MediaPlug-inlibrary

Objective systems
Version Version 0.1
8/9/2004 2:12 PM

Table of Contents

MediaPlug-inlibrary Module Index

MediaPlug-inlibrary Modules

Here is a list of all modules:

MediaPlug-inlibrary Data Structure Index

MediaPlug-inlibrary Data Structures

Here are the data structures with brief descriptions:

MediaPlug-inlibrary File Index

MediaPlug-inlibrary File List

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

MediaPlug-inlibrary Module Documentation

Media API Internal Common Helper Functions

Media API Internal Common Helper Functions

Defines

- #define **OORTPPACKETDATASIZE** 240 /* Send Receive Packet Data Size*/
- #define **MAXLOGMSGLEN** 1024
- #define **OOLOG2**(a, b) ooLog(a,b)
- #define **OOLOG3**(a, b, c) ooLog(a,b,c)
- #define **OOLOG4**(a, b, c, d) ooLog(a,b,c,d)
- #define **OOLOG5**(a, b, c, d, e) ooLog(a,b,c,d,e)
- #define **OOLOG9**(a, b, c, d, e, f, g, h, i) ooLog(a,b,c,d,e,f,g,h,i)

Functions

- EXTERN void **ooLog** (int level, const char *fmtspec,...)
  This function logs a trace message into a log file.

- EXTERN void **ooSleep** (int milliseconds)
  Platform independent sleep function.

Variables

- FILE * **fpLog**

Function Documentation

EXTERN void ooLog (int *level*, const char * *fmtspec*, ...)

This function logs a trace message into a log file.

Parameters:

level Log level(Currently not used)
fmtspec Format specification for the log message.
... Variable number of arguments representing the message.

EXTERN void ooSleep (int *milliseconds*)

Platform independent sleep function.

Parameters:

milliseconds Sleep time in milliseconds.

Media API Internal Linux Helper Functions

Media API Internal Linux Helper Functions

Functions

- EXTERN int **ooOpenWaveFileForRead** (char *filename)
  Opens a RAW audio data file for read.

- EXTERN int **ooReadWaveFileData** (char *databuf, int size)
  Reads data from the opened raw audio file.

- EXTERN int **ooCloseWaveFile** ()
  Close the open raw audio data file.

- EXTERN int **ooOpenAudioDevice** ()
  Opens the audio device for read/write operation.

- EXTERN int **ooPlayAudioBuffer** (unsigned char *buff, long size)
  Plays a buffer full of audio data onto the audio device.

- EXTERN int **ooGetMicAudioBuffer** (unsigned char *buff, long size)
  Reads audio data from the microphone device.

Variables

- int **ghSoundDevice**
  Global handle to open sound device.

- int **ghSndFile**
  Global handle to open raw audio data file.

Function Documentation

EXTERN int ooCloseWaveFile ()

Close the open raw audio data file.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooGetMicAudioBuffer (unsigned char * *buff*, long *size*)

Reads audio data from the microphone device.

Parameters:

buff Buffer in which data has to be captured.
size Size of the capture buffer

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooOpenAudioDevice ()

Opens the audio device for read/write operation.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooOpenWaveFileForRead (char * *filename*)

Opens a RAW audio data file for read.

Parameters:

filename Name of the file to be opened.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooPlayAudioBuffer (unsigned char * *buff*, long *size*)

Plays a buffer full of audio data onto the audio device.

Parameters:

buff Buffer containing the audio data to be played.
size Size of the audio data in the buffer

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooReadWaveFileData (char * *databuf*, int *size*)

Reads data from the opened raw audio file.

The number of bytes to be read is specified by the size parameter.

Parameters:

databuf Pointer to a buffer in which data is returned.
size Number of bytes to be read.

Returns:

Number of bytes read on success, -1 on failure.

Media API

Media API

Data Structures

- struct **OORTPChannel**

Defines

- #define **OO_CHAN_CLOSE** 0
- #define **OO_CHAN_OPEN** 1

Functions

- EXTERN int **ooInitializePlugin** ()
  This function initializes the plugin library.

- EXTERN int **ooCreateTransmitRTPChannel** (int *channelId, char *destip, int port)
  This function is invoked to create Transmit RTP channel.

- EXTERN int **ooCloseTransmitRTPChannel** (int channelId)
  This function is invoked to close the Transmit RTP channel.

- EXTERN int **ooCreateReceiveRTPChannel** (int *channelId, char *localip, int localport)
  This function is invoked to create Receive RTP channel.

- EXTERN int **ooCloseReceiveRTPChannel** (int channelId)

This function is invoked to close the Receive RTP channel.

- EXTERN int **ooStartTransmitWaveFile** (int channelId, char *filename)
  This function is used to transmit a wave file on already created transmit RTP channel.

- EXTERN int **ooStopTransmitWaveFile** (int channelId)
  This function is used to stop transmitting the wave file.

- EXTERN int **ooStartTransmitMic** (int channelId)
  This function is used to start capturing and transmitting the data from microphone.

- EXTERN int **ooStopTransmitMic** (int channelId)
  This function is used to stop transmitting the data from microphone.

- EXTERN int **ooStartReceiveAudioAndPlayback** (int channelId)
  This function is used to start receiving rtp stream and playing the received audio onto the speaker device.

- EXTERN int **ooStopReceiveAudioAndPlayback** (int channelId)
  This function is used to stop receiving rtp stream.

