameters:

idSrc sound source ID

virtual SLABError CSLABAPI::EnvPlanes(double *xp*,
double *xn*,
double *yp*,
double *yn*,
double *zp*,
double *zn*
) `[pure virtual]`

The EnvPlanes() function sets the locations of the planes (walls) in the rendered environment. The room model consists of a rectangular room with six walls. Each plane location is defined by the signed distance to the origin along the Cartesian axis to which the plane is perpendicular. The default plane locations are (5,-5,5,-5,5,-5).

Returns :

SLABError

Remarks:

non-blocking function

See also:

Environment Functions

Parameters:

xp positive x-axis plane location (meters)
xn negative x-axis plane location (meters)
yp positive y-axis plane location (meters)
yn negative y-axis plane location (meters)
zp positive z-axis plane location (meters)
zn negative z-axis plane location (meters)

virtual SLABError CSLABAPI::EnvMaterial(int *nPlane*, MaterialType *matType*) `[pure virtual]`

The EnvMaterial() function sets the plane material for a given plane. Materials are rendered using a first-order IIR filter. The IIR filter responses approximate the absorption coefficients listed below. The material type MAT_NONE disables material processing for the given plane. The default material for all planes is MAT_NONE.

```
Absorption Coefficients (percentage absorbed energy):

Material                      125   250   500   1000  2000  4000 Hz
--------                      ----  ----  ----  ----  ----  ----
heavy carpet / foam rubber    0.08  0.24  0.57  0.69  0.71  0.73
concrete                      0.01  0.012 0.017 0.02  0.02  0.02
heavy glass                   0.18  0.06  0.04  0.03  0.02  0.02
2 layer gypsum board          0.28  0.12  0.10  0.17  0.13  0.09
1" thick wood with airspace   0.19  0.14  0.09  0.06  0.06  0.05
plaster on metal lath         0.14  0.10  0.06  0.05  0.04  0.03
```

Materials are identified by the MaterialType enum. Strings corresponding to the MaterialType enum values can be found in the MATERIAL_STRINGS define in SLABDEFS.h.

Returns :

SLABError

Remarks:

blocking function, non-process-time function

See also:

Environment Functions

Parameters:

nPlane plane identifier (see Planes enum in SLABDEFS.h, RVB_XP thru RVB_ZN)

matType material type identifier

virtual IDHRTF CSLABAPI::LstHRTFLoad(const char * *pName*) `[pure virtual]`

The LstHRTFLoad() function loads a Head-Related Transfer Function (HRTF) database, returning an HRTF ID. This database is used by some renderers for sound source spatialization. All HRTFs loaded with LstHRTFLoad() should be freed with LstHRTFFree().

Returns :

HRTF ID, NULL if in error state or unable to load HRTF

Remarks:

blocking function

LstHRTFLoad() does not set SLAB error state. If the filename is specified via a user interface, this behavior makes it easier to recover from a bad filename.

The number of HRTF files that can be loaded at the same time is limited to 100 due to the size of the HRTF ID garbage collection array.

See also:

Listener Functions, LstHRTFFree(), LstHRTF()

Parameters:

pName HRTF database filename

virtual SLABError CSLABAPI::LstHRTFFree(IDHRTF *idHRTF*) `[pure virtual]`

The LstHRTFFree() function frees a previously loaded Head-Related Transfer Function (HRTF) database. All HRTFs loaded with LstHRTFLoad() should be freed with LstHRTFFree().

Returns :

SLABError

Remarks:

non-blocking function

See also:

Listener Functions, LstHRTFLoad(), LstHRTF()

Parameters:

idHRTF HRTF ID

virtual SLABError CSLABAPI::LstHRTF(IDHRTF *idHRTF*,
IDSRC *idSrc* = NULL
) `[pure virtual]`

The LstHRTF() function selects the Head-Related Transfer Function (HRTF) database used to render the listener's head and ears. In typical use, only the `idHRTF` parameter need be specified. For research purposes, an optional parameter, `idSrc`, allows an HRTF to be attached to a sound source. This allows different sound sources to be listened to with different HRTFs.

Returns :

SLABError

Remarks:

non-blocking function

See also:

Listener Functions, LstHRTFLoad(), LstHRTFFree()

Parameters:

idHRTF HRTF ID
idSrc sound source ID

virtual SLABError CSLABAPI::LstPosition(double *x*,
double *y*,
double *z*,
double *yaw*,
double *pitch*,
double *roll*
) `[pure virtual]`

The LstPosition() function positions the listener in world coordinates. The listener position is a six degree of freedom vector composed of x, y, z *location* components and yaw, pitch, roll *orientation* components. The default listener position is (0,0,0,0,0,0).

Returns :

SLABError

Remarks:

non-blocking function

Due to sound sample delay line length limitations, the world should not be considered infinite. Since SLAB uses a delay line of 500ms, the listener location coordinate values should not exceed +/- 45 meters.

See also:

Listener Functions

Parameters:

x	x-axis location coordinate (meters)
y	y-axis location coordinate (meters)
z	z-axis location coordinate (meters)
yaw	yaw orientation coordinate (radians)
pitch	pitch orientation coordinate (radians)
roll	roll orientation coordinate (radians)

virtual SLABError CSLABAPI::LstSensorOffset(double *x*, double *y*, double *z*) `[pure virtual]`

The LstSensorOffset() function sets the x, y, z coordinate offsets of the head tracker sensor from the center of the head. This function allows head tracker location data to be passed directly to LstPosition(). The default sensor offset is (0,0,0).

Returns :

SLABError

Remarks:

non-blocking function

See also:

Listener Functions

Parameters:

x	x-axis sensor offset (meters)
y	y-axis sensor offset (meters)
z	z-axis sensor offset (meters)

virtual SLABError CSLABAPI::ScriptBegin(double *dUpdateRate*) `[pure virtual]`

The ScriptBegin() function enables script processing and sets the update rate at which the script is parsed. The update rate will be normalized to a multiple of the frame size (32 samples) (e.g. an update rate parameter of 120Hz yields an actual update rate of 125.28Hz). ScriptBegin() will free a previously defined script (i.e. only one script exists at a time). Before calling RenderStart() and after calling ScriptBegin(), use the ScriptAdd() function to add script commands. Once RenderStart() is called, the time is set to 0 and the script parsed. Scripting is disabled when rendering stops.

Warning:

The callback and scripting mechanisms are mutually exclusive!

Returns :

SLABError

Remarks:

blocking function, non-process-time function

See also:

Scripting Functions

Parameters:

dUpdateRate script parsing update rate (Hz)

```
virtual SLABError CSLABAPI::ScriptAdd( IDSS      idSS,
                                       double    dTime,
                                       ...
                                      ) [pure virtual]
```

The ScriptAdd() function adds a script command to the current script. Available script commands are defined by IDSS and correspond to a subset of the SRAPI. The script command parameters are the same as the corresponding SRAPI function.

Warning:

Script commands must be added as ordered in time (i.e. command at 1s added before command at 2s)!

The callback and scripting mechanisms are mutually exclusive!

Returns :

SLABError

Remarks:

blocking function, non-process-time function

See also:

Scripting Functions

Parameters:

idSS script command
dTime time index of script command (seconds)

```
virtual SLABError CSLABAPI::ScriptRead( char *     pFilename,
                                        IDSRC *    pIDSrc,
                                        int        nLen
                                       ) [pure virtual]
```

The ScriptRead() function reads a previously saved SLAB script.

Note:

When using this function in client/server mode, the number of sources is limited to 32. This is because all SLAB packets are of fixed size.

