Description of VoiceChat Library

Version 1.0.0

The contents in this document are highly confidential and should be handled accordingly.

NTR-06-0466-001-A
Released: August 14, 2006

Table of Contents

Code

Tables

Figures

Revision History

Version	Revision Date	Description
1.0.0	05/01/2006	Initial version.

1 Introduction

The VoiceChat Library is a set of libraries used for voice communication over a TCP/IP network. It contains libraries for voice communications and controlling incoming/receiving functions.

This document describes the basic structure and use of the libraries.

The VoiceChat Library does not require specific hardware and software for voice communication, but the NITRO-SDK Wi-Fi library and NITRO-DWC must be used for communication.

VoiceChat Library enables users to engage in three different conversation styles:

- One-to-one (Telephone Mode)
- One-to-group (Transceiver Mode)
- Group-to-group simultaneously (Conference Mode)

2 Modes of Use

The VoiceChat Library uses the following communication modes:

- One-to-one (Telephone Mode)
- One-to-group (Transceiver Mode)
- Group-to-group simultaneously (Conference Mode)

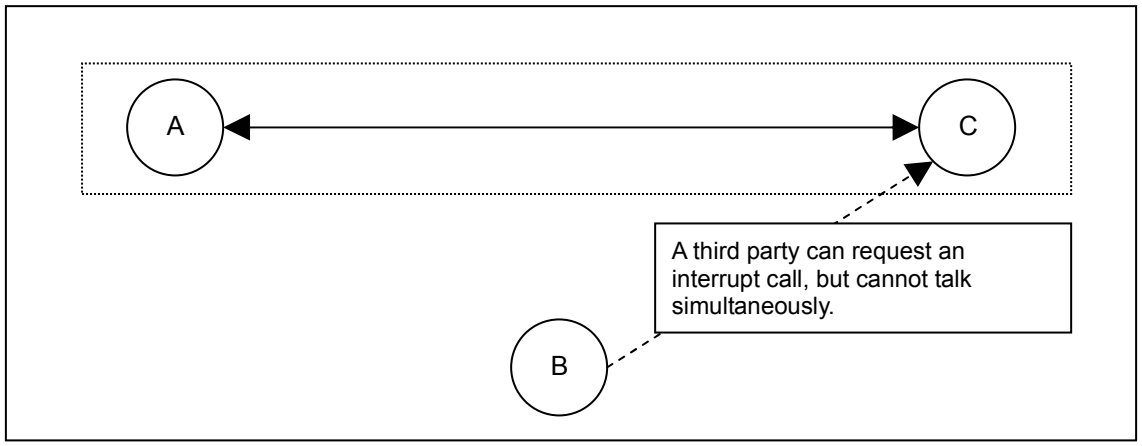
2.1 One-to-One Conversation

One-to-one conversation begins similarly to an ordinary telephone conversation: the calling party calls the receiving party and the receiving party responds to the call. This mode is defined as Telephone Mode in VoiceChat Library.

The library supports Call Waiting and Interrupt Call functionality, but the actual conversation can only take place between the calling party and the receiving party. Three-way calling is not supported.

Telephone Mode is ideally designed for game situations requiring a player to discuss specific game content with another player while playing. For example, a problem-solving game which does not require much instantaneous action, such as an RPG, is ideal for Telephone Mode.

Figure 2-1 One-to-One Conversation



In the above figure, parties A and C are talking together. B can send a request to start a conversation with C during the conversation between C and A, but C must terminate the conversation with A to start talking with B.

2.2 Group Conversation

Two modes are available for group conversation:

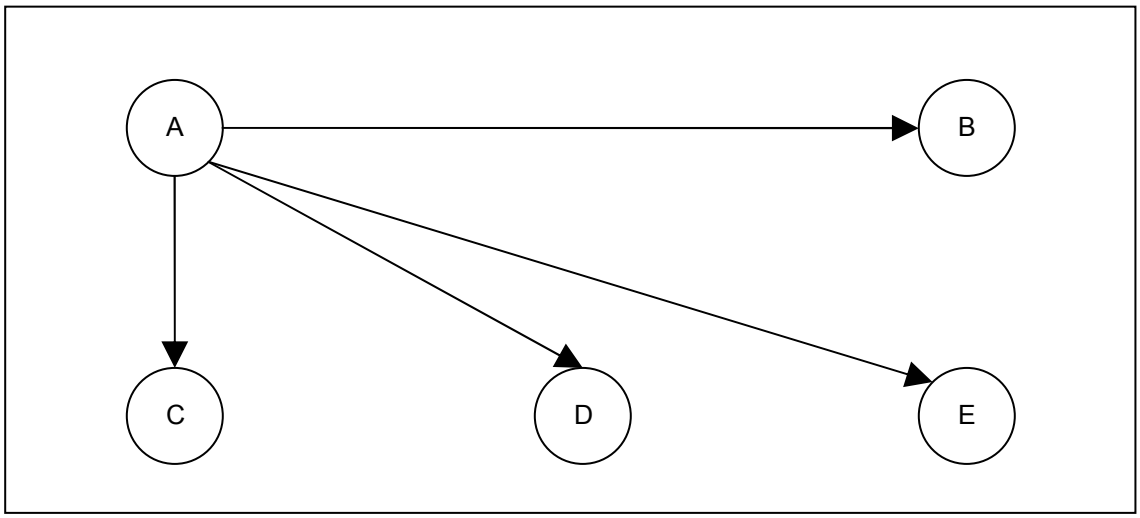
- Transceiver Mode: This mode enables one-to-group conversation
- Conference Mode: This mode enables simultaneous talking between all participating parties

2.2.1 Conversation in Transceiver Mode

Transceiver Mode allows one party to speak at a time. It is designed for situations where short, instantaneous communication is helpful for all players involved.

For example, a multi-player action game where one player gives quick information to other players is ideal for Transceiver Mode. Transceiver Mode supports a maximum of eight (8) parties.

Figure 2-2 One-to-Group Conversation



In the above figure, five parties (parties A through E) participate in the chat session. Party A talks to all of the other parties; the other parties cannot talk and requests to talk are denied when party A controls the conversation.

In this mode, one host machine must control 1) who participates in the session, 2) who can talk, and 3) who can send a request to talk. The game developer should specify the host machine in advance.

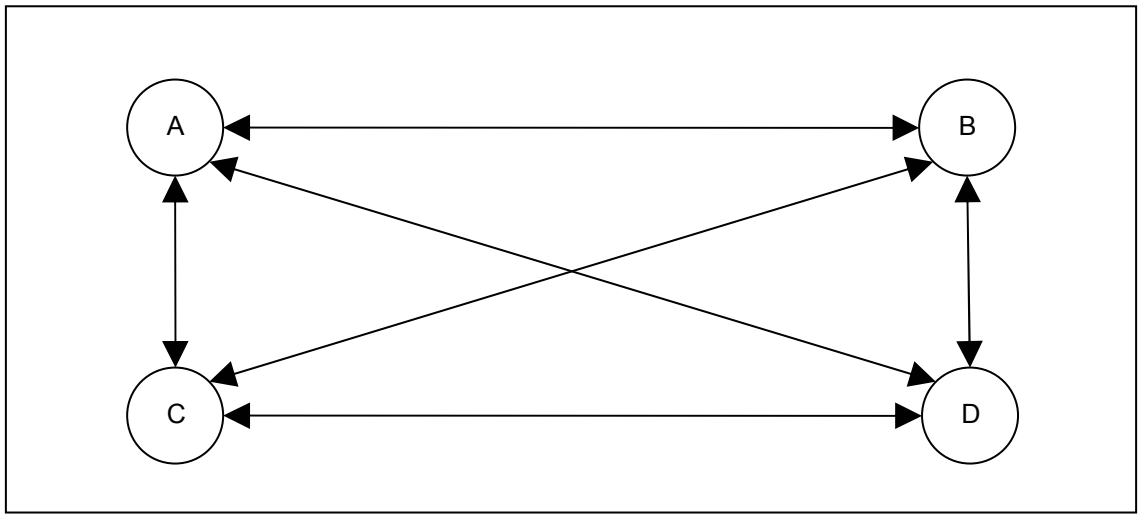
* VoiceChat Library allows conversation with up to eight (8) parties. However, the actual number of parties that can talk depends upon the availability of network bandwidth and the audio format bit rate. See Section 5.2: Audio Format for more about necessary bandwidth.

2.2.2 Conversation in Conference Mode

Conference Mode allows all participating parties to talk simultaneously.