- EXTERN int **ooStartReceiveAudioAndRecord** (int channelId)
- EXTERN int **ooStopReceiveAudioAndRecord** (int channelId)

Variables

- OORTPChannel **gXmitChannel**
- OORTPChannel **gRecvChannel**
- pthread_t **gXmitThrdHdl**
- pthread_t **gRecvThrdHdl**

Function Documentation

EXTERN int ooCloseReceiveRTPChannel (int *channelId*)

This function is invoked to close the Receive RTP channel.

Parameters:

channelId An integer value indicating the RTP channel to be closed. Not used currently as only two channels are supported, 1 xmit channel and 1 recv channel.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooCloseTransmitRTPChannel (int *channelId*)

This function is invoked to close the Transmit RTP channel.

Parameters:

channelId An integer value indicating the RTP channel to be closed. Not used currently as only two channels are supported, 1 xmit channel and 1 recv channel.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooCreateReceiveRTPChannel (int * *channelId*, char * *localip*, int *localport*)

This function is invoked to create Receive RTP channel.

Parameters:

channelId Pointer to int for returning the newly created channel id. Not used currently as only two channels are supported, 1 xmit channel and 1 recv channel.
localip IP address of the local endpoint.
localport RTP receive port number at the local endpoint.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooCreateTransmitRTPChannel (int * *channelId*, char * *destip*, int *port*)

This function is invoked to create Transmit RTP channel.

Parameters:

channelId Pointer to int for returning the newly created channel id. Not used currently as only two channels are supported, 1 xmit channel and 1 recv channel.
destip IP address of the destination endpoint.
port RTP receive port number at the destination endpoint.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooInitializePlugin ()

This function initializes the plugin library.

Returns:

0, on success. -ve on failure.

EXTERN int ooStartReceiveAudioAndPlayback (int *channelId*)

This function is used to start receiving rtp stream and playing the received audio onto the speaker device.

Parameters:

channelId Indicates the receive channel on which data reception needs to be started. Not used currently as only one transmit channel is supported.

Returns:

Completion status - 0 on success, -1 on failure.

EXTERN int ooStartTransmitMic (int *channelId*)

This function is used to start capturing and transmitting the data from microphone.

Parameters:

channelId Indicates the transmit channel on data transmission should begin. Not used currently as only one transmit channel is supported.

Returns:

Completion status - 0 on success, -1 on failure.

EXTERN int ooStartTransmitWaveFile (int *channelId*, char * *filename*)

This function is used to transmit a wave file on already created transmit RTP channel.

This basically, creates a thread which will start reading from the wave file and transmit data as rtp packets on the transmit channel.

Parameters:

channelId Indicates the transmit channel to be used. Not used currently as only one transmit channel is supported.
filename Name of the wave file to be transmitted.

Returns:

Completion status - 0 on success, -1 on failure.

EXTERN int ooStopReceiveAudioAndPlayback (int *channelId*)

This function is used to stop receiving rtp stream.

Parameters:

channelId Indicates the receive channel on which data reception needs to be halted. Not used currently as only one transmit channel is supported.

Returns:

Completion status - 0 on success, -1 on failure.

EXTERN int ooStopTransmitMic (int *channelId*)

This function is used to stop transmitting the data from microphone.

Parameters:

channelId Indicates the transmit channel on which data transmission needs to be halted. Not used currently as only one transmit channel is supported.

Returns:

Completion status - 0 on success, -1 on failure.

EXTERN int ooStopTransmitWaveFile (int *channelId*)

This function is used to stop transmitting the wave file.

Parameters:

channelId Indicates the transmit channel on which wave file transmission needs to be halted. Not used currently as only one transmit channel is supported.

Returns:

Completion status - 0 on success, -1 on failure.

Media API Internal Socket Layer

Media API Internal Socket Layer

Defines

- #define **OOSOCKET_INVALID** ((**OOSOCKET**)-1)
- #define **ASN_OK** 0

- #define **ASN_E_NOTINIT** -1
- #define **ASN_E_INVSOCKET** -2
- #define **ASN_E_INVPARAM** -3
- #define **ASN_E_BUFOVFLW** -4
- #define **OOIPADDR_ANY** ((**OOIPADDR**)0)
- #define **OOIPADDR_LOCAL** ((**OOIPADDR**)0x7f000001UL) /* 127.0.0.1 */

Typedefs

- typedef int **OOSOCKET**
  Socket's handle.

- typedef unsigned long **OOIPADDR**
  The IP address represented as unsigned long value.

- typedef char **ASN1OCTET**
- typedef unsigned int **ASN1UINT**

Functions

- EXTERN int **ooSocketAccept** (**OOSOCKET** socket, **OOSOCKET** *pNewSocket, **OOIPADDR** *destAddr, int *destPort)
  This function permits an incoming connection attempt on a socket.

- EXTERN int **ooSocketAddrToStr** (**OOIPADDR** ipAddr, char *pbuf, int bufsize)
  This function converts an IP address to its string representation.

- EXTERN int **ooSocketBind** (**OOSOCKET** socket, **OOIPADDR** addr, int port)
  This function associates a local address with a socket.

- EXTERN int **ooSocketClose** (**OOSOCKET** socket)
  This function closes an existing socket.

- EXTERN int **ooSocketConnect** (**OOSOCKET** socket, const char *host, int port)
  This function establishes a connection to a specified socket.

- EXTERN int **ooSocketCreate** (**OOSOCKET** *psocket)
  This function creates a socket.