Returns :

SLABError

Remarks:

blocking function, non-process-time function

See also:

Scripting Functions

Parameters:

pFilename	script filename
pIDSrc	sound source ID to integer index mapping array; the first IDSRC specified when written will be mapped to the first IDSRC specified when read
nLen	*pIDSrc* array length

```
virtual SLABError CSLABAPI::ScriptWrite( char *     pFilename,
                                         IDSRC *    pIDSrc,
                                         int        nLen
                                        ) [pure virtual]
```

The ScriptWrite() function writes the current SLAB script.

Note:

When using this function in client/server mode, the number of sources is limited to 32. This is because all SLAB packets are of fixed size.

Returns :

SLABError

Remarks:

blocking function

See also:

Scripting Functions

Parameters:

pFilename	script filename
pIDSrc	sound source ID to integer index mapping array; the first IDSRC specified when written will be mapped to the first IDSRC specified when read
nLen	*pIDSrc* array length

```
virtual SLABError CSLABAPI::OutDS( int     nOut,
                                   int     nWrite
                                  ) [pure virtual]
```

The OutDS() function sets the sound output destination to DirectSound. The output device opened is the Windows preferred sound playback device set using the Windows Sounds and Multimedia Properties settings dialog (Audio | Sound Playback | Preferred device). DirectSound is the default sound output destination.

Buffer Settings

The larger the output buffer, the longer the internal latency. If the buffer is too small, sample underflow occurs. The goal is to select the smallest buffer size possible while maintaining reliable sample output. The default output buffer size is 4096 bytes.

The write buffer is written to and filled prior to writing to the DirectSound output buffer. The default write buffer size is 512 bytes.

The write buffer exists to optimize DirectSound output buffer management. Writing to the DirectSound output buffer is more computationally expensive than writing to memory. Thus, it is beneficial to queue up some samples before writing to the DirectSound output buffer. How many samples to queue up, however, is not obvious. The default write buffer size of 512 bytes has yielded acceptable results for DirectSound output buffer sizes of 4096 and 8192 bytes.

Warning:

Results vary between different DirectSound peripherals and drivers. Common problems include clicks'n'pops, frequent buffer underflow, and unsupported output (e.g. analog supported, SPDIF not supported).

Returns :

SLABError

Remarks:

blocking function, non-process-time function
A rough estimate of SLAB's internal latency can be calculated by dividing the output buffer size by 176400 bytes/s ((44100 samples/s * 2 bytes/sample)/channel * 2 channels).

See also:

Output Functions

Parameters:

nOut output buffer size (bytes)
nWrite write buffer size (bytes)

virtual SLABError CSLABAPI::OutMemory(char * *pFilename*) `[pure virtual]`

The OutMemory() function selects memory as the destination for SLAB sound output. When rendering stops, the memory samples are saved to a file.

Returns :

SLABError

Remarks:

blocking function, non-process-time function

OutMemory() is primarily a diagnostic function used for SLAB development. It is used to capture very small segments (a few seconds) of output. OutMemory()'s memory management is not robust. Using OutMemory() to capture extended segments (several minutes) of SLAB output is not advised.

See also:

Output Functions

Parameters:

pFilename file for saving SLAB memory output

virtual SLABError CSLABAPI::SmoothTime(double *dSmoothTime*) `[pure virtual]`

The SmoothTime() function sets the time constant of SLAB's leaky integrator parameter smoothing (aka tracking) filter. This parameter adjusts how quickly the DSP parameters change in response to changing scenario parameters. The default smooth time is 25 ms.

Returns :

SLABError

Remarks:

blocking function, non-process-time function

A smooth time of zero yields the most responsive system, but audible artifacts (clicks, zippering) will likely be heard as scenario parameters change. Large values (tens of milleseconds) yield audibly smooth systems, but may seem sluggish.

See also:

Signal Processing Functions

Parameters:

dSmoothTime smoothing filter time constant (milliseconds)

virtual SLABError CSLABAPI::FIRPoints(int *nFIRPoints*) `[pure virtual]`

The FIRPoints() function sets the number of FIR points used when performing HRIR filtering. Since this is the most computationally intensive operation in the SLAB Spatial renderer, reducing the number of FIR points is an easy way to free up computational resources. The tradeoff is audio fidelity. Reflections are rendered at a lower fidelity, floor(*nFIRPoints* / 4) points. For HRIR filter sizes smaller than 128, the filters are truncated. The default number of FIR points is 128.

Returns :

SLABError

Remarks:

blocking function, non-process-time function

See also:

Signal Processing Functions

Parameters:

nFIRPoints number of FIR points for HRIR filtering

virtual SLABError CSLABAPI::SetNotify(CWnd * *pWnd*,
WPARAM *wParam*
) `[pure virtual]`

The SetNotify() function sets the notification window and notification message used by SLAB to notify the user's thread of process-time events.

Returns :

SLABError

Remarks:

non-blocking function, non-process-time function

SetNotify() is one of the rare cases where a non-process-time function is a non-blocking function. This is because, in client-server mode, SetNotify() does not communicate with the server.

Currently, the user thread is only notified when an error occurs or when one-shot playback completes.

See also:

Error Functions, SrcFile

Parameters:

pWnd	the user's window for receiving notification messages
wParam	the message ID sent to the user's notification window

virtual void CSLABAPI::Reset( ) `[pure virtual]`

The Reset() function resets the state of SLAB (rendering stopped, sources freed, rendering type set to RENDER_SPATIAL, output type set to device, DirectSound buffer sizes set to defaults, script freed).

Note: Prior to version 4.4.0, Reset() was called as part of the error handling process. Reset() is now omitted in an effort to preserve state. If a source error occurs, it might be necessary to call Reset() to eliminate the error.

Returns :

none

Remarks:

non-blocking function

Reset() is primarily for internal use. It is called during CSLABAPI construction and destruction.

See also:

Rendering Functions, Error Functions

virtual Stats* CSLABAPI::GetStats( ) `[pure virtual]`

The GetStats() function returns SLAB event statistics (e.g. DirectSound underflow). The use of Render plug-in statistics (e.g. mixer clips) is implementation-dependent. In general, statistics are reset in RenderStart().

Returns :

Stats structure containing SLAB statistics;
if SLAB was allocated as a client and is in an error state, existing statistics are returned (i.e. statistics are not updated from server)

See also:

Stats structure in SLABDEFS.h

virtual char* CSLABAPI::ErrorString( ) `[pure virtual]`

The ErrorString() function returns a string describing the current SLAB error condition.

Returns :

string describing the current SLAB error condition (see ERROR_STRINGS in SLABDEFS.h)

Remarks:

In client/server mode, the ErrorString() function returns the local SLAB error string; it does not communicate with the server. See Error Handling.

See also:

Error Functions

virtual char* CSLABAPI::ErrorStack( ) `[pure virtual]`

The ErrorStack() function is used to obtain detailed error information after an error occurs. It returns a string describing the error state at each point in the internal function calling sequence at the time of the error. This function is primarily for development and bug reporting.

Returns :

string listing the error state at each point in the internal function calling sequence at the time of the last error

Remarks:

In client/server mode, the ErrorStack() function returns the local SLAB error stack; it does not communicate with the server. See Error Handling.

See also:

Error Functions

virtual void CSLABAPI::ErrorClear( ) `[inline, virtual]`

The ErrorClear() function clears the current SLAB error. When SLAB encounters an error, it remains in an error state until the user clears the error. This prevents any further SRAPI functions from executing until the error is explicitly cleared by the SLAB user. This is necessary since the user's application can be in a scenario updating loop when an error occurs. Once the error occurs, all SRAPI functions do nothing but return the current error. Within a few iterations, the user's application will receive an error notification through the SetNotify() mechanism.