It is designed for situations where all players benefit by the ability to talk without waiting or performing an action - such as a keystroke - to obtain the right to talk. For example, an action game where players need to remain in constant, instantaneous contact is ideal in Conference Mode. The maximum number of people that can participate in this mode is four (4).

Figure 2-3 Group-to-Group Conversation



In the above figure, four parties (parties A through D) participate in the session. Unlike Transceiver Mode, parties B, C, and D can talk while A is talking.

In this mode, no host machine is required because controlling party participation (who is talking and who can make requests to talk) is unnecessary. However, terminals that participate in Conference Mode must be determined on the game's side.

VoiceChat Library does not have a software mixer. The NintendoDS system mixer may be used (in this case, a maximum of 3 channels are used), or a game application's mixer may be used.

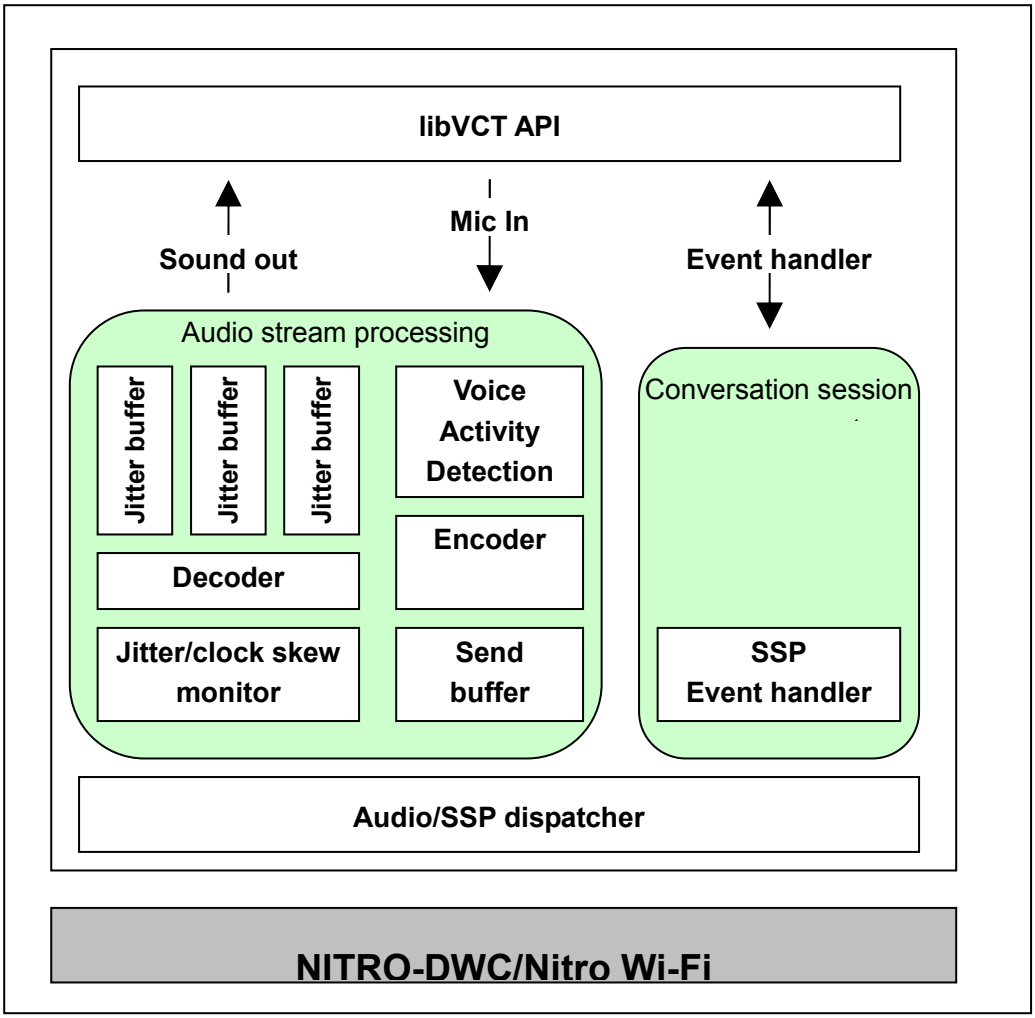
VoiceChat Library allows conversation with up to four (4) parties. However, the actual number of parties that can talk depends upon the availability of network bandwidth and the audio format bit rate. See Section 5.2: Audio Format, for more about necessary bandwidth.

3 Library Overview

VoiceChat Library has two function blocks:

- Conversation session management
- Audio stream processing

Figure 3-1 Library Overview



4 Conversation Session Management

4.1 Simple Session Protocol

Clients connected by NITRO-DWC (matchmaking) start VoiceChat by using a protocol called Simple Session Protocol (SSP).

SSP is similar to Session Initiation Protocol (SIP); however, SSP has been customized for high-speed operation on the Nintendo DS. SSP uses a binary format to optimize processing speed and implementation size.

4.2 Format

SSP has a unique format of 16-bit binary data as shown in the table below. The data byte order is little-endian.

Table 4-1 SSP Format

Information	Length	Description
Magic Token	4 byte	'_VCT' little-endian
Method type	1 byte	0x00 or 0xFF
Version	1 byte	Version number (0x10)
Message code	1 byte	Request and Response codes
Codec	1 byte	Format number of audio format currently used
AID	2 byte	Transmitter, Receiver AID
Extension information	1 byte	Extension information for NOTIFY method
Right to talk AID	1 byte	AID with right to talk in Transceiver Mode
AID bitmap	4 byte	Client list in Transceiver Mode

The first four bytes of SSP are known as Magic Token and are used for identification because SSP sends data using NITRO-DWC functions (`DWC_SendReliable`). The first four bytes of data used for gaming must be different than those used by VoiceChat Library

4.3 Request Method and Response Code

SSP supports the following request methods and response codes:

Table 4-2 SSP Request Method Mode

Method Name	Description	Code
INVITE	Request to establish a session (request for connection)	00
BYE	Request to end a session (request for disconnection)	01
CANCEL	Cancel a request to establish a session	02
CONTACT	Request for transceiver	03
NOTIFY	Notice of status information	04

Table 4-3 SSP Response Code

Method Name	Description	Code
200 OK	Request was successfully accepted	00
400 Bad Request	Request is invalid	01
406 Not Acceptable	Requested media type is not supported	02
486 Busy Here	Client is now busy	03
487 Request Terminated	Request was terminated by either BYE or CANCEL	04
603 Decline	Request was denied. Receiving party does not want talk or cannot respond.	05

4.4 Event Code

In VoiceChat Library, the following event codes are assigned in accordance with the SSP sequence. The Library user programs know how to respond to a request (or response) by dispatching the following event codes:

Table 4-4 Event Code

Event Name	Description
NONE	Nothing happened
INCOMING	Received a request to talk (INVITE)
REJECT	Denied by other party (denial INVITE or different codec)
BUSY	Other party is busy
CANCEL	Received a cancellation of call
NOTIFY_FREE	Conversation with other party ended in Transceiver Mode
NOTIFY_BUSY	Conversation with other party started in Transceiver Mode
CONNECTED	Connection was established
RESPONDBYE	Received BYE request
DISCONNECTED	Disconnected (200 OK against BYE or 487 Request Terminated against CANCEL)
CONTACT	Received CONTACT
TIMEOUT	Request was timed out
ABORT	Session was terminated by invalid sequence

4.5 Sequence in Telephone Mode

This section describes the sequence that occurs in Telephone Mode. The sequence in Telephone Mode is the fundamental framework used in both Transceiver Mode and Conference Mode.

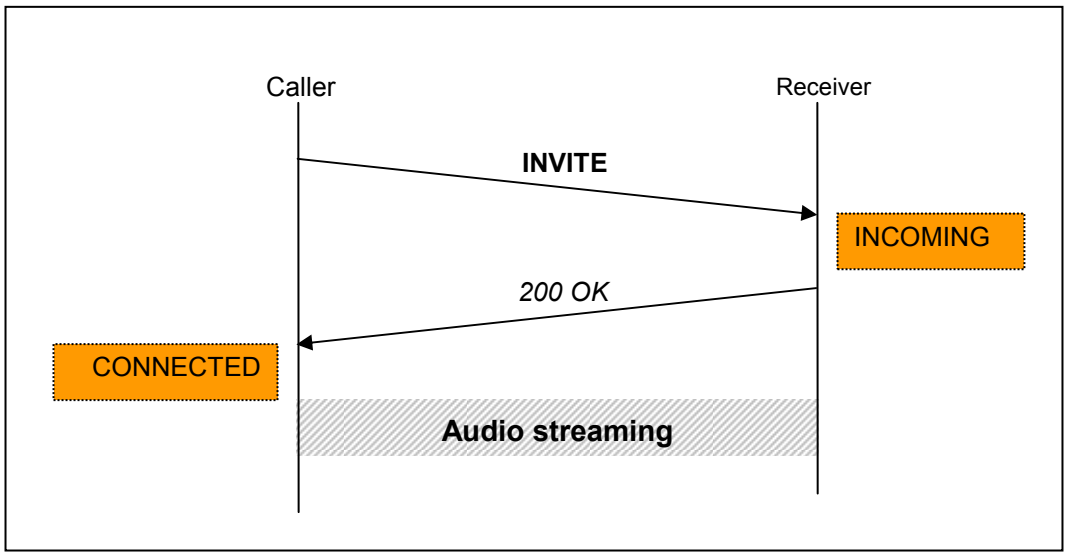
Request methods are distinguished by bold font and Response code is distinguished by italicized font. Font using an orange background indicates an event that occurs when request or response is received.