- EXTERN int **ooSocketCreateUDP** (**OOSOCKET** *psocket)
  This function creates a UDP datagram socket.

- EXTERN int **ooSocketsInit** (void)
  This function initiates use of sockets by an application.

- EXTERN int **ooSocketsCleanup** (void)

This function terminates use of sockets by an application.

- EXTERN int **ooSocketListen** (**OOSOCKET** socket, int maxConnection)
  This function places a socket a state where it is listening for an incoming connection.

- EXTERN int **ooSocketRecv** (**OOSOCKET** socket, ASN1OCTET *pbuf, ASN1UINT bufsize)
  This function receives data from a connected socket.

- EXTERN int **ooSocketRecvFrom** (**OOSOCKET** socket, ASN1OCTET *pbuf, ASN1UINT bufsize, char *remotehost, int *remoteport)
  This function receives data from a connected/unconnected socket.

- EXTERN int **ooSocketSend** (**OOSOCKET** socket, const ASN1OCTET *pdata, ASN1UINT size)
  This function sends data on a connected socket.

- EXTERN int **ooSocketSendTo** (**OOSOCKET** socket, const ASN1OCTET *pdata, ASN1UINT size, const char *remotehost, int remoteport)
  This function sends data on a connected or unconnected socket.

- EXTERN int **ooSocketSelect** (int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout)
  This function is used for synchronous monitoring of multiple sockets.

- EXTERN int **ooSocketStrToAddr** (const char *pIPAddrStr, **OOIPADDR** *pIPAddr)
  This function converts the string with IP address to a double word representation.

- EXTERN int **ooGetLocalIPAddress** (char *pIPAddrs)
  This function retrives the IP address of the local host.

- EXTERN long **ooHTONL** (long val)
  This function converts a long value from host to network byte order.

- EXTERN short **ooHTONS** (short val)
  This function converts a short value from host to network byte order.

Typedef Documentation

typedef unsigned long OOIPADDR

The IP address represented as unsigned long value.

The most significant 8 bits in this unsigned long value represent the first number of the IP address. The least significant 8 bits represent the last number of the IP address.

Definition at line 80 of file ooSock.h.

Function Documentation

EXTERN int ooGetLocalIPAddress (char * *pIPAddrs*)

This function retrives the IP address of the local host.

Parameters:

pIPAddrs Pointer to a char buffer in which local IP address will be returned.

Returns:

Completion status of operation: 0 (ASN_OK) = success, negative return value is error.

EXTERN long ooHTONL (long *val*)

This function converts a long value from host to network byte order.

Parameters:

val Value to be converted.

Returns:

Value converted to network byte order.

EXTERN short ooHTONS (short *val*)

This function converts a short value from host to network byte order.

Parameters:

val Value to be converted.

Returns:

Value converted to network byte order.

EXTERN int ooSocketAccept (OOSOCKET *socket*, OOSOCKET * *pNewSocket*, OOIPADDR * *destAddr*, int * *destPort*)

This function permits an incoming connection attempt on a socket.

It extracts the first connection on the queue of pending connections on socket. It then creates a new

socket and returns a handle to the new socket. The newly created socket is the socket that will handle the actual connection and has the same properties as original socket. See description of 'accept' socket function for further details.

Parameters:

socket The socket's handle created by call to ::rtSocketCreate function.
pNewSocket The pointer to variable to receive the new socket's handle.
destAddr Optional pointer to a buffer that receives the IP address of the connecting entity. It may be NULL.
destPort Optional pointer to a buffer that receives the port of the connecting entity. It may be NULL.

Returns:

Completion status of operation: 0 (ASN_OK) = success, negative return value is error.

EXTERN int ooSocketAddrToStr (OOIPADDR *ipAddr*, char * *pbuf*, int *bufsize*)

This function converts an IP address to its string representation.

Parameters:

ipAddr The IP address to be converted.
pbuf Pointer to the buffer to receive a string with the IP address.
bufsize Size of the buffer.

Returns:

Completion status of operation: 0 (ASN_OK) = success, negative return value is error.

EXTERN int ooSocketBind (OOSOCKET *socket*, OOIPADDR *addr*, int *port*)

This function associates a local address with a socket.

It is used on an unconnected socket before subsequent calls to the ::rtSocketConnect or ::rtSocketListen functions. See description of 'bind' socket function for further details.

Parameters:

socket The socket's handle created by call to ::rtSocketCreate function.
addr The local IP address to assign to the socket.
port The local port number to assign to the socket.

Returns:

Completion status of operation: 0 (ASN_OK) = success, negative return value is error.

EXTERN int ooSocketClose (OOSOCKET *socket*)

This function closes an existing socket.

Parameters:

socket The socket's handle created by call to ::rtSocketCreate or ::rtSocketAccept function.

Returns:

Completion status of operation: 0 (ASN_OK) = success, negative return value is error.

EXTERN int ooSocketConnect (OOSOCKET *socket*, const char * *host*, int *port*)

This function establishes a connection to a specified socket.

It is used to create a connection to the specified destination. When the socket call completes successfully, the socket is ready to send and receive data. See description of 'connect' socket function for further details.

Parameters:

socket The socket's handle created by call to ::rtSocketCreate function.
host The null-terminated string with the IP address in the following format:
"NNN.NNN.NNN.NNN", where NNN is a number in the range (0..255).
port The destination port to connect.