Returns :

none

Remarks:

non-blocking function

See also:

Error Functions

bool CSLABAPI::ErrorState( ) `[inline]`

The ErrorState() function returns the SLAB error state. An error state exists once an error occurs and lasts until the error is cleared with ErrorClear.

Returns :

true if SLAB is in an error state. false if SLAB is not in an error state

Remarks:

In client/server mode, the ErrorState() function checks error state locally; it does not communicate with the server. See Error Handling.

See also:

Error Functions

SLABError CSLABAPI::Error( ) `[inline]`

The Error() function returns the current SLAB error code. This error code state exists until the error is cleared with ErrorClear().

Returns :

The current SLAB error. Error returns ERR_SLAB_NONE if SLAB is not in an error state.

Remarks:

In client/server mode, the Error() function returns the local SLAB error; it does not communicate with the server. See Error Handling.

See also:

Error Functions

CScene Class Reference

List of all members.

Detailed Description

The CScene class allows SLAB Render plug-ins to access SLAB's scenario parameters and scene information. "Scenario" parameters are the parameters set using the SLAB Render API. "Scene" parameters are parameters computed from scenario parameters (e.g. the location of image reflections, listener-relative coordinates). A pointer to the CScene class is passed to the plug-in via CRPlugIn::Init().

The Update Functions manage the conversion from scenario to scene parameters.

The Query Functions are for querying scenario parameters. These parameters are either set using the SLAB Render API or are constants set internally by SLAB. The following parameters are currently fixed constants: sample rate, frame size, input delay line length, and speed of sound.

The Image Functions provide access to image-specific scenario and scene parameters. In use, the user renders each sound image by iterating through the list of all sound images.

See also:

CRPlugIn

Header: scene.h
Library: none

Public Methods

Update Functions

virtual bool Updated ()=0
checks if scenario parameters have been updated

virtual void Recalc ()=0
recalculates the scene

Query Functions

virtual int GetSrcNum ()=0
returns the number of sound sources

virtual double GetSrcRadius ()=0
returns the source radius of all sound sources

virtual double GetSrcSpread ()=0

returns the source spread of all sound sources

virtual double GetEnvSndSpd ()=0
returns the speed of sound

virtual MaterialType GetEnvMaterial (int n)=0
returns the material type of a surface

virtual Position * GetLstPosition ()=0
returns the listener position

virtual double GetSmoothTime ()=0
returns the DSP parameter smoothing filter time constant

virtual double GetSampleRate ()=0
returns the sampling rate

virtual int GetFrameSize ()=0
returns the frame size

virtual int GetDelayIn ()=0
returns the input delay line size

virtual short * GetOutL ()=0
returns the left-channel output buffer pointer

virtual short * GetOutR ()=0
returns the right-channel output buffer pointer

Image Functions

virtual void ResetImages ()=0
resets the sound image list

virtual void NextImage ()=0
advances to the next sound image

virtual void NextSource ()=0
advances to the next sound source

virtual bool GiEmpty ()=0
gets the image list empty flag

virtual float GiIn (float fIndex)=0
gets a sample from a sound image's input delay line

virtual SHORT GiIn (int nIndex)=0
gets a sample from a sound image's input delay line

virtual bool GiEnabled ()=0
gets the sound image enabled flag

virtual float GiGain ()=0

gets the scalar gain of the sound image

virtual double GiX ()=0
gets the x-coordinate of the sound image

virtual double GiY ()=0
gets the y-coordinate of the sound image

virtual double GiZ ()=0
gets the z-coordinate of the sound image

virtual double GiAz ()=0
gets the listener-relative azimuth of the sound image

virtual double GiEl ()=0
gets the listener-relative elevation of the sound image

virtual double GiRange ()=0
gets the image-listener range

virtual int GiFIRn ()=0
gets the number of FIR taps

virtual void GiHRTF (float fGain, float *pfHRIRL, float *pfHRIRR, float *pfITDL, float *pfITDR)=0
gets the HRIR and ITD for the image

virtual int GiRefID ()=0
gets the reflection ID of the image

virtual void * GiData ()=0
gets the plug-in image data pointer

virtual void SiData (void *pData)=0
sets the plug-in image data pointer

Member Function Documentation

virtual bool CScene::Updated( ) `[pure virtual]`

The Updated() function checks if scenario parameters have been updated. It is used with Recalc as follows:

```
if( m_pScene->Updated() )
{
  m_pScene->Recalc();
  if( !SceneToSigProc() )
    return false;
}
```

Where to place this code and how to use it is discussed further in the documentation for CRPlugIn::Process().

Returns :

true if scenario updated, otherwise false

See also:

Update Functions, Recalc(), CRPlugIn::Process()

virtual void CScene::Recalc( ) `[pure virtual]`

The Recalc() function recalculates the scene, computing new scene parameters from updated scenario parameters. It is used in conjunction with Updated(). See Updated() for a code fragment demonstrating the use of this function.

Returns :

none

See also:

Update Functions, Updated(), CRPlugIn::Process()

virtual int CScene::GetSrcNum( ) `[pure virtual]`

The GetSrcNum() function returns the number of sound sources.

Returns :

number of sound sources

See also:

Query Functions

virtual double CScene::GetSrcRadius( ) `[pure virtual]`

The GetSrcRadius() function returns the source radius of all sound sources.

Returns :

source radius of all sound sources (meters)

See also:

Query Functions, CSLABAPI::SrcRadius()

virtual double CScene::GetSrcSpread( ) `[pure virtual]`

The GetSrcSpread() function returns the source spread of all sound sources.

Returns :

source spread of all sound sources

See also:

Query Functions, CSLABAPI::SrcSpread()

virtual double CScene::GetEnvSndSpd( ) `[pure virtual]`

The GetEnvSndSpd() function returns the speed of sound.

Returns :

speed of sound (meters/s)

See also:

Query Functions

virtual MaterialType CScene::GetEnvMaterial(int *n*) `[pure virtual]`

The GetEnvMaterial() function returns the material type of a surface.

Returns :

surface material type

See also:

Query Functions, CSLABAPI::EnvMaterial()

Parameters:

n surface identifier (see Planes in SLABDEFS.h)

virtual Position* CScene::GetLstPosition( ) `[pure virtual]`

The GetEnvMaterial() function returns the listener position in a Position structure.

Returns :

listener position (meters)

See also:

Query Functions, CSLABAPI::LstPosition()

virtual double CScene::GetSmoothTime( ) `[pure virtual]`

The GetSmoothTime() function returns the DSP parameter smoothing filter time constant.

Returns :

DSP parameter smoothing filter time constant (milliseconds)

See also:

Query Functions, CSLABAPI::SmoothTime()

virtual double CScene::GetSampleRate( ) `[pure virtual]`

The GetSampleRate() function returns the sampling rate. The default sample rate is 44100 samples/s.

Returns :

sampling rate (samples/s)

See also:

Query Functions

virtual int CScene::GetFrameSize( ) `[pure virtual]`

The GetFrameSize() function returns the frame size. The default frame size is 32 samples.

Returns :

frame size (samples)

See also:

Query Functions

virtual int CScene::GetDelayIn( ) `[pure virtual]`

The GetDelayIn() function returns the input delay line size. The default input delay line size is 500ms.

Returns :

input delay line size (milliseconds)

See also:

Query Functions, GiIn()

virtual short* CScene::GetOutL( ) `[pure virtual]`

The GetOutL() function returns the left-channel output buffer pointer. This buffer receives the samples to be presented to the left ear of the listener.