4.5.1 Request to Talk

A caller issues an INVITE method to place a request to talk. A conversation is established when a receiver returns **200 OK**.

The process occurring after **200 OK** is for audio data in real time and is handled by a protocol other than SSP.

Figure 4-1 Request to Talk

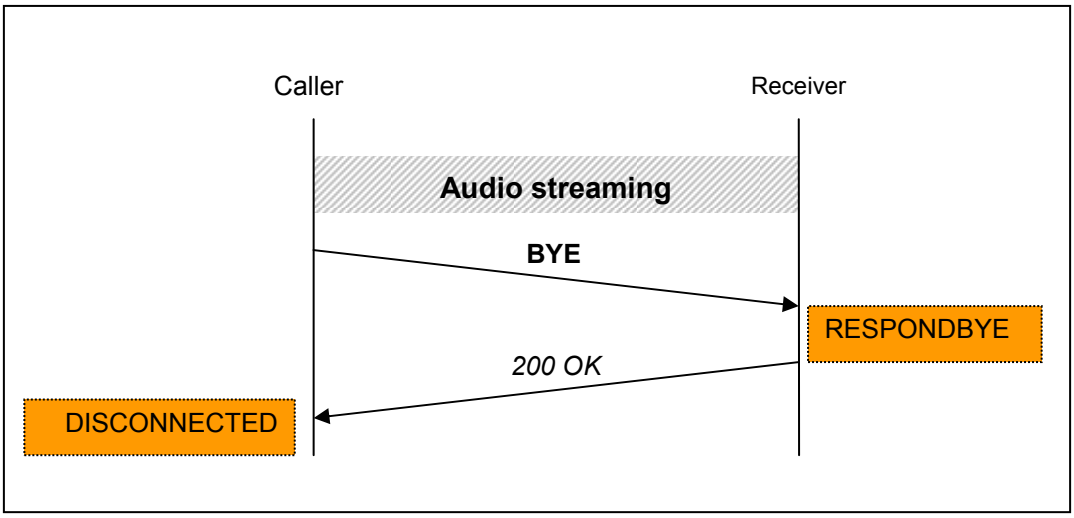


4.5.2 End of Communication (Normal State)

Normal communication ends according to the sequence illustrated in the above figure. A party wanting to end communication issues a BYE method. A party receiving the BYE returns **200 OK**. In the example illustrated in the above figure, a caller who initiated the conversation issues BYE; however, either party has the ability to issue BYE.

The request for ending communication cannot be denied. The session is deleted even if the request is ignored.

Figure 4-2 End of Communication

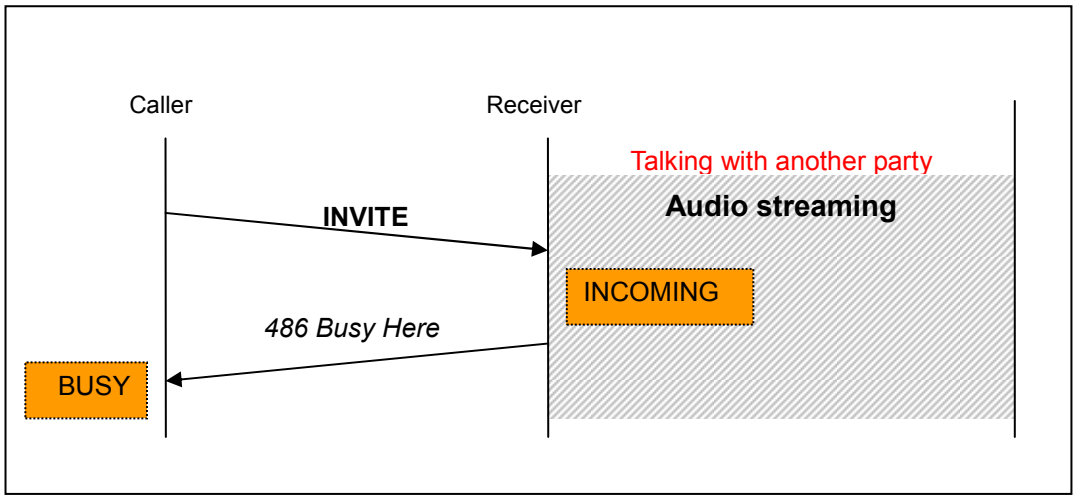


4.5.3 Busy

If a party is currently talking to another party and receives an INVITE request, the receiver indicates a busy status to the caller by returning **486 Busy Here** The receiving party can return **486 Busy Here** even though real conversation (establishing a session with another terminal) has not started.

If the receiving party of INVITE wants to accept the request instead of issuing **486 Busy Here**, the party must terminate the current conversation first, then return **200 OK**.

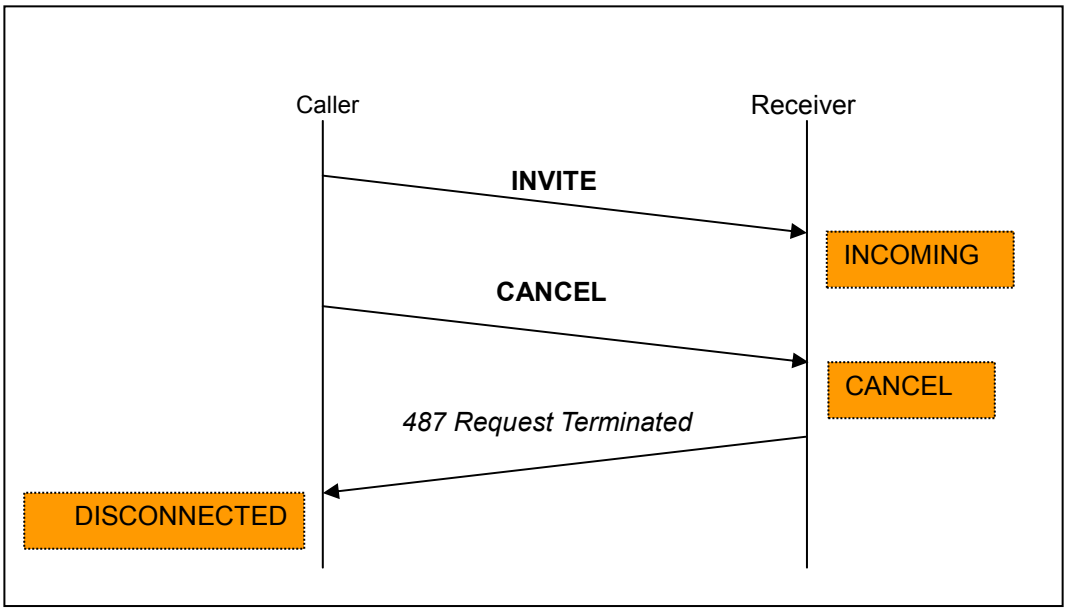
Figure 4-3 Busy



4.5.4 Cancellation of the Request to Talk

If a caller wants to cancel the request to talk before the session is established, the caller issues a CANCEL method. The receiving party, who received the CANCEL method, issues **487 Request Terminated** to accept the cancellation request.