Returns:

Completion status of operation: 0 (ASN_OK) = success, negative return value is error.

EXTERN int ooSocketCreate (OOSOCKET * *psocket*)

This function creates a socket.

The only streaming TCP/IP sockets are supported at the moment.

Parameters:

psocket The pointer to the socket's handle variable to receive the handle of new socket.

Returns:

Completion status of operation: 0 (ASN_OK) = success, negative return value is error.

EXTERN int ooSocketCreateUDP (OOSOCKET * *psocket*)

This function creates a UDP datagram socket.

Parameters:

psocket The pointer to the socket's handle variable to receive the handle of new socket.

Returns:

Completion status of operation: 0 (ASN_OK) = success, negative return value is error.

EXTERN int ooSocketListen (OOSOCKET *socket*, int *maxConnection*)

This function places a socket a state where it is listening for an incoming connection.

To accept connections, a socket is first created with the ::rtSocketCreate function and bound to a local address with the ::rtSocketBind function, a maxConnection for incoming connections is specified with ::rtSocketListen, and then the connections are accepted with the ::rtSocketAccept function. See description of 'listen' socket function for further details.

Parameters:

socket The socket's handle created by call to ::rtSocketCreate function.
maxConnection Maximum length of the queue of pending connections.

Returns:

Completion status of operation: 0 (ASN_OK) = success, negative return value is error.

EXTERN int ooSocketRecv (OOSOCKET *socket*, ASN1OCTET * *pbuf*, ASN1UINT *bufsize*)

This function receives data from a connected socket.

It is used to read incoming data on sockets. The socket must be connected before calling this function. See description of 'recv' socket function for further details.

Parameters:

socket The socket's handle created by call to ::rtSocketCreate or ::rtSocketAccept function.
pbuf Pointer to the buffer for the incoming data.
bufsize Length of the buffer.

Returns:

If no error occurs, returns the number of bytes received. Otherwise, the negative value is error code.

EXTERN int ooSocketRecvFrom (OOSOCKET *socket*, ASN1OCTET * *pbuf*, ASN1UINT *bufsize*, char * *remotehost*, int * *remoteport*)

This function receives data from a connected/unconnected socket.

It is used to read incoming data on sockets. It populates the remotehost and remoteport parameters with information of remote host. See description of 'recvfrom' socket function for further details.

Parameters:

socket The socket's handle created by call to ooSocketCreate
pbuf Pointer to the buffer for the incoming data.
bufsize Length of the buffer.
remotehost Pointer to a buffer in which remote ip address will be returned.
remoteport Pointer to an int in which remote port number will be returned.

Returns:

If no error occurs, returns the number of bytes received. Otherwise, the negative value is error code.

EXTERN int ooSocketsCleanup (void)

This function terminates use of sockets by an application.

This function must be called after done with sockets.

Returns:

Completion status of operation: 0 (ASN_OK) = success, negative return value is error.

EXTERN int ooSocketSelect (int *nfds*, fd_set * *readfds*, fd_set * *writefds*, fd_set * *exceptfds*, struct timeval * *timeout*)

This function is used for synchronous monitoring of multiple sockets.

For more information refer to documnetation of "select" system call.

Parameters:

nfds The highest numbered descriptor to be monitored plus one.
readfds The descriptors listed in readfds will be watched for whether read would block on them.
writefds The descriptors listed in writefds will be watched for whether write would block on them.
exceptfds The descriptors listed in exceptfds will be watched for exceptions.
timeout Upper bound on amout of time elapsed before select returns.

Returns:

Completion status of operation: 0 (ASN_OK) = success, negative return value is error.

EXTERN int ooSocketSend (OOSOCKET *socket*, const ASN1OCTET * *pdata*, ASN1UINT *size*)

This function sends data on a connected socket.

It is used to write outgoing data on a connected socket. See description of 'send' socket function for further details.

Parameters:

socket The socket's handle created by call to ::rtSocketCreate or ::rtSocketAccept function.
pdata Buffer containing the data to be transmitted.

size Length of the data in pdata.

Returns:

Completion status of operation: 0 (ASN_OK) = success, negative return value is error.

EXTERN int ooSocketSendTo (OOSOCKET *socket*, const ASN1OCTET * *pdata*, ASN1UINT *size*, const char * *remotehost*, int *remoteport*)

This function sends data on a connected or unconnected socket.

See description of 'sendto' socket function for further details.

Parameters:

socket The socket's handle created by call to ::rtSocketCreate or ::rtSocketAccept function.
pdata Buffer containing the data to be transmitted.
size Length of the data in pdata.
remotehost Remote host ip address to which data has to be sent.
remoteport Remote port ip address to which data has to be sent.

Returns:

Completion status of operation: 0 (ASN_OK) = success, negative return value is error.

EXTERN int ooSocketsInit (void)

This function initiates use of sockets by an application.

This function must be called first before use sockets.

Returns:

Completion status of operation: 0 (ASN_OK) = success, negative return value is error.

EXTERN int ooSocketStrToAddr (const char * *pIPAddrStr*, OOIPADDR * *pIPAddr*)

This function converts the string with IP address to a double word representation.

The converted address may be used with the ::rtSocketBind function.

Parameters:

pIPAddrStr The null-terminated string with the IP address in the following format: "NNN.NNN.NNN.NNN", where NNN is a number in the range (0..255).
pIPAddr Pointer to the converted IP address.

Returns:

Completion status of operation: 0 (ASN_OK) = success, negative return value is error.