Returns :

left-channel output buffer pointer

See also:

Query Functions

virtual short* CScene::GetOutR( ) `[pure virtual]`

The GetOutR() function returns the right-channel output buffer pointer. This buffer receives the samples to be presented to the right ear of the listener.

Returns :

right-channel output buffer pointer

See also:

Query Functions

virtual void CScene::ResetImages( ) `[pure virtual]`

The ResetImages() function resets the sound image list. This function must be called before iterating through the images with NextImage() or NextSource().

Returns :

none

See also:

Image Functions, NextImage(), NextSource()

virtual void CScene::NextImage( ) `[pure virtual]`

The NextImage() function advances to the next sound image (source or reflection) in the sound image list. Immediately after calling this function, call GiEmpty() to check if the image is valid. If it is, the Gi* and Si* functions can be used to access the scene parameters of the image. If the image is not valid, the end of the image list has been reached.

Returns :

none

See also:

Image Functions, ResetImages(), NextSource(), GiEmpty()

virtual void CScene::NextSource( ) `[pure virtual]`

The NextSource() function advances to the next sound source in the sound image list. Immediately after calling this function, call GiEmpty() to check if the source is valid. If it is, the Gi* and Si* functions can be used to access the scene parameters of the source. If the source is not valid, the end of the image list has been reached.

Returns :

none

See also:

Image Functions, ResetImages(), NextImage(), GiEmpty()

virtual bool CScene::GiEmpty( ) `[pure virtual]`

The GiEmpty() function checks if the image list is empty. This function should be called immediately after NextImage() or NextSource().

Returns :

true if the image list is empty, otherwise false

See also:

Image Functions, NextImage(), NextSource()

virtual float CScene::GiIn(float *fIndex*) `[pure virtual]`

The GiIn() function returns a sample from a sound image's input delay line. The samples at indices -GetFrameSize() to 0 correspond to the current frame of input samples at a pick-up point adjacent to the sound source. Using this frame of samples ignores sound propagation through the environment. To implement time-of-arrival effects such as doppler shift, ITD, and early reflections, you need to calculate the time-of-arrival and index into the delay line accordingly.

The float indexed version of this routine provides fractional indexing, the integer indexed version does not.

Warning:

You must ensure you do not overflow the input delay line (see GetDelayIn()).

Returns :

float input sample

See also:

Image Functions, GiIn(int nIndex), GetDelayIn()

Parameters:

fIndex fractional index of input sample

virtual SHORT CScene::GiIn(int *nIndex*) `[pure virtual]`

The GiIn() function returns a sample from a sound image's input delay line. For details, see the float indexed version of this routine.

For implementations that do not require fractional indexing of the delay line, this routine is slightly faster.

Warning:

You must ensure you do not overflow the input delay line (see GetDelayIn()).

Returns :

SHORT input sample

See also:

Image Functions, GiIn(float fIndex), GetDelayIn()

Parameters:

nIndex index of input sample

virtual bool CScene::GiEnabled( ) `[pure virtual]`

The GiEnabled() function checks if a sound image is enabled.

Returns :

true if sound image enabled, otherwise false

See also:

Image Functions, CSLABAPI::SrcEnable(), CSLABAPI::SrcRefEnable()

virtual float CScene::GiGain( ) `[pure virtual]`

The GiGain() function returns the scalar gain of the sound image. This gain corresponds to the dB-specified gain set using the SLAB Render API. If the image is a reflection, the gain of the corresponding sound source is returned.

Returns :

scalar gain of the sound image

See also:

Image Functions, CSLABAPI::SrcGain()

virtual double CScene::GiX( ) `[pure virtual]`

The GiX() function returns the x-coordinate of the sound image. For reflections, this value is computed using an image model.

Returns :

x-coordinate of the sound image (meters)

See also:

Image Functions, CSLABAPI::SrcLocate()

virtual double CScene::GiY( ) `[pure virtual]`

The GiY() function returns the y-coordinate of the sound image. For reflections, this value is computed using an image model.

Returns :

y-coordinate of the sound image (meters)

See also:

Image Functions, CSLABAPI::SrcLocate()

virtual double CScene::GiZ( ) `[pure virtual]`

The GiZ() function returns the z-coordinate of the sound image. For reflections, this value is computed using an image model.

Returns :

z-coordinate of the sound image (meters)

See also:

Image Functions, CSLABAPI::SrcLocate()

virtual double CScene::GiAz( ) `[pure virtual]`

The GiAz() function returns the listener-relative azimuth of the sound image.

Returns :

listener-relative azimuth of the sound image (radians)

See also:

Image Functions

virtual double CScene::GiEl( ) `[pure virtual]`

The GiEl() function returns the listener-relative elevation of the sound image.

Returns :

listener-relative elevation of the sound image (radians)

See also:

Image Functions

virtual double CScene::GiRange( ) `[pure virtual]`

The GiRange() function returns the image-listener range.

Returns :

image-listener range (meters)

See also:

Image Functions

virtual int CScene::GiFIRn( ) `[pure virtual]`

The GiFIRn() function returns the SRAPI-specified number of FIR taps for the HRIR filter operation. This can be ignored or used for another purpose by renderers that do not perform an HRIR FIR operation. The FIRn value for reflections is 1/4 the FIRn for the direct path. This reserves more CPU resources for the direct path, causing the direct path to be rendered at a higher fidelity than the reflections.

Returns :

number of FIR taps

See also:

Image Functions, CSLABAPI::FIRPoints()

```
virtual void CScene::GiHRTF( float    fGain,
                             float *  pfHRIRL,
                             float *  pfHRIRR,
                             float *  pfITDL,
                             float *  pfITDR
                           ) [pure virtual]
```

The GiHRTF() function gets the HRIR and ITD for the image. Internally, the GiAz() and GiEl() values are used to index into the HRTF database. The number of HRIR taps retrieved is specified by GiFIRn(). If the az,el index is not found in the HRTF database, linear interpolation is used to generate the HRIR pair and ITD from the four nearest neighbors.

Returns :

none

See also:

Image Functions, GiAz(), GiEl(), GiFIRn()

Parameters:

fGain	scalar gain applied to HRIR taps
pfHRIRL	left HRIR
pfHRIRR	right HRIR
pfITDL	left ITD (lag or zero)
pfITDR	right ITD (lag or zero)

virtual int CScene::GiRefID( ) `[pure virtual]`

The GiRefID() function returns the reflection ID of the image. This will either be -1 for the direct path or one of the Plane enum values in SLABDEFS.h.

Returns :

reflection ID

See also:

Image Functions, Planes

virtual void* CScene::GiData( ) `[pure virtual]`

The GiData() function returns the plug-in image data pointer. See SiData() for a discussion of plug-in image data.

Returns :

plug-in image data pointer

See also:

Image Functions, SiData()

virtual void CScene::SiData(void * *pData*) `[pure virtual]`

The SiData() function sets the plug-in image data pointer. Image data is data allocated by the plug-in and attached to the image via SiData(). This data is typically allocated in the plug-in override of CRPlugIn::Begin() and freed in the override of CRPlugIn::End(). Examples of image data are FIR and IIR memory data and tracked DSP parameters (e.g. delay line indices, FIR taps).

Returns :

none

See also:

Image Functions, GiData()

Parameters:

pData address of plug-in image data

SRAPI Reference Manual generated Thu Dec 19 14:06:11 2002 by doxygen 1.2.17

Stats Struct Reference

List of all members.