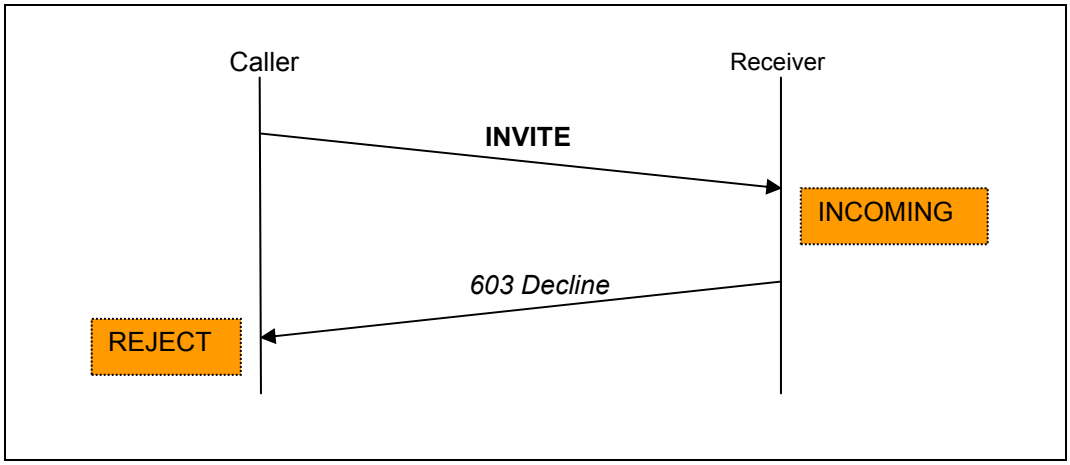
Figure 4-4 Cancellation of Request to Talk



4.5.5 Call Rejection

A receiving party returns **603 Decline** to the calling party when the receiving party wants to deny the request to talk.

Figure 4-5 Call Rejection



4.6 Sequence in Transceiver Mode

Transceiver Mode is used if three or more parties participate in the conversation. Transceiver Mode enables group conversation, but restricts talking to one party at a time (half-duplex operation).

4.6.1 Participation to Session and Notice

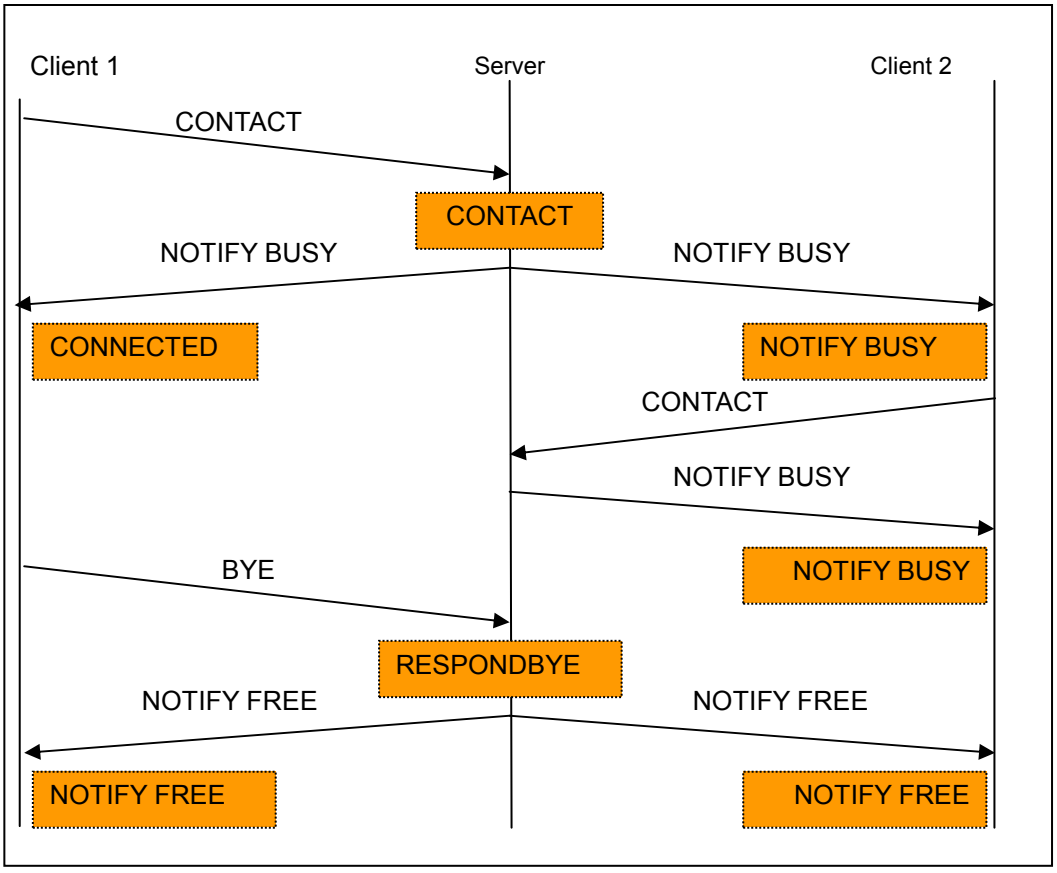
Transceiver Mode requires a grouping process for conversation with multiple parties. For this process, one machine must be assigned as a server. The game developer should assign the server because VoiceChat Library does not implement group management functionality due to game content consideration.

VoiceChat Library only handles requests for starting and ending conversation and is not involved in session and group management.

4.6.2 Sequence from Request to Talk Until Obtaining the Right to Talk

The sequence to obtain the right to talk is defined as follows:

Figure 4-6 Sequence in Transceiver Mode (Client Side)



Client 1 issues a CONTACT method to the server. The server transmits NOTIFY BUSY to all clients regardless of who is participating in the conversation. The server also adds the AID of the client who has the right to talk to the header of NOTIFY BUSY; the header of NOTIFY BUSY shows the right to talk.

A client receiving NOTIFY BUSY can determine if the client has the right to talk by looking at the AID. A CONNECTED event occurs for a client with the right to talk while a NOTIFY BUSY event occurs for the other clients. When the client with the right to talk transmits a voice signal, all participating clients receive the voice packet.

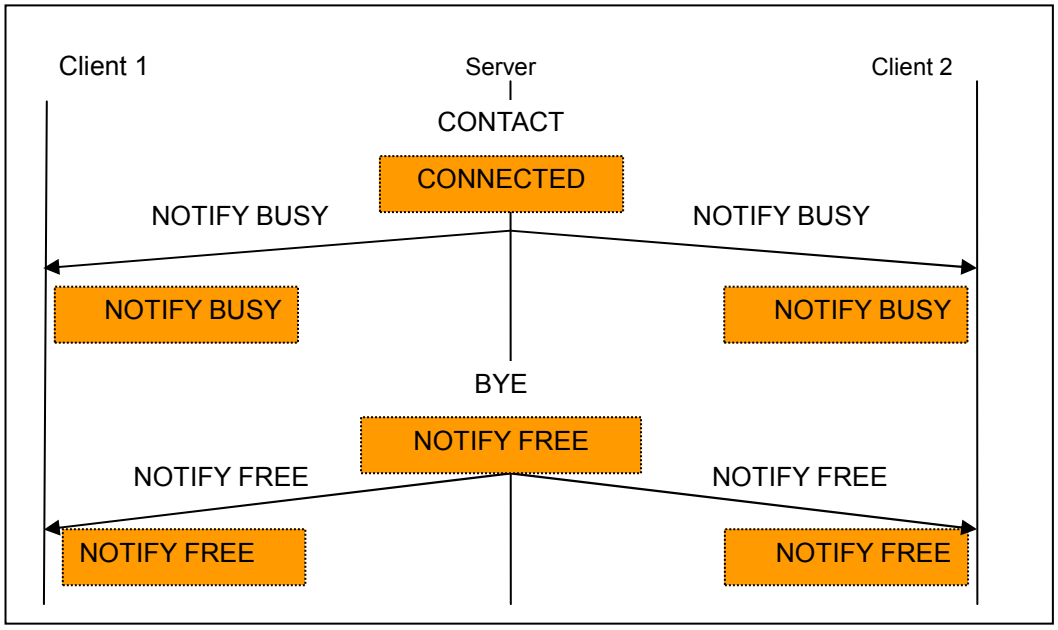
Even if Client 2 issues a CONTACT method afterwards, the server returns NOTIFY BUSY to Client 2 because Client 1 has begun conversation. However, the CONTACT request from Client 2 usually returns an error from the Library without an inquiry to the network server because the Library determines that another party is currently in a conversation. The server receives inquiries when CONTACT methods from both Client1 and Client 2 arrive at approximately the same time.

The server transmits NOTIFY FREE when Client 1 issues a BYE request or the conversation time in Transceiver Mode elapses a pre-defined period (30 seconds). In this example, a NOTIFY FREE event occurs in all client machines regardless of whether each has the right to talk.

4.6.3 Request to Talk (Server Side)

When a server issues CONTACT, the CONTACT request is not transmitted to the network. Instead, it processes inside the Library and generates the same event that the client issues. Therefore, common code can be used for event handler functions in both the server and the client.