Media API Internal Windows Helper Functions

Media API Internal Windows Helper Functions

Data Structures

- struct **ooWaveFile**
  Helper structure for reading wavefile.

- struct **WaveBuffer**
  This holds a list of buffers holding recorded data from MIC.

- struct **BufferNode**
  This holds the list of free buffers which can be used for sending data to waveout device.

Functions

- EXTERN int **ooOpenWaveFileForRead** (char *filename)
  Opens a WaveFile for read and traverses upto the data chunk, so that next mmioRead will return wavedata.

- EXTERN int **ooCloseWaveFile** ()
  Closes the open WaveFile.

- EXTERN int **ooReadWaveFileData** (char *buffer, int size)
  Reads data from already opened wave file.

- EXTERN int **ooOpenSpeaker** (HWAVEOUT *phWaveOut, WAVEFORMATEX waveFormat)
  Opens a waveOut device, i.e., speaker for playback.

- EXTERN int **ooPlayWaveBuffer** (HWAVEOUT hWaveOut, unsigned char *buff, long size)
  Plays the number of bytes specified by the size parameter from the buff onto the speaker device.

- EXTERN int **ooCloseSpeaker** (HWAVEOUT hWaveOut)
  Release all the buffers queued into the waveOut device(speaker) for playback and close speaker.

- EXTERN int **ooOpenMic** ()
  Opens the WaveIn device, MIC, for recording and queues the buffers into the device which can be used for storing recorded data.

- EXTERN int **ooStartMicDataCapture** (HWAVEIN hWaveIn)

Start recording audio into the buffers.

- EXTERN int **ooStopMicDataCapture** (HWAVEIN hWaveIn)
  Stop recording audio.

- EXTERN int **ooCloseMic** (HWAVEIN hWaveIn)
  This releases all the buffers queued inside wave-in device, mic, and then closes the mic.

- EXTERN void CALLBACK **ooMICCallback** (HWAVEIN hwi, UINT uMsg, DWORD dwInstance, DWORD dwParam1, DWORD dwParam2)
  This is a callback function registered with the mic.

- EXTERN void CALLBACK **ooSpeakerCallback** (HWAVEOUT hwo, UINT uMsg, DWORD dwInstance, DWORD dwParam1, DWORD dwParam2)
  This is a callback function registered with the speaker.

- EXTERN int **ooAddToWaveBufferList** (WAVEHDR *waveHdr)
  Helper function to maintain the list of buffers returned by mic, after recording.

- EXTERN int **ooRemoveHeadOfWaveBufferList** ()
  Helper function which removes the WAVEHDR at the front of the list.

- EXTERN int **ooAddToFreeBufferList** (char *buffer)
  Helper function to put a buffer back into the free buffer list.

- EXTERN char * **ooGetFreeBuffer** ()
  Get a free buffer from free buffer list.

Variables

- **ooWaveFile gWaveFile**
  Global handle to open wave file.

- **WaveBuffer * gpWaveHead**
  Global pointers to list of wave buffers.

- **WaveBuffer * gpWaveTail**
  Global pointers to list of wave buffers.

- int **gRecording**
- int **gQueuedBufCount**
  Count of number of buffers queued inside the wave-in device.

- **BufferNode * gpFreeBufHead**

Global pointers to the list holding free buffers.

- **BufferNode * gpFreeBufTail**
  Global pointers to the list holding free buffers.

- int **gPlayQueueCount**
- HWAVEIN **ghWaveIn**
  Global handle to open wave-in device.

- HWAVEOUT **ghWaveOut**
  Global handle to open wave-out device.

- CRITICAL_SECTION **gPlayMutex**
  As callback functions run in their own threads, we need mutex protection for data which is used by multiple threads.

- CRITICAL_SECTION **gReadMutex**
  As callback functions run in their own threads, we need mutex protection for data which is used by multiple threads.

Function Documentation

EXTERN int ooAddToFreeBufferList (char * *buffer*)

Helper function to put a buffer back into the free buffer list.

Once the buffer playback is done, it can be added to the free list

Parameters:

buffer Pointer to the buffer to be added to the free list.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooAddToWaveBufferList (WAVEHDR * *waveHdr*)

Helper function to maintain the list of buffers returned by mic, after recording.

These buffers are kept in this list till there processing is done and then again queued back into the mic for further recording.

Parameters:

waveHdr Pointer to wave header structure which in turn contains the data buffer.

Returns:

Completion status - 0 on success, -1 on failure.

EXTERN int ooCloseMic (HWAVEIN *hWaveIn*)

This releases all the buffers queued inside wave-in device, mic, and then closes the mic.

Parameters:

hWaveIn Handle to the wave-in device.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooCloseSpeaker (HWAVEOUT *hWaveOut*)

Release all the buffers queued into the waveOut device(speaker) for playback and close speaker.

Parameters:

hWaveOut Handle to the wave out device to be closed.

Returns:

Completion status - 0 on success, -1 on failure.

EXTERN int ooCloseWaveFile ()

Closes the open WaveFile.

Returns:

Completion status - 0 on success, -1 on failure.

EXTERN char* ooGetFreeBuffer ()

Get a free buffer from free buffer list.

A free buffer is retrieved using this function for storing data which will then passed onto ooPlayWaveBuffer function for playback.