Detailed Description

The Stats structure provides statistics (i.e. counters) for various SLAB events (e.g. DirectSound underflow).

Public Attributes

unsigned long lClip
unsigned long lUnderflow

Member Data Documentation

unsigned long Stats::lClip

number of Render plug-in mixer clips since RenderStart()

unsigned long Stats::lUnderflow

number of DirectSound underflows since RenderStart()

CSLABClient Class Reference

Inheritance diagram for CSLABClient:

List of all members.

Detailed Description

The CSLABClient class provides client/server-mode access to a SLAB server. In client/server-mode, the SLAB renderer is executed remotely on a SLAB server workstation. Thus, all SLAB processing and audio output is isolated from the user's computer. The benefits of client/server-mode are better robustness, higher performance, and lower audio latency. The cost, however, is increased hardware requirements. A second workstation must be allocated as a SLAB server and a local network installed.

CSLABClient objects are allocated with the SLABAPIClient support function declared in SLABAPIClient.h.

Note:

The user interacts with the CSLABClient object through the CSLABAPI class interface.

See also:

CSLABAPI
SLABAPIClient.h

Header: SLABAPI.h
Library: slabc

Public Methods

Client Functions
void MessageBox (bool bEnable)
displays a message box if an error occurs

Member Function Documentation

void CSLABClient::MessageBox(bool *bEnable*) `[inline, virtual]`

The MessageBox() function allows the user to request a SLAB error message box. If a SLAB error occurs, a message box pops up displaying a SLAB error string. This occurs in addition to the standard error reporting mechanism.

Returns :

none

Remarks:

This function does not communicate with the SLAB server.

See also:

Client Functions

Parameters:

bEnable enable error message box flag

Reimplemented from CSLABAPI.

SRAPI Reference Manual generated Thu Dec 19 14:06:11 2002 by doxygen 1.2.17

Desc Struct Reference

List of all members.

Detailed Description

The Desc structure provides a description of the plug-in.

Public Attributes

short nPIVersion
CRPlugIn version.

char * strName
plug-in name

short nVerMajor
plug-in version - major

short nVerMinor
plug-in version - minor

short nVerRelease
plug-in version - release

char * strAuthor
plug-in author

short nDateMonth
plug-in date - month

short nDateDay
plug-in date - day

short nDateYear
plug-in date - year

char * strDesc
plug-in text description

Position Struct Reference

List of all members.

Detailed Description

The Position structure contains elements for defining the location and orientation of an object.

Public Attributes

double x
x location

double y
y location

double z
z location

double yaw
yaw orientation

double pitch
pitch orientation

double roll
roll orientation

SLABDEFS.h File Reference

Detailed Description

SLABDEFS.h contains defines and types for writing SLAB-based applications.

Author:

Joel D. Miller

Date:

June 6, 2000

Compounds

struct Position
position (location and orientation) structure More...

struct Polar
polar location structure More...

struct Stats
SLAB statistics structure. More...

Defines

#define ERROR_STRINGS
SLABError error strings (see SLABDEFS.h).

#define SLAB_FS 44100
sampling rate (int)

#define dSLAB_FS 44100.0
sampling rate (double)

#define dSoundSpeed 346.0
speed of sound (meters/s)

#define dGainMutedB -120.0
mute gain (dB)

#define dPI 3.14159265358979323846

pi (double)

#define dPId2 (dPI / 2.0)
pi/2 (double)

#define dPIx2 (dPI * 2.0)
*2*pi (double)*

#define fPI float(dPI)
pi (float)

#define fPId2 float(dPId2)
pi/2 (float)

#define fPIx2 float(dPIx2)
*2*pi (float)*

#define dDeg2Rad(x) (double(x) * dPI / 180.0)
degrees-to-radians (double)

#define dRad2Deg(x) (double(x) * 180.0 / dPI)
radians-to-degrees (double)

#define fDeg2Rad(x) (float(x) * fPI / 180.0f)
degrees-to-radians (float)

#define fRad2Deg(x) (float(x) * 180.0f / fPI)
radians-to-degrees (float)

#define dM2In(x) (double(x) / 0.0254)
meters-to-inches (double)

#define dIn2M(x) (double(x) * 0.0254)
inches-to-meters (double)

#define fM2In(x) (float(x) / 0.0254f)
meters-to-inches (float)

#define fIn2M(x) (float(x) * 0.0254f)
inches-to-meters (float)

#define dINTERAURALin 7.0
width of head along interaural axis (inches) (double)

#define dINTERAURALm dIn2M(dINTERAURALin)
width of head along interaural axis (meters) (double)

#define fINTERAURALin float(dINTERAURALin)
width of head along interaural axis (inches) (float)

#define fINTERAURALm float(dINTERAURALm)
width of head along interaural axis (meters) (float)

#define dSENSOROFFSETin 7.5

tracker sensor offset relative to interaural axis (inches) (double)

#define dSENSOROFFSETm dIn2M(dSENSOROFFSETin)
tracker sensor offset relative to interaural axis (meters) (double)

#define fSENSOROFFSETin float(dSENSOROFFSETin)
tracker sensor offset relative to interaural axis (inches) (float)

#define fSENSOROFFSETm float(dSENSOROFFSETm)
tracker sensor offset relative to interaural axis (meters) (float)

#define RENDER_MAX 100
maximum number of Render plug-ins

#define MATERIAL_STRINGS
MaterialType strings.

Enumerations

enum SLABError
SLABError errors (see ERROR_STRINGS in SLABDEFS.h for error descriptions).

enum IDRender {
RENDER_SPATIAL, RENDER_MONOTIC_L, RENDER_MONOTIC_R, RENDER_DIOTIC,
RENDER_DIOTIC_V, RENDER_PLUGIN
}
Render IDs. More...

enum IDSS {
SS_SrcLocate, SS_SrcLocatePolar, SS_SrcGain, SS_SrcEnable,
SS_SrcRefEnable, SS_SrcTrajLStart, SS_SrcTrajLMod, SS_SrcTrajPStart,
SS_SrcTrajStart, SS_SrcTrajStop, SS_LstPosition, SS_EnvPlanes,
SS_Tail
}
SLABScript commands. More...

enum PositionIndex {
POS_X, POS_Y, POS_Z, POS_YAW,
POS_PCH, POS_ROL
}
position data array indices More...

enum Planes {
RVB_XP, RVB_XN, RVB_YP, RVB_YN,
RVB_ZP, RVB_ZN, RVB_PLANES, RVB_PATHS,
RVB_ALL
}

reflection identifiers More...

enum MaterialType {
MAT_CARPET, MAT_CONCRETE, MAT_GLASS, MAT_GYPSUM,
MAT_WOODAIR, MAT_PLASTER, MAT_NONE, MAT_NUM
}
wall material types More...

Define Documentation

#define MATERIAL_STRINGS

Value:

```
{ "Heavy Carpet", "Concrete", "Heavy Glass", "Gypsum Board", \
  "Wood With Airspace", "Plaster On Metal", "Perfect Reflector" }
```

The MATERIAL_STRINGS define provides strings for the MaterialType enum values.

Enumeration Type Documentation

enum IDRender

The spatial, monotic, and diotic Render IDs correspond to the default Render plug-ins included in the SLAB User Release. User plug-ins begin at *RENDER_PLUGIN* (see CSLABAPI::RenderID()).