Figure 4-7 Sequence in Transceiver Mode (Server Side)



4.6.4 NOTIFY Method

In the NOTIFY method, notice information is added to its extended information and message. The notice information content is defined as follows:

Table 4-5 NOTIFY Method

Code	Description
0x00	Conversation is enabled (NOTIFY_FREE)
0x01	Another party is in conversation (NOTIFY_BUSY)

The NOTIFY content allows participants to know whether CONTACT can be issued to initiate conversation. It also indicates the number of parties able to participate in the session as defined by the message body (the number of remote hosts receiving voice packet transmission).

4.7 Sequence in Conference Mode

Unlikely Telephone Mode or Transceiver Mode, Conference Mode does not require exchanging information between terminals because defining the right to talk is unnecessary. A Library user can immediately participate in the conference by calling API, specifying the terminals currently participating in the conference, and activating the voice stream engine. Conference Mode does not require an information exchange over SSP.

Library users initiate and manage the conference group according to the game, such as Transceiver Mode.

4.8 Status Machine

A status machine manages any abnormal sequence in VoiceChat Library. The different statuses supported by VoiceChat Library are defined as follows:

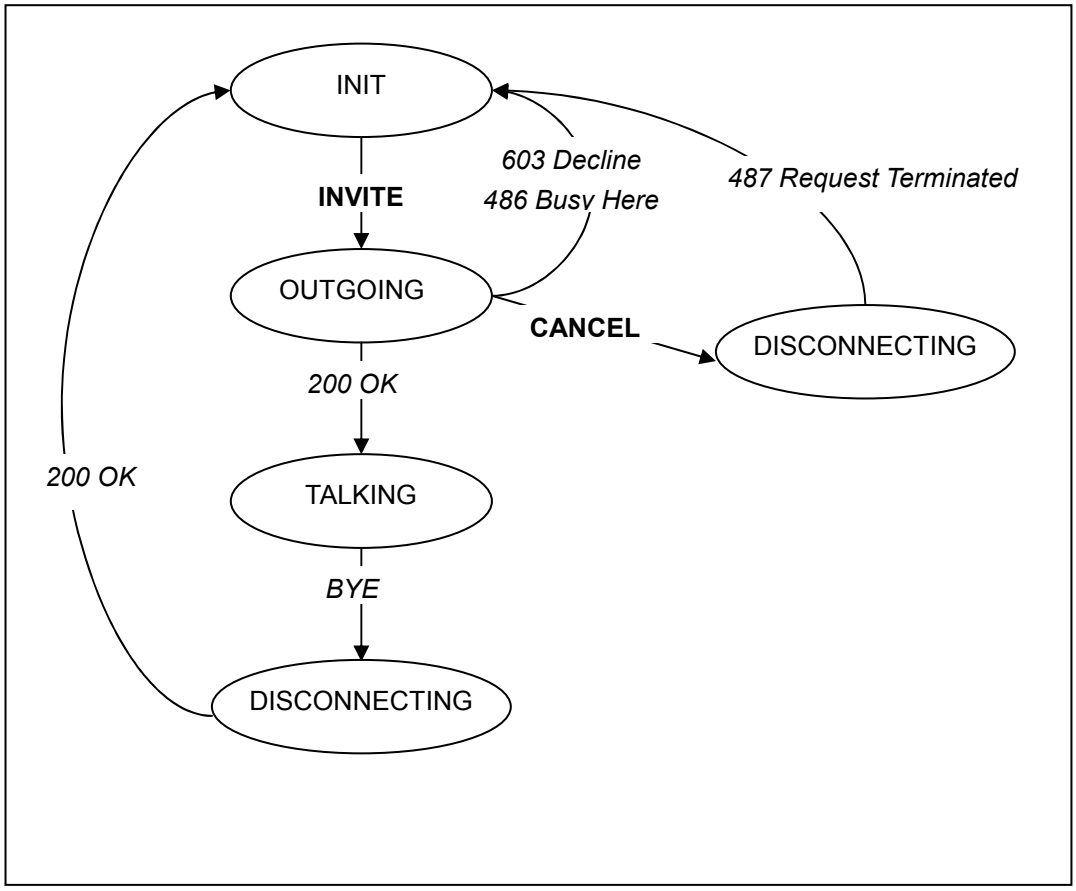
Table 4-6 Internal State of the Library

State Name	Description
INIT	Initial state
OUTGOING	Already issued INVITE/CONTACT
TALKING	Connected or in conversation
CONNECTED	Connected, another party is in conversation (Transceiver Mode only)
INCOMING	Received a request from outside
DISCONNECTING	Disconnecting

4.8.1 Status Transition in Telephone Mode

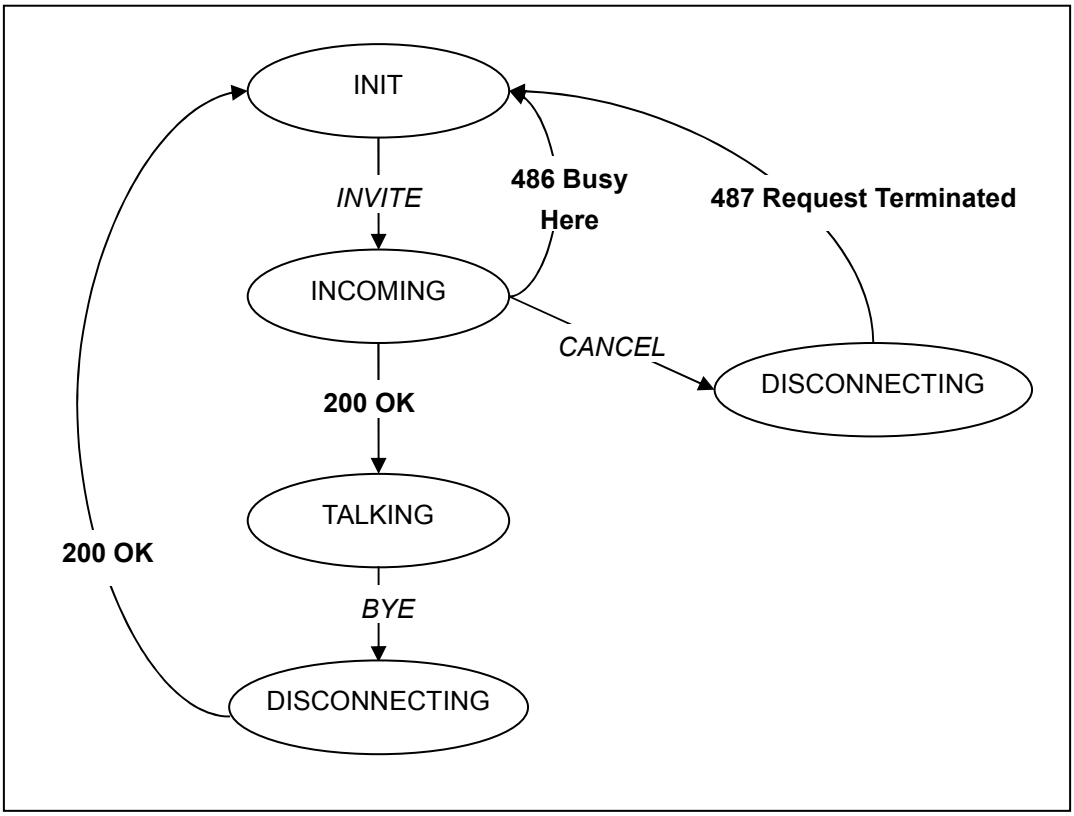
The following figure illustrates the status transition in Telephone Mode after INVITE is issued. Transmission is distinguished by bold font and reception is distinguished by italicized font.

Figure 4-8 Telephone Mode Status Transition (Transmitter Side)



The following figure shows the status transition upon receiving INVITE.