Returns:

Pointer to a character buffer

EXTERN void CALLBACK ooMICCallback (HWAVEIN *hwi*, UINT *uMsg*, DWORD *dwInstance*, DWORD *dwParam1*, DWORD *dwParam2*)

This is a callback function registered with the mic.

It will be called by mic device when a buffer full of data is recorded.

Parameters:

hwi Handle to the wave-in device.
uMsg Event message sent by the device.
dwInstance User data.
dwParam1 Message parameter.
dwParam2 Message parameter.

Returns:

None

EXTERN int ooOpenMic ()

Opens the WaveIn device, MIC, for recording and queues the buffers into the device which can be used for storing recorded data.

Returns:

Completion status - 0 on success, -1 on failure.

EXTERN int ooOpenSpeaker (HWAVEOUT * *phWaveOut*, WAVEFORMATEX *waveFormat*)

Opens a waveOut device, i.e., speaker for playback.

waveFormat specifies the format to be used for playback.

Parameters:

phWaveOut Pointer to an empty HWAVEOUT handle which will contain the handle to the opened device on return.
waveFormat Wave format to be used for playback.

Returns:

Completion status - 0 on success, -1 on failure.

EXTERN int ooOpenWaveFileForRead (char * *filename*)

Opens a WaveFile for read and traverses upto the data chunk, so that next mmioRead will return wavedata.

Parameters:

filename Name of the wave file to be opened.

Returns:

Completion status - 0 on success, -1 on failure.

EXTERN int ooPlayWaveBuffer (HWAVEOUT *hWaveOut*, unsigned char * *buff*, long *size*)

Plays the number of bytes specified by the size parameter from the buff onto the speaker device.

Parameters:

hWaveOut Handle to the speaker device.
buff Pointer to the buffer containing the data to be played.
size Size of the buffer to be played out.

Returns:

Completion status - 0 on success, -1 on failure.

EXTERN int ooReadWaveFileData (char * *buffer*, int *size*)

Reads data from already opened wave file.

Number of bytes to be read is specified by the "size" parameter. The data is returned in "buffer" parameter and the number of bytes read is returned as a return value.

Parameters:

buffer Buffer which will contain the data read.
size Size of the buffer passed.

Returns:

Number of bytes read on success, -1 on failure

EXTERN int ooRemoveHeadOfWaveBufferList ()

Helper function which removes the WAVEHDR at the front of the list.

Note this does not free up the mem used by WAVEHDR as it will be queued into the mic for further recording.

Returns:

Completion status - 0 on success, -1 on failure.

EXTERN void CALLBACK ooSpeakerCallback (HWAVEOUT *hwo*, UINT *uMsg*, DWORD *dwInstance*, DWORD *dwParam1*, DWORD *dwParam2*)

This is a callback function registered with the speaker.

It is called by the speaker, when a buffer full of data is played back.

Parameters:

hwo Handle to the wave out device.
uMsg Waveform audio output message.
dwInstance User instance data passed during waveOutOpen.
dwParam1 Message parameter.
dwParam2 Message parameter

Returns:

None

EXTERN int ooStartMicDataCapture (HWAVEIN *hWaveIn*)

Start recording audio into the buffers.

Parameters:

hWaveIn Handle to wave-in device.

Returns:

Completion status - 0 on success, -1 on failure.

EXTERN int ooStopMicDataCapture (HWAVEIN *hWaveIn*)

Stop recording audio.

Note that this marks current buffer as done and frees it by calling callback function. However, other queued buffers will stay there in wave-in device.

Parameters:

hWaveIn Handle to the wave-in device.

Returns:

Completion status - 0 on success, -1 on failure

MediaPlug-inlibrary Data Structure Documentation

BufferNode Struct Reference

BufferNodeThis holds the list of free buffers which can be used for sending data to waveout device.

`#include <ooWave.h>`

Data Fields

- char * **buf**
- **BufferNode * next**

Detailed Description

This holds the list of free buffers which can be used for sending data to waveout device.

Definition at line 83 of file ooWave.h.

The documentation for this struct was generated from the following file:

- **ooWave.h**

ooWaveFile Struct Reference

ooWaveFileHelper structure for reading wavefile.

`#include <ooWave.h>`

Data Fields

- char **filename** [1024]
- HMMIO **hWaveFile**
  Wave file name.

- WAVEFORMATEX **waveFormat**
  Wave file handle.

- int **dataSize**
  Wave file format.

Detailed Description

Helper structure for reading wavefile.

Definition at line 54 of file ooWave.h.

The documentation for this struct was generated from the following file:

- **ooWave.h**

WaveBuffer Struct Reference

WaveBufferThis holds a list of buffers holding recorded data from MIC.

`#include <ooWave.h>`

Data Fields

- WAVEHDR * **pWaveHdr**
- **WaveBuffer * next**

Detailed Description

This holds a list of buffers holding recorded data from MIC.

This data can then be sent on rtp channel or played back on speakers.

Definition at line 70 of file ooWave.h.

The documentation for this struct was generated from the following file:

- **ooWave.h**

MediaPlug-inlibrary File Documentation

ooCommon.h File Reference

ooCommon.hThis file contains common helper functions.

```
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include "ooSock.h"
```

Defines

- #define **OORTPPACKETDATASIZE** 240 /* Send Receive Packet Data Size*/
- #define **MAXLOGMSGLEN** 1024
- #define **OOLOG2**(a, b) ooLog(a,b)
- #define **OOLOG3**(a, b, c) ooLog(a,b,c)
- #define **OOLOG4**(a, b, c, d) ooLog(a,b,c,d)
- #define **OOLOG5**(a, b, c, d, e) ooLog(a,b,c,d,e)
- #define **OOLOG9**(a, b, c, d, e, f, g, h, i) ooLog(a,b,c,d,e,f,g,h,i)

Functions

- EXTERN void **ooLog** (int level, const char *fmtspec,...)
  This function logs a trace message into a log file.