See also:

CSLABAPI Render Functions

Enumeration values:

RENDER_SPATIAL	spatial display
RENDER_MONOTIC_L	left-monotic display
RENDER_MONOTIC_R	right-monotic display
RENDER_DIOTIC	diotic display
RENDER_DIOTIC_V	mixed monotic and diotic display depending on source location
RENDER_PLUGIN	beginning of user Render plug-in IDs

enum IDSS

The IDSS enum contains SLABScript commands. These are used with CSLABAPI::ScriptAdd() to create SLAB scripts.

See also:

CSLABAPI Scripting Functions

Enumeration values:

SS_SrcLocate	SrcLocate().
SS_SrcLocatePolar	SrcLocatePolar().
SS_SrcGain	SrcGain().
SS_SrcEnable	SrcEnable().
SS_SrcRefEnable	SrcRefEnable().
SS_SrcTrajLStart	SrcTrajLStart().
SS_SrcTrajLMod	SrcTrajLMod().
SS_SrcTrajPStart	SrcTrajPStart().
SS_SrcTrajStart	SrcTrajStart().
SS_SrcTrajStop	SrcTrajStop().
SS_LstPosition	LstPosition().
SS_EnvPlanes	EnvPlanes().
SS_Tail	for internal use only!

enum PositionIndex

The PositionIndex enum provides array indices for position data arrays.

Enumeration values:

POS_X	x location
POS_Y	y location
POS_Z	z location
POS_YAW	yaw orientation
POS_PCH	pitch orientation
POS_ROL	roll orientation

enum Planes

The first six Planes enum values identify the first-order reflections in a six wall rectangular room. The suffix indicates the Positive or Negative axis to which the wall is orthogonal.

Remarks:

Currently, SLAB is limited to a first-order early reverb model. Late reverberation is not modeled.

Enumeration values:

RVB_XP	positive x plane
RVB_XN	negative x plane
RVB_YP	positive y plane
RVB_YN	negative y plane
RVB_ZP	positive z plane
RVB_ZN	negative z plane
RVB_PLANES	number of planes

RVB_PATHS	number of source-to-listener paths: direct + reflections
RVB_ALL	all reflections (RVB_XP thru RVB_ZN) flag

enum MaterialType

The MaterialType enum defines a set of wall material types for use with CSLABAPI::EnvMaterial().

Enumeration values:

MAT_CARPET	heavy carpet / foam rubber
MAT_CONCRETE	concrete
MAT_GLASS	heavy glass
MAT_GYPSUM	gypsum board, 2 layer
MAT_WOODAIR	1" thick wood with airspace
MAT_PLASTER	plaster on metal lath
MAT_NONE	no material processing
MAT_NUM	number of defined materials

---

rplugin.h File Reference

Detailed Description

rplugin.h contains the class interface for CRPlugIn, the base class for writing SLAB Render plug-ins.

Author:

Joel D. Miller

Date:

August 13, 2002

Compounds

struct Desc
plug-in description struct More...

class CRPlugIn
SLAB Render plug-in base class. More...

Defines

#define DESC_VERSION 1
plug-in description struct version number

#define PIESL 512
plug-in error string length

SLABAPIClient.h File Reference

Detailed Description

SLABAPIClient.h provides routines for CSLABClient allocation and allocation error checking.

Header: SLABAPIClient.h
Library: slabc
Author:

Joel D. Miller

Date:

March 8, 2000

SLABAPIClient Functions

CSLABAPI * SLABAPIClient (char *IP)
allocates a CSLABClient object

SLABError SLABAPIClientError ()
returns the SLAB allocation error code

char * SLABAPIClientErrorString ()
returns the SLAB allocation error string

char * SLABAPIClientErrorStack ()
returns the SLAB allocation error strack

Function Documentation

CSLABAPI* SLABAPIClient(char * *IP*)

The SLABAPIClient() function allocates a CSLABClient object and establishes a socket connection to a SLAB Server.

Returns :

CSLABClient object pointer if successful, NULL if an error occurred

Further error information can be obtained using SLABAPIClientError().

Note:

The CSLABClient object is referenced through the CSLABAPI interface.

See also:

SLABAPIClient Functions

Parameters:

IP IP address of a SLAB server; the IP address is passed as a string in standard IP format (e.g. "10.0.0.1")

SLABError SLABAPIClientError( )

The SLABAPIClientError() function returns the SLAB allocation error code.

Returns :

SLAB allocation error status. Returns ERR_SLAB_NONE if SLAB allocated without an error.

See also:

SLABAPIClient Functions

char* SLABAPIClientErrorString( )

The SLABAPIClientErrorString() function returns a string describing the SLAB allocation error condition.

Returns :

string describing the SLAB allocation error condition

See also:

SLABAPIClient Functions
ERROR_STRINGS in SLABDEFS.h

char* SLABAPIClientErrorStack( )

The SLABAPIClientErrorStack() function is used to obtain detailed error information after an error occurs. It returns a string describing the error state at each point in the internal function calling sequence at the time of the error. This function is primarily for development and bug reporting.

Returns :

string listing the error state at each point in the internal function calling sequence at the time of the allocation error

See also:

SLABAPIClient Functions

SRAPI Reference Manual generated Thu Dec 19 14:06:11 2002 by doxygen 1.2.17

SLABAPIHost.h File Reference

Detailed Description

SLABAPIHost.h provides routines for CSLABHost allocation and allocation error checking.

Header: SLABAPIHost.h
Library: slabh
Author:

Joel D. Miller

Date:

June 22, 2000

SLABAPIHost Functions

CSLABAPI * SLABAPIHost ()
allocates a CSLABHost object

SLABError SLABAPIHostError ()
returns the SLAB allocation error code

char * SLABAPIHostErrorString ()
returns the SLAB allocation error string

char * SLABAPIHostErrorStack ()
returns the SLAB allocation error strack

Function Documentation

CSLABAPI* SLABAPIHost( )

The SLABAPIHost() function allocates a CSLABHost object.

Returns :

CSLABHost object pointer if successful, NULL if an error occurred

Further error information can be obtained using SLABAPIHostError().

Note:

The CSLABHost object is referenced through the CSLABAPI interface.

See also:

SLABAPIHost Functions

SLABError SLABAPIHostError( )

The SLABAPIHostError() function returns the SLAB allocation error code.

Returns :

SLAB allocation error status. Returns ERR_SLAB_NONE if SLAB allocated without an error.

See also:

SLABAPIHost Functions

char* SLABAPIHostErrorString( )

The SLABAPIHostErrorString() function returns a string describing the SLAB allocation error condition.

Returns :

string describing the SLAB allocation error condition

See also:

SLABAPIHost Functions
ERROR_STRINGS in SLABDEFS.h

char* SLABAPIHostErrorStack( )

The SLABAPIHostErrorStack() function is used to obtain detailed error information after an error occurs. It returns a string describing the error state at each point in the internal function calling sequence at the time of the error. This function is primarily for development and bug reporting.

Returns :

string listing the error state at each point in the internal function calling sequence at the time of the allocation error

See also:

SLABAPIHost Functions

Polar Member List

This is the complete list of members for Polar, including all inherited members.

az	Polar
el	Polar
r	Polar

CRPlugIn Member List

This is the complete list of members for CRPlugIn, including all inherited members.

Begin(void)	CRPlugIn	[inline, virtual]
End(void)	CRPlugIn	[inline, virtual]
ErrorClear()	CRPlugIn	[inline]
ErrorSet(char *strError)	CRPlugIn	[inline, protected]
ErrorState()	CRPlugIn	[inline]
ErrorString()	CRPlugIn	[inline]
GetClips(void)	CRPlugIn	[inline]
Init(CScene *pScene, void **ppDesc)=0	CRPlugIn	[pure virtual]
m_bError	CRPlugIn	[private]
m_lClip	CRPlugIn	[protected]
m_pScene	CRPlugIn	[protected]
m_strError	CRPlugIn	[private]
Process(void)=0	CRPlugIn	[pure virtual]

CSLABHost Member List

This is the complete list of members for CSLABHost, including all inherited members.