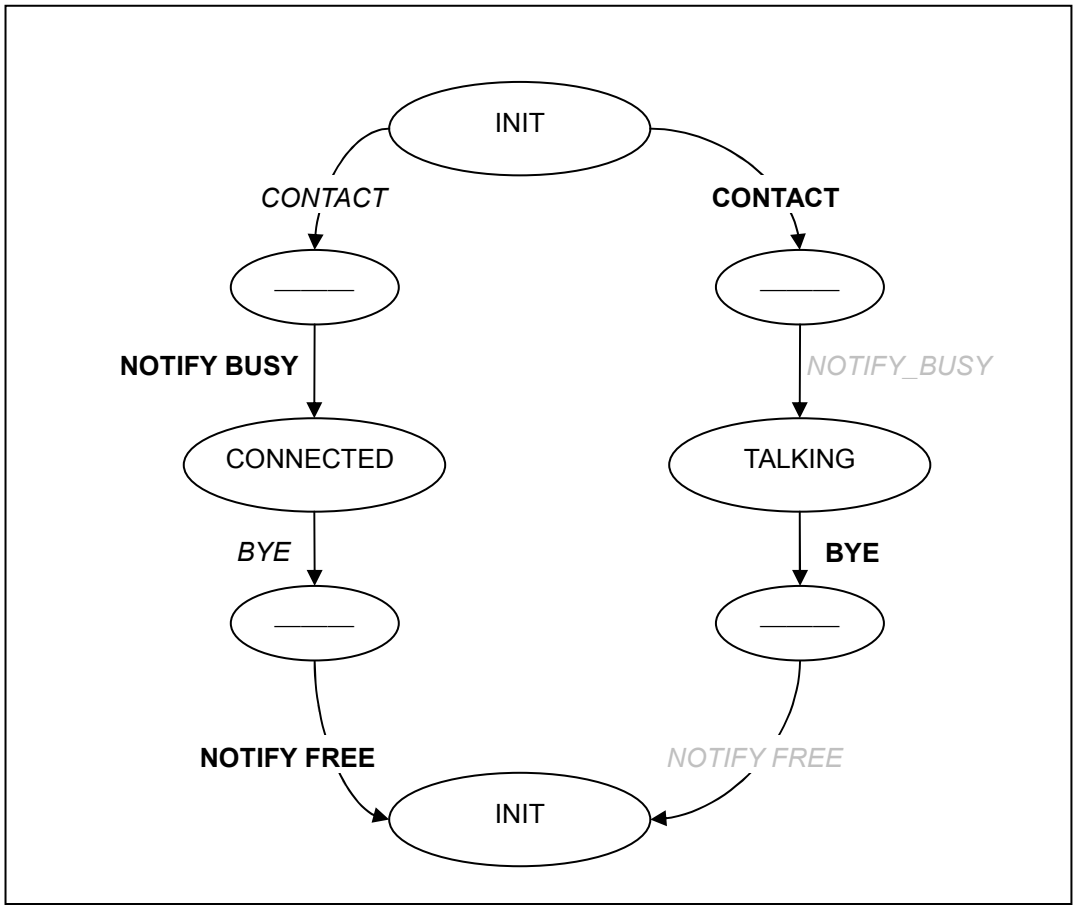
Figure 4-9 Telephone Mode Status Transition (Receiver Side)



4.8.2 Status Transition in Transceiver Mode

The following figure shows the status transition in a server machine in Transceiver Mode.

Figure 4-10 Telephone Mode Status Transition (Server Side)



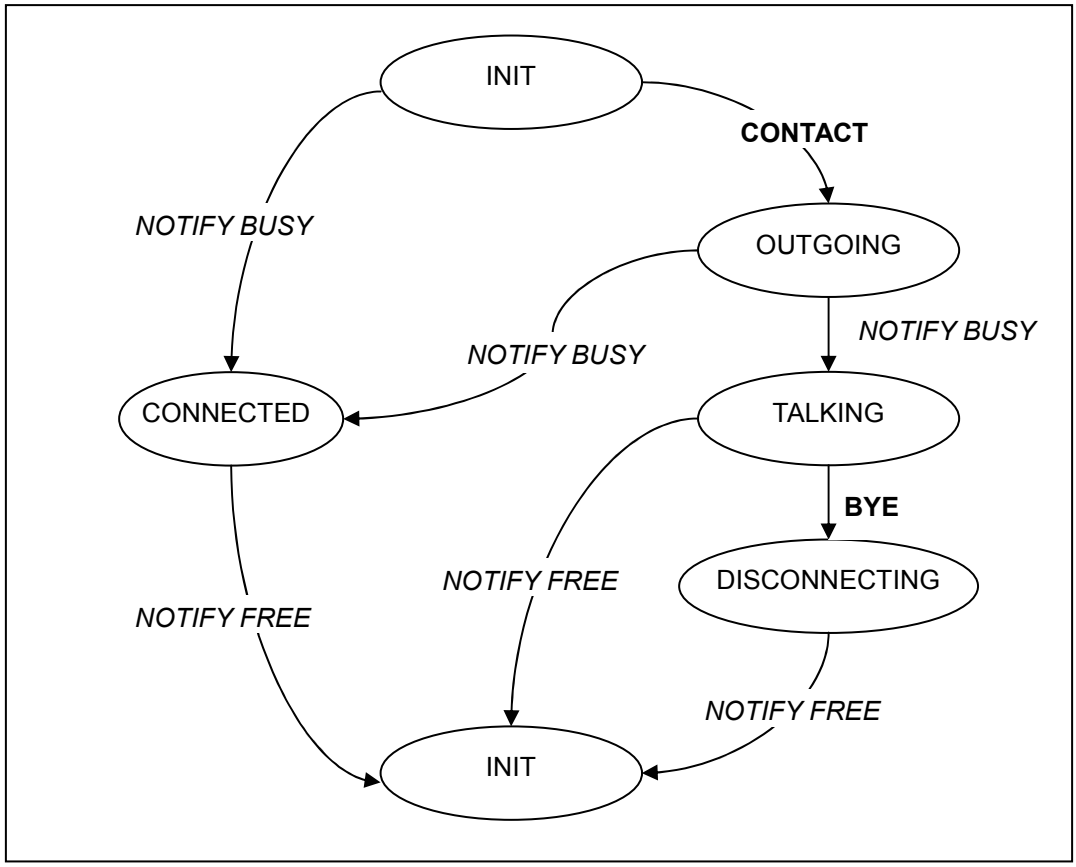
The Library automatically processes a request to talk in Transceiver Mode since only one terminal can talk at a time in this mode.

Upon receiving CONTACT from a client, the state automatically changes to CONNECTED. If the server is talking, the state changes to TALKING inside the Library. The figure appears to show the server as sending CONTACT, but this status change actually occurs within the Library instead of the transmitting packet.

The CONNECTED status indicates that talking is available. The TALKING status indicates that talking is currently taking place.

The client's status transition in Transceiver Mode is defined as follows:

Figure 4-11 Telephone Mode Status Transition (Client Side)



Although a client machine never receives a CONTACT request, the server sends NOTIFY to the client machine. The status changes to CONNECTED upon receiving NOTIFY BUSY; the status changes to INIT upon receiving NOTIFY FREE (illustrated in the left portion of the above figure).

If the client issues CONTACT to the server, the client moves to OUTGOING status before receiving NOTIFY BUSY from the server. NOTIFY BUSY shows the addresses of all clients capable of talking. Client statuses change to TALKING if the client starts talking; the statuses change to CONNECTED if another client has already talked. When this status is active, issuing BTY returns the status to INIT after receiving NOTIFY FREE (illustrated right portion of the above figure).

The talking client receives NOTIFY FREE even during TALKING. The conversation automatically terminates if it exceeds the 30-second time limit used in Transceiver Mode.

4.8.3 Status Transition in Conference Mode

Conference Mode only uses two internal statuses: INIT and TALKING.

4.9 Keep-Alive Processing and Time Out

No keep-alive processes exist to keep sessions in SSP. The game application should control connection-related processes.

By default, conversations have a 30-second time limit in Transceiver Mode.

5 Audio Streaming Process

The terminals involved with sessions established by SSP transfer audio data using a real-time protocol.

This real time protocol is similar to the RTP protocol, but does not conform to all RTP requirements in an effort to optimize processing speed, thus making it incompatible with RTP.

5.1 Overview

Anything spoken by the user is digitized using a 16-Bit / 8KHz sampling rate.

The VoiceChat Library has Voice Activity Detection (VAD) to prevent voice data packet transmission during mute. Packet transmission permissions can be set in the Library.

The audio frame length of each data packet transmitted is 68ms.

5.2 Audio Format

The VoiceChat Library can encode in the formats listed in the following table. The file format before encoding is 16-bit / 8KHz sampling data.

Table 5-1 Audio Format and Bit Rate

Format	Bit Rate	Sound Quality (In Order of Quality)
8bit / 8KHz Raw	64Kbps	Better (2)
G.711 u-Law	64Kbps	Best (1)
4bit ADPCM	32Kbps	Good (3)
3bit ADPCM	24Kbps	Fair (4)
2bit ADPCM	16Kbps	Bad (5)

The bit rates listed in the table are applicable strictly for one direction of a conversation. Bi-directional conversation in Telephone Mode doubles the bit rate used. The audio header and IP/UDP header are added during actual usage.