- EXTERN void **ooSleep** (int milliseconds)
  Platform independent sleep function.

Variables

- FILE * **fpLog**

Detailed Description

This file contains common helper functions.

Definition in file **ooCommon.h**.

oortp.h File Reference

oortp.hThis file contains functions to create and use media channels.

```
#include <stdio.h>
#include "ooSock.h"
#include "ooCommon.h"
```

Data Structures

- struct **OORTPChannel**

Defines

- #define **OO_CHAN_CLOSE** 0
- #define **OO_CHAN_OPEN** 1

Functions

- EXTERN int **ooInitializePlugin** ()
  This function initializes the plugin library.

- EXTERN int **ooCreateTransmitRTPChannel** (int *channelId, char *destip, int port)
  This function is invoked to create Transmit RTP channel.

- EXTERN int **ooCloseTransmitRTPChannel** (int channelId)
  This function is invoked to close the Transmit RTP channel.

- EXTERN int **ooCreateReceiveRTPChannel** (int *channelId, char *localip, int localport)
  This function is invoked to create Receive RTP channel.

- EXTERN int **ooCloseReceiveRTPChannel** (int channelId)
  This function is invoked to close the Receive RTP channel.

- EXTERN int **ooStartTransmitWaveFile** (int channelId, char *filename)
  This function is used to transmit a wave file on already created transmit RTP channel.

- EXTERN int **ooStopTransmitWaveFile** (int channelId)
  This function is used to stop transmitting the wave file.

- EXTERN int **ooStartTransmitMic** (int channelId)
  This function is used to start capturing and transmitting the data from microphone.

- EXTERN int **ooStopTransmitMic** (int channelId)
  This function is used to stop transmitting the data from microphone.

- EXTERN int **ooStartReceiveAudioAndPlayback** (int channelId)
  This function is used to start receiving rtp stream and playing the received audio onto the speaker device.

- EXTERN int **ooStopReceiveAudioAndPlayback** (int channelId)
  This function is used to stop receiving rtp stream.

- EXTERN int **ooStartReceiveAudioAndRecord** (int channelId)
- EXTERN int **ooStopReceiveAudioAndRecord** (int channelId)

Variables

- OORTPChannel **gXmitChannel**
- OORTPChannel **gRecvChannel**
- pthread_t **gXmitThrdHdl**
- pthread_t **gRecvThrdHdl**

Detailed Description

This file contains functions to create and use media channels.

Definition in file **oortp.h**.

ooSock.h File Reference

ooSock.hCommon runtime constants, data structure definitions, and run-time functions to support the sockets' operations.

```
#include <sys/types.h>
#include "sys/time.h"
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <arpa/inet.h>
```

Defines

- #define **EXTERN**
- #define **OOSOCKET_INVALID** ((**OOSOCKET**)-1)
- #define **ASN_OK** 0
- #define **ASN_E_NOTINIT** -1
- #define **ASN_E_INVSOCKET** -2
- #define **ASN_E_INVPARAM** -3
- #define **ASN_E_BUFOVFLW** -4
- #define **OOIPADDR_ANY** ((**OOIPADDR**)0)
- #define **OOIPADDR_LOCAL** ((**OOIPADDR**)0x7f000001UL) /* 127.0.0.1 */

Typedefs

- typedef int **OOSOCKET**
  Socket's handle.

- typedef unsigned long **OOIPADDR**
  The IP address represented as unsigned long value.

- typedef char **ASN1OCTET**
- typedef unsigned int **ASN1UINT**

Functions

- EXTERN int **ooSocketAccept** (**OOSOCKET** socket, **OOSOCKET** *pNewSocket, **OOIPADDR** *destAddr, int *destPort)
  This function permits an incoming connection attempt on a socket.

- EXTERN int **ooSocketAddrToStr** (**OOIPADDR** ipAddr, char *pbuf, int bufsize)
  This function converts an IP address to its string representation.

- EXTERN int **ooSocketBind** (**OOSOCKET** socket, **OOIPADDR** addr, int port)
  This function associates a local address with a socket.

- EXTERN int **ooSocketClose** (**OOSOCKET** socket)
  This function closes an existing socket.

- EXTERN int **ooSocketConnect** (**OOSOCKET** socket, const char *host, int port)
  This function establishes a connection to a specified socket.

- EXTERN int **ooSocketCreate** (**OOSOCKET** *psocket)
  This function creates a socket.

- EXTERN int **ooSocketCreateUDP** (**OOSOCKET** *psocket)
  This function creates a UDP datagram socket.

- EXTERN int **ooSocketsInit** (void)
  This function initiates use of sockets by an application.

- EXTERN int **ooSocketsCleanup** (void)
  This function terminates use of sockets by an application.

- EXTERN int **ooSocketListen** (**OOSOCKET** socket, int maxConnection)
  This function places a socket a state where it is listening for an incoming connection.

- EXTERN int **ooSocketRecv** (**OOSOCKET** socket, ASN1OCTET *pbuf, ASN1UINT bufsize)
  This function receives data from a connected socket.