Callback(void(*pCallback)(void *pPrm), void *pParam, int nPeriod)	CSLABHost	[inline, virtual]
Close()	CSLABAPI	[inline, virtual]
Error()	CSLABAPI	[inline]
ErrorState()	CSLABAPI	[inline]
GetDefHRTF(char *pStr, int nSize)	CSLABAPI	[static]
GetHRTFDir(char *pStr, int nSize)	CSLABAPI	[static]
GetInstDir(char *pStr, int nSize)	CSLABAPI	[static]
GetLogDir(char *pStr, int nSize)	CSLABAPI	[static]
GetLstPosition()	CSLABHost	[inline, virtual]
GetSrcGain(IDSRC idSrc)	CSLABHost	[inline, virtual]
GetSrcID(int nSrc)	CSLABHost	[inline, virtual]
GetSrcLocation(IDSRC idSrc)	CSLABHost	[inline, virtual]
GetSrcNum()	CSLABHost	[inline, virtual]
GetSrcRel(IDSRC idSrc)	CSLABHost	[inline, virtual]
GetVersion(char *pStr, int nSize)	CSLABAPI	[static]
LogEntry(char *pLogEntry)	CSLABHost	[inline, virtual]
LogName(char *pLogName)	CSLABHost	[virtual]
LogTime()	CSLABHost	[inline, virtual]
MessageBox(bool bEnable)	CSLABAPI	[inline, virtual]
Open(void(*pFuncLog)(char *strLogEntry))	CSLABAPI	[inline, virtual]
PCount()	CSLABAPI	[inline, virtual]
PCountReset()	CSLABAPI	[inline, virtual]
SetDefHRTF(const char *pStr)	CSLABAPI	[static]

SetHRTFDir(const char *pStr)	CSLABAPI	[static]

SRAPI Reference Manual generated Thu Dec 19 14:06:11 2002 by doxygen 1.2.17

SLABAPI.h File Reference

Detailed Description

SLABAPI.h is the primary include file necessary when writing a SLAB-based application.

Author:

Joel D. Miller

Date:

May 19, 1999

Compounds

class CSLABAPI

The CSLABAPI class is the user's primary interface to SLAB. More...

CSLABAPI Member List

This is the complete list of members for CSLABAPI, including all inherited members.

Callback(void(*pCallback)(void *pPrm), void *pParam, int nPeriod)	CSLABAPI	[inline, virtual]
Close()	CSLABAPI	[inline, virtual]
EnvMaterial(int nPlane, MaterialType matType)=0	CSLABAPI	[pure virtual]
EnvPlanes(double xp, double xn, double yp, double yn, double zp, double zn)=0	CSLABAPI	[pure virtual]
Error()	CSLABAPI	[inline]
ErrorClear()	CSLABAPI	[inline, virtual]
ErrorStack()=0	CSLABAPI	[pure virtual]
ErrorState()	CSLABAPI	[inline]
ErrorString()=0	CSLABAPI	[pure virtual]
FIRPoints(int nFIRPoints)=0	CSLABAPI	[pure virtual]
GetDefHRTF(char *pStr, int nSize)	CSLABAPI	[static]
GetHRTFDir(char *pStr, int nSize)	CSLABAPI	[static]
GetInstDir(char *pStr, int nSize)	CSLABAPI	[static]
GetLogDir(char *pStr, int nSize)	CSLABAPI	[static]
GetLstPosition()	CSLABAPI	[inline, virtual]
GetSrcGain(IDSRC idSrc)	CSLABAPI	[inline, virtual]
GetSrcID(int nSrc)	CSLABAPI	[inline, virtual]
GetSrcLocation(IDSRC idSrc)	CSLABAPI	[inline, virtual]
GetSrcNum()	CSLABAPI	[inline, virtual]
GetSrcRel(IDSRC idSrc)	CSLABAPI	[inline, virtual]
GetStats()=0	CSLABAPI	[pure virtual]
GetVersion(char *pStr, int nSize)	CSLABAPI	[static]
LogEntry(char *pLogEntry)	CSLABAPI	[inline, virtual]
LogName(char *pLogName)	CSLABAPI	[inline, virtual]

LogTime()	CSLABAPI	[inline, virtual]
LstHRTF(IDHRTF idHRTF, IDSRC idSrc=NULL)=0	CSLABAPI	[pure virtual]
LstHRTFFree(IDHRTF idHRTF)=0	CSLABAPI	[pure virtual]
LstHRTFLoad(const char *pName)=0	CSLABAPI	[pure virtual]
LstPosition(double x, double y, double z, double yaw, double pitch, double roll)=0	CSLABAPI	[pure virtual]
LstSensorOffset(double x, double y, double z)=0	CSLABAPI	[pure virtual]
MessageBox(bool bEnable)	CSLABAPI	[inline, virtual]
Open(void(*pFuncLog)(char *strLogEntry))	CSLABAPI	[inline, virtual]
OutDS(int nOut, int nWrite)=0	CSLABAPI	[pure virtual]
OutMemory(char *pFilename)=0	CSLABAPI	[pure virtual]
PCount()	CSLABAPI	[inline, virtual]
PCountReset()	CSLABAPI	[inline, virtual]
RenderChange(int nID)=0	CSLABAPI	[pure virtual]
RenderEnum()=0	CSLABAPI	[pure virtual]
RenderID(char *pName)=0	CSLABAPI	[pure virtual]
RenderStart(int nID)=0	CSLABAPI	[pure virtual]
RenderStop()=0	CSLABAPI	[pure virtual]
Reset()=0	CSLABAPI	[pure virtual]
ScriptAdd(IDSS idSS, double dTime,...)=0	CSLABAPI	[pure virtual]
ScriptBegin(double dUpdateRate)=0	CSLABAPI	[pure virtual]
ScriptRead(char *pFilename, IDSRC *pIDSrc, int nLen)=0	CSLABAPI	[pure virtual]
ScriptWrite(char *pFilename, IDSRC *pIDSrc, int nLen)=0	CSLABAPI	[pure virtual]
SetDefHRTF(const char *pStr)	CSLABAPI	[static]
SetHRTFDir(const char *pStr)	CSLABAPI	[static]
SetNotify(CWnd *pWnd, WPARAM wParam)=0	CSLABAPI	[pure virtual]
SmoothTime(double dSmoothTime)=0	CSLABAPI	[pure virtual]
SrcEnable(IDSRC idSrc, bool bEnable)=0	CSLABAPI	[pure virtual]
SrcFile(char *pFilename, bool bLoop=true)=0	CSLABAPI	[pure virtual]
SrcFree()=0	CSLABAPI	[pure virtual]
SrcGain(IDSRC idSrc, double dDB)=0	CSLABAPI	[pure virtual]
SrcGenImpulse(int nPeriod, int nAmp)=0	CSLABAPI	[pure virtual]
SrcGenNoise(int nAmp)=0	CSLABAPI	[pure virtual]
SrcGenSine(int nFreq, int nAmp)=0	CSLABAPI	[pure virtual]
SrcGenSquare(int nFreq, int nAmp)=0	CSLABAPI	[pure virtual]