In Transceiver Mode, the bandwidth of a receiving terminal remains the same regardless of the number of clients participating in the session because only one terminal sends data at a time. However, bandwidth used by the transmitting terminal depends on the number of receivers and is defined as the sent bandwidth usage multiplied by the number of receivers. In Conference Mode, the number of participants must be taken into consideration when calculating bandwidth. Conference Mode calculations require duplicating the obtained bandwidth (due to two-way transmission) and multiplying that by the number of clients participating in a session. Due to bandwidth constraints, 8-bit / 8KHz Raw and G.711 u-Law formats cannot be used in Transceiver and Conference Modes.

The above bit rate is for audio data only. Actual data uses an additional 40 bytes for IP/UDP header and VoiceChat header. The following table describes the actual bandwidth used. IP data size is defined as the value remaining after IP header (20 bytes), UDP header (8 bytes), and VoiceChat header (12 bytes) are added to the Payload size.

Table 5-2 Audio Format and Bandwidth Used

Codec	Payload Size (Bytes)	IP Data Size (Bytes)	IP Data Size (Bits per Second)
bit Raw/G.711 u-law	544	584	68706
4bit ADPCM	276	316	37176
3bitADPCM	208	248	29176
2bit ADPCM	140	180	21176

The audio format can be changed while voice streaming. Format changes in Transceiver and Conference Modes should be made based on the number of parties participating to maintain the best balance of bandwidth usage and sound quality.

5.3 Adjustment of Voice Activity Detection

VoiceChat Library uses Voice Activity Detection (VAD) functionality to sense if talking or muting is currently taking place. Using VAD reduces CPU and network loads because packets are not transmitted onto the network when no talking activity (system on mute) occurs.

VAD's quick response stops packet transmission immediately after silence is detected. This can make the transmitted voice difficult to hear since small activities, such as a short breath, can stop packet transmission. VAD release time (default value of 1020ms (68ms * 15) can be configured to address this problem. This value can be changed by the `VCT_SetVADReleaseTime` function.

5.4 Adaptive Jitter Buffer

Generally, data is buffered once to absorb network jitter in the audio stream over an IP network. Buffering time is generally set to a fixed length (such as 50ms) without causing any negative impact to audio playback delay.

VoiceChat Library uses an algorithm called Adaptive Jitter Buffer (AJB) to ensure proper audio streaming, even under a jittering network environment.

All AJB processing occurs inside the Library to allow users usage of this feature without consideration of buffering status.

5.5 Audio Packet Details

Each packet's data format is defined by the figure below (horizontally 32-bit):

Figure 5-1 Audio Packet Format

Magic Token ('_VCT')						
V	P	X	PH	M	PT	Sequence number
Time stamp						
Payload data (140-544byte)						

Audio data is transmitted with the Magic Token (first four bytes). This is added to the top as an SSP packet because the audio data also transfers using the NITRO-DWC API.

6 About API

This chapter describes how the Library is used in practical situations. Setting PATH in the NITROLIBVCT_ROOT environment variable is required before using the Library. The following common building tools are necessary for the makefile used to create application libraries:

- include $(NITROLIBVCT_ROOT)/build/buildtools/commondefs
- include $(NITROLIBVCT_ROOT)/build/buildtools/modulerules

Additional NITRO-WiFi and NITRO-DWC building tools are included upon inclusion of the primary building tools.

6.1 Initializing and Terminating Library

VoiceChat Library must be initialized using the VCT_Init function before any Library function can be used. This function must be called every time the mode changes.

During initialization, both the mode and the receiving buffer used for audio streaming must be specified. Configuring the VCTConfig Structure specifies the operating mode, audio stream receiving buffer, and other functions as detailed in the table below:

Table 6-1 VCTConfig Structure

Name of Variable	Operation
session	Specify area where the VCTSession Structure is saved
numSession	Specify size of area specified by session
mode	Specify operating mode of VoiceChat
aid	Specify AID of terminals
audioBuffer	Specify receiving buffer for audio stream (must be 4 byte alignment)
audioBufferSize	Specify size of buffer specified by audioBuffer
callback	Specify callback function upon event occurrence
userData	Specify user data to be passed upon call of Callback function

An example of Telephone Mode initialization is shown below:

Code 6-1 Telephone Mode Initialization Sample

```
static VCTSession sSession[2];   /* location to save VCTSession Structure */
static u8         sBuffer[8 * VCT_AUDIO_BUFFER_SIZE]; /* receiving buffer for
audio */

VCTConfig config;
config.session               = sSession;
config.numSession            = 2;
config.mode                  = VCT_MODE_PHONE;
config.aid                   = DWC_GetMyAID();
config.audioBuffer           = stBuffer;
config.audioBufferSize       = VCT_AUDIO_BUFFER_SIZE * 8;
config.callback              = VoiceChatEventCallback;
config.userData              = myData;

VCT_Init(&config)
```

In the above sample, two sessions are reserved for responding to requests such as an interruption request. In the event that only one session is reserved, VoiceChat Library automatically returns BUSY upon receiving an interrupt request. Transceiver Mode requires setting a value of one for the necessary number of sessions. In Conference Mode, the necessary number of sessions is determined by subtracting one from the maximum number of participants (Max. Participants – 1).

After specifying VoiceChat mode and own host AID, the audio stream receiving buffer size must be specified. At least 100ms must be reserved for the audio stream buffer to avoid only receiving voice data fragments. Because of this, a buffer size of more than 250ms is recommended. The above sample code reserves 512ms (8 x 64). In Conference Mode, the number of required buffers must increase to match the number of clients participating in the conference. For instance, if eight buffers are provided for each client and four parties participate in the conference, a total of 24 buffers (8 x 3) are required.

Increasing the number of buffers does not cause any delay in audio playback. The dynamic jitter buffer dynamically determines the actual number of buffers by measuring jitters on the network.

The audio steaming buffer pointer must be aligned to four bytes, and `audioBufferSize` must be an integral multiple of `VCT_AUDIO_BUFFER_SIZE`. In addition, the Callback function and its respective parameters must be specified.

Upon finishing use of VoiceChat Library, the `VCT_Cleanup` function must be called.

The library must be finalized using the `VCT_Cleanup` function before calling `VCT_Init`, even if the mode is being changed.

6.2 Handling VoiceChat Data

VoiceChat Library transfers audio and SSP data using the NITRO-DWC API. The data receiving function of NITRO-DWC (the function specified by the `DWC_SetUserRecvCallback` function) must be implemented to handle both game data and VoiceChat data.

The `VCT_HandleData` function looks at the first four bytes of the received data to determine if it is VoiceChat data. If the first four bytes are the Magic Token of VoiceChat, the function assumes the data is VoiceChat-related. Any game data must not have the same first four bytes as VoiceChat's Magic Token. See Section 4.2: Format, and Section 5.5:

Audio Packet for more information.

An example for using the function is shown below.

Code 6-2 Handing Received Data

```
/* DWC callback function for receiving specified by DWC_SetUserRecvCallback
function */
void UserRecvCallback(u8 aid, u8* buffer, int size)
{
    BOOL flag;

    flag = VCT_HandleData(aid, buffer, size);
    if (flag == FALSE) {
        /* If return value is FALSE, receiving data are not for VoiceChat or
VoiceChat Library is not initialized. Implement here process necessary for the
game. */
    }
}
```

If the return value is TRUE, the received data is VoiceChat Library data. Therefore, the necessary processes for the `VCT_HandleData` function and the Callback function are called.

In addition, `VCT_Main` must be called at every picture frame (1/60 sec) to confirm information such as the audio encoding, transmitting process, and time-out.

Code 6-3 Handing Receiving Data

```
/* Main function of game */
void GameMain()
{

    while (1) {
       OS_WaitIrq ( TRUE, OS_IE_V_BLANK );
       VCT_Main();
       /*
          Other processes…
       */
    }
}
```

6.3 SSP Event Process

Sessions in Telephone and Transceiver Modes with other players must be established using the SSP functions. This section describes processes beginning from issuing a request to talk to closing of sessions in Telephone Mode. For details about the process sequence, see Section 4.5: Sequence in Telephone Mode.

When a terminal issues a request to start talking, a session is initially created using `VCT_CreateSession`. After creating the session, the terminal transmits the request to talk to a receiver using the `VCT_Request` function.

Code 6-4 Sample of Request to Talk

```
VCTSession *session = VCT_CreateSession ( aid );
VCT_Request ( session, VCT_REQUEST_INVITE );
```

When the receiver receives the event, the callback function specified by `VCT_Init` is called via `VCT_HandleData`. The sSession static variable prevents processing of multiple sessions.