- EXTERN int **ooSocketRecvFrom** (**OOSOCKET** socket, ASN1OCTET *pbuf, ASN1UINT bufsize, char *remotehost, int *remoteport)
  This function receives data from a connected/unconnected socket.

- EXTERN int **ooSocketSend** (**OOSOCKET** socket, const ASN1OCTET *pdata, ASN1UINT size)
  This function sends data on a connected socket.

- EXTERN int **ooSocketSendTo** (**OOSOCKET** socket, const ASN1OCTET *pdata, ASN1UINT size, const char *remotehost, int remoteport)
  This function sends data on a connected or unconnected socket.

- EXTERN int **ooSocketSelect** (int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout)
  This function is used for synchronous monitoring of multiple sockets.

- EXTERN int **ooSocketStrToAddr** (const char *pIPAddrStr, **OOIPADDR** *pIPAddr)
  This function converts the string with IP address to a double word representation.

- EXTERN int **ooGetLocalIPAddress** (char *pIPAddrs)

This function retrives the IP address of the local host.

- EXTERN long **ooHTONL** (long val)
  This function converts a long value from host to network byte order.

- EXTERN short **ooHTONS** (short val)
  This function converts a short value from host to network byte order.

Detailed Description

Common runtime constants, data structure definitions, and run-time functions to support the sockets' operations.

Definition in file **ooSock.h**.

ooWave.h File Reference

ooWave.hThis file contains low level wave functions.

```
#include <stdio.h>
#include "ooCommon.h"
#include <Windows.h>
#include <Mmsystem.h>
#include <Basetsd.h>
```

Data Structures

- struct **ooWaveFile**
  Helper structure for reading wavefile.

- struct **WaveBuffer**
  This holds a list of buffers holding recorded data from MIC.

- struct **BufferNode**
  This holds the list of free buffers which can be used for sending data to waveout device.

Functions

- EXTERN int **ooOpenWaveFileForRead** (char *filename)
  Opens a WaveFile for read and traverses upto the data chunk, so that next mmioRead will return wavedata.

- EXTERN int **ooCloseWaveFile** ()
  Closes the open WaveFile.

- EXTERN int **ooReadWaveFileData** (char *buffer, int size)
  Reads data from already opened wave file.

- EXTERN int **ooOpenSpeaker** (HWAVEOUT *phWaveOut, WAVEFORMATEX waveFormat)
  Opens a waveOut device, i.e., speaker for playback.

- EXTERN int **ooPlayWaveBuffer** (HWAVEOUT hWaveOut, unsigned char *buff, long size)
  Plays the number of bytes specified by the size parameter from the buff onto the speaker device.

- EXTERN int **ooCloseSpeaker** (HWAVEOUT hWaveOut)
  Release all the buffers queued into the waveOut device(speaker) for playback and close speaker.

- EXTERN int **ooOpenMic** ()
  Opens the WaveIn device, MIC, for recording and queues the buffers into the device which can be

used for storing recorded data.

- EXTERN int **ooStartMicDataCapture** (HWAVEIN hWaveIn)
  Start recording audio into the buffers.

- EXTERN int **ooStopMicDataCapture** (HWAVEIN hWaveIn)
  Stop recording audio.

- EXTERN int **ooCloseMic** (HWAVEIN hWaveIn)
  This releases all the buffers queued inside wave-in device, mic, and then closes the mic.

- EXTERN void CALLBACK **ooMICCallback** (HWAVEIN hwi, UINT uMsg, DWORD dwInstance, DWORD dwParam1, DWORD dwParam2)
  This is a callback function registered with the mic.

- EXTERN void CALLBACK **ooSpeakerCallback** (HWAVEOUT hwo, UINT uMsg, DWORD dwInstance, DWORD dwParam1, DWORD dwParam2)
  This is a callback function registered with the speaker.

- EXTERN int **ooAddToWaveBufferList** (WAVEHDR *waveHdr)
  Helper function to maintain the list of buffers returned by mic, after recording.

- EXTERN int **ooRemoveHeadOfWaveBufferList** ()
  Helper function which removes the WAVEHDR at the front of the list.

- EXTERN int **ooAddToFreeBufferList** (char *buffer)
  Helper function to put a buffer back into the free buffer list.

- EXTERN char * **ooGetFreeBuffer** ()
  Get a free buffer from free buffer list.

Variables

- **ooWaveFile gWaveFile**
  Global handle to open wave file.

- **WaveBuffer * gpWaveHead**
  Global pointers to list of wave buffers.

- **WaveBuffer * gpWaveTail**
  Global pointers to list of wave buffers.

- int **gRecording**
- int **gQueuedBufCount**

Count of number of buffers queued inside the wave-in device.

- **BufferNode * gpFreeBufHead**
  Global pointers to the list holding free buffers.

- **BufferNode * gpFreeBufTail**
  Global pointers to the list holding free buffers.

- int **gPlayQueueCount**
- HWAVEIN **ghWaveIn**
  Global handle to open wave-in device.

- HWAVEOUT **ghWaveOut**
  Global handle to open wave-out device.

- CRITICAL_SECTION **gPlayMutex**
  As callback functions run in their own threads, we need mutex protection for data which is used by multiple threads.

- CRITICAL_SECTION **gReadMutex**
  As callback functions run in their own threads, we need mutex protection for data which is used by multiple threads.

Detailed Description

This file contains low level wave functions.

Definition in file **ooWave.h**.

Index