SrcLocate(IDSRC idSrc, double x, double y, double z)=0	CSLABAPI	[pure virtual]
SrcLocatePolar(IDSRC idSrc, double az, double el, double r)=0	CSLABAPI	[pure virtual]
SrcRadius(double dSrcRadius)=0	CSLABAPI	[pure virtual]
SrcRefEnable(IDSRC idSrc, int nRef, bool bEnable)=0	CSLABAPI	[pure virtual]
SrcRefOffset(IDSRC idSrc, int nRef, bool bRefOffset, double x=0.0, double y=0.0, double z=0.0)=0	CSLABAPI	[pure virtual]
SrcSpread(double dSrcSpread)=0	CSLABAPI	[pure virtual]
SrcTrajEnable(bool bEnable)=0	CSLABAPI	[pure virtual]
SrcTrajLMod(IDSRC idSrc, double dRateX, double dRateY, double dRateZ)=0	CSLABAPI	[pure virtual]
SrcTrajLStart(IDSRC idSrc, double dX, double dY, double dZ, double dRateX, double dRateY, double dRateZ)=0	CSLABAPI	[pure virtual]
SrcTrajPMod(IDSRC idSrc, double dRateAz, double dRateEl, double dRateR)=0	CSLABAPI	[pure virtual]
SrcTrajPStart(IDSRC idSrc, double dAz, double dEl, double dR, double dRateAz, double dRateEl, double dRateR)=0	CSLABAPI	[pure virtual]
SrcTrajRate(double dUpdateRate)=0	CSLABAPI	[pure virtual]
SrcTrajStart(IDSRC idSrc)=0	CSLABAPI	[pure virtual]
SrcTrajStop(IDSRC idSrc)=0	CSLABAPI	[pure virtual]

CScene Member List

This is the complete list of members for CScene, including all inherited members.

GetDelayIn()=0	CScene	[pure virtual]
GetEnvMaterial(int n)=0	CScene	[pure virtual]
GetEnvSndSpd()=0	CScene	[pure virtual]
GetFrameSize()=0	CScene	[pure virtual]
GetLstPosition()=0	CScene	[pure virtual]
GetOutL()=0	CScene	[pure virtual]
GetOutR()=0	CScene	[pure virtual]
GetSampleRate()=0	CScene	[pure virtual]
GetSmoothTime()=0	CScene	[pure virtual]
GetSrcNum()=0	CScene	[pure virtual]
GetSrcRadius()=0	CScene	[pure virtual]
GetSrcSpread()=0	CScene	[pure virtual]
GiAz()=0	CScene	[pure virtual]
GiData()=0	CScene	[pure virtual]
GiEl()=0	CScene	[pure virtual]
GiEmpty()=0	CScene	[pure virtual]
GiEnabled()=0	CScene	[pure virtual]
GiFIRn()=0	CScene	[pure virtual]
GiGain()=0	CScene	[pure virtual]
GiHRTF(float fGain, float *pfHRIRL, float *pfHRIRR, float *pfITDL, float *pfITDR)=0	CScene	[pure virtual]
GiIn(float fIndex)=0	CScene	[pure virtual]
GiIn(int nIndex)=0	CScene	[pure virtual]
GiRange()=0	CScene	[pure virtual]
GiRefID()=0	CScene	[pure virtual]
GiX()=0	CScene	[pure virtual]
GiY()=0	CScene	[pure virtual]
GiZ()=0	CScene	[pure virtual]
NextImage()=0	CScene	[pure virtual]
NextSource()=0	CScene	[pure virtual]
Recalc()=0	CScene	[pure virtual]
ResetImages()=0	CScene	[pure virtual]

SiData(void *pData)=0	CScene	[pure virtual]
Updated()=0	CScene	[pure virtual]

SRAPI Reference Manual generated Thu Dec 19 14:06:11 2002 by doxygen 1.2.17

scene.h File Reference

Detailed Description

scene.h contains the class interface for CScene, the class providing access to the scenario and scene information when writing SLAB Render plug-ins.

Author:

Joel D. Miller

Date:

August 8, 2002

Compounds

class CScene

SLAB scenario and scene parameter interface. More...

Stats Member List

This is the complete list of members for Stats, including all inherited members.

lClip	Stats
lUnderflow	Stats

CSLABClient Member List

This is the complete list of members for CSLABClient, including all inherited members.

Callback(void(*pCallback)(void *pPrm), void *pParam, int nPeriod)	CSLABAPI	[inline, virtual]
Close()	CSLABAPI	[inline, virtual]
Error()	CSLABAPI	[inline]
ErrorState()	CSLABAPI	[inline]
GetDefHRTF(char *pStr, int nSize)	CSLABAPI	[static]
GetHRTFDir(char *pStr, int nSize)	CSLABAPI	[static]
GetInstDir(char *pStr, int nSize)	CSLABAPI	[static]
GetLogDir(char *pStr, int nSize)	CSLABAPI	[static]
GetLstPosition()	CSLABAPI	[inline, virtual]
GetSrcGain(IDSRC idSrc)	CSLABAPI	[inline, virtual]
GetSrcID(int nSrc)	CSLABAPI	[inline, virtual]
GetSrcLocation(IDSRC idSrc)	CSLABAPI	[inline, virtual]
GetSrcNum()	CSLABAPI	[inline, virtual]
GetSrcRel(IDSRC idSrc)	CSLABAPI	[inline, virtual]
GetVersion(char *pStr, int nSize)	CSLABAPI	[static]
LogEntry(char *pLogEntry)	CSLABAPI	[inline, virtual]
LogName(char *pLogName)	CSLABAPI	[inline, virtual]
LogTime()	CSLABAPI	[inline, virtual]
MessageBox(bool bEnable)	CSLABClient	[inline, virtual]
Open(void(*pFuncLog)(char *strLogEntry))	CSLABAPI	[inline, virtual]
PCount()	CSLABAPI	[inline, virtual]
PCountReset()	CSLABAPI	[inline, virtual]

SetDefHRTF(const char *pStr)	CSLABAPI	[static]
SetHRTFDir(const char *pStr)	CSLABAPI	[static]

SRAPI Reference Manual generated Thu Dec 19 14:06:11 2002 by doxygen 1.2.17

Desc Member List

This is the complete list of members for Desc, including all inherited members.

nDateDay	Desc
nDateMonth	Desc
nDateYear	Desc
nPIVersion	Desc
nVerMajor	Desc
nVerMinor	Desc
nVerRelease	Desc
strAuthor	Desc
strDesc	Desc
strName	Desc

Position Member List

This is the complete list of members for Position, including all inherited members.

pitch	Position
roll	Position
x	Position
y	Position
yaw	Position
z	Position

SLABClient.h File Reference

Detailed Description

SLABClient.h contains the class interface for CSLABClient, the class used to communicate with a SLAB server. Note, though, users access this class through the CSLABAPI class interface. This enables the selection of local or remote SLAB use at run-time depending on how the CSLABAPI-derived class is allocated. It also conceals the implementation details of the host and client interfaces.

Note:

This file is not included in the SLAB User Release.

Author:

Joel D. Miller

Date:

April 21, 1999

Compounds

class CSLABClient

The CSLABClient class provides a client interface to SLAB. More...

SLABHost.h File Reference

Detailed Description

SLABHost.h contains the class interface for CSLABHost, the class used to execute SLAB locally. Note, though, users access this class through the CSLABAPI class interface. This enables the selection of local or remote SLAB use at run-time depending on how the CSLABAPI-derived class is allocated. It also conceals the implementation details of the host and client interfaces.

Note:

This file is not included in the SLAB User Release.

Author:

Joel D. Miller

Date:

April 19, 1999

Compounds

class CSLABHost

The CSLABHost class provides a host interface to SLAB. More...