If the session is new, `VCT_RESPONSE_OK (200 OK)` is returned using the `VCT_Response` function before starting an audio stream using `VCT_StartStreaming`.

Code 6-5 Processing of INCOMING Event

```
static *sSession = NULL;

/* Callback function specified by VCT_Init */
static void VoiceChatEventCallback(u8 aid,
                                   VCTEvent event,
                                   VCTSession *session,
                                   void *data)
{
    switch (event) {
    case VCT_EVENT_INCOMING:
        if (sSession) {
            ret = VCT_Response(session, VCT_RESPONSE_BUSY_HERE);
            VCT_DeleteSession(session);
            break;
        }
        sSession = session;
        VCT_Response (session, VCT_RESPONSE_OK );
        VCT_StartStreaming ( session );
        break;
    }
}
```

As the receiver returns `VCT_RESPONSE_OK (200 OK)`, a CONNECTED event occurs in the transmitting terminal.

Code 6-6 Processing CONNECTED Event

```
static void VoiceChatEventCallback(u8 aid,
                                   VCTEvent event,
                                   VCTSession *session,
                                   void *data)
{
    switch (event) {
    case VCT_EVENT_CONNECTED:
        VCT_StartStreaming ( session );
        break;
    }
}
```

In this sample, the response takes place immediately after the event occurs. Operations designed to motivate user acceptance of the request, such as a unique ring tone or displaying an animation upon the occurrence of an INCOMING event, can be used here.

For such operations, a session must be saved, and the status of the `VCTSession` Structure state member upon the receiver's response is checked. If the session state is INCOMING, `VCT_RESPONSE_OK (200 OK)` is returned.

Code 6-7 Processing of INCOMING Event

```
static *sSession = NULL;

static void VoiceChatEventCallback(u8 aid,
                                   VCTEvent event,
                                   VCTSession *session,
                                   void *data)
{
    switch (event) {
    case VCT_EVENT_INCOMING:
        if (sSession) {
            ret = VCT_Response(session, VCT_RESPONSE_BUSY_HERE);
            VCT_DeleteSession(session);
            break;
        }
        sSession = session;
        VCT_Response (session, VCT_RESPONSE_OK );
        VCT_StartStreaming ( session );
        break;
    }
}

/* Function when A button is pushed */
void onAButtonDown()
{
    if (sSession->state == VCT_STATE_INCOMING) {
        VCT_Response( sSession, VCT_RESPONSE_OK );
        VCT_StartStreaming (sSession );
    }
}
```

In Transceiver Mode, the VCT_Contact and VCT_Release functions are used to send various requests such as CONTACT or BYE. The VCT_Request and VCT_Response functions cannot be used for sending requests in Transceiver Mode.

6.4 Handling Audio Data

To minimize delay time and smoothly process audio record/playback and transmission/receipt, audio data should be processed at a constant interval using the appropriate interrupt process.

This section describes the handling of audio data while using NitroSystem streaming.

Sound is initialized using MIC_AutoSampling and NNS_SndStrmSetup. The sound format should be 16-bit with code, and the sample rate should be 8KHz. The digitalization parameter of a microphone (mic) is set to MIC_SAMPLING_TYPE_SIGNED_12BIT. In addition, the sampling callback call interval should be set to 68ms (the packet size of VoiceChat Library).

The sampling rate for the mic should be set to (NNS_SND_STRM_TIMER_CLOCK / 8000) * 64 instead of using MICSamplingRate specified by NitroSDK. This ensures that the mic's audio signal samples at a rate synchronized with the stream's playback clock.

Code 6-8 Initialization of Audio Processing

```
/* Parameters related to audio */
#define SAMPLING_RATE       VCT_AUDIO_FRAME_RATE     // 8kHz
#define SAMPLING_TIME       VCT_AUDIO_FRAME_LENGTH   // 68ms
#define SAMPLING_BYTE       2                        // 16bit

/* Reserve buffer for NNS_SndStrm */
#define WAVE_SAMPLES ((int)(SAMPLING_RATE * SAMPLING_TIME * SAMPLING_BYTE) /
1000)

static u8  sPlayBuffer[WAVE_SAMPLES * 2] ATTRIBUTE_ALIGN(32);
static u8   sRecBuffer[WAVE_SAMPLES * 2] ATTRIBUTE_ALIGN(32);

static struct NNSSndStrm sSndStream;

/*
   Initialization of sound stream and mic
*/
void InitSound()
{
    MICAutoParam micParam;
    u32          length;
    u8           cArray[1] = {7};    /* Use 7 channels */

    MIC_Init();
    PM_Init();
    NNS_SndInit();
    NNS_SndStrmInit( &sSndStream );
    NNS_SndStrmAllocChannel(&sSndStream, 1, &cArray);

    length = (u32)(SAMPLING_RATE * SAMPLING_TIME * SAMPLING_BYTE) / 1000;

    /* Start of automatic sampling for Mic */
    micParam.type             = MIC_SAMPLING_TYPE_SIGNED_12BIT;
    micParam.buffer    = sRecBuffer;
    micParam.size             = length * 2;
    micParam.rate             = (u32)((NNS_SND_STRM_TIMER_CLOCK / SAMPLING_RATE)
* 64;
    micParam.loop_enable              = TRUE;
    micParam.full_callback    = NULL;
    micParam.full_arg         = NULL;
    MIC_StartAutoSampling( &micParam );

    /* Start of stream playback */
    NNS_SndStrmSetup(
        &sSndStream,
        NNS_SND_STRM_FORMAT_PCM16,
        sPlayBuffer,
        length * 2,
        NNS_SND_STRM_TIMER_CLOCK / SAMPLING_RATE,
        2,
        SndCallback,
        sRecBuffer);
}
```

VCT functions can search for the mic's current sampling position and transfer (transmit/receive) audio data.

Code 6-9 Processing of Sound Callback Functions

```
/*
   Sound callback functions
*/

static void SndCallback(
       NNSSndStrmCallbackStatus sts,
       int nChannels,
         void* buffer[],
       u32 length,
       NNSSndStrmFormat format,
       void* arg)
{
    const void *micAddr;
    u8*         *micSrc;
    u32         offset;
    u32         ch;

    /* When sts is NNS_SND_STRM_CALLBACK_SETUP, buffer must be cleared */
    if ( sts == NNS_SND_STRM_CALLBACK_SETUP )
    {
        for ( ch = 0; ch < nChannels; ++ch )
        {
            MI_CpuClearFast( buffer[ch], length );
        }
        return;
    }

    /* Search for Mic's current sampling position */
    micSrc   = (u8*)arg;
    micAddr = MIC_GetLastSamplingAddress();
    offset   = (u32)((u8*)micAddr - micSrc);
    if ( offset < length )
    {
        micSrc = micSrc + length;
    }

    /* Transmitting and receiving audio data */
    VCT_SendAudio( micSrc, length )

    for ( ch = 0; ch < nChannels; ++ch)
    {
        VCT_ReceiveAudio( buffer[ch], length, ch, NULL );
    }
}
```

Processes such as encoding/decoding audio and transferring packets must not occur inside `VCT_SendAudio`/`VCT_ReceiveAudio` functions; these functions are designed to occur inside interrupt functions. The `VCT_Main` function encodes audio and transmits processes, while `VCT_HandleData` receives and decodes processes.

6.5 Audio Recording/playback and BGM Playback Control

The return value of VCT_ReceiveAudio indicates whether any audio data exists for playback testing. VCT_ReceiveAudio returns TRUE if audio exists in the received data (the buffer is filled with audio data) and returns FALSE if no audio exists (the buffer is cleared). Using this functionality, operations, such as muting BGM or lowering the volume when a player talks, can be programmed.

VCT_SendAudio returns TRUE when audio data is in the queue for the transmitting buffer. This is similar to when VAD detects audio (when VAD is enabled by VCT_Enable).

This can be used for different purposes, such as preventing the transmission of unnecessary sounds during conversations.

The following sample shows a modified version of Code 6-9 that sets the BGM to mute.

Code 6-10 Mute BGM While Transferring Voice

```
BOOL needMute = FALSE;

for ( ch = 0; ch < nChannels; ++ch)
{
    if (VCT_ReceiveAudio( buffer[ch], length, ch, NULL )) {
        needMute = TRUE;
    }
}
if (VCT_SendAudio (micSrc, length)) {
    needMute = TRUE;
}
if (needMute) {
     /* BGM on mute here */
}
```

If VAD is OFF, VCT_SendAudio always returns TRUE unless any error occurs in the process of queuing audio data to the transmitting buffer. Setting VAD to OFF to send all audio packets, while also muting the BGM whenever any game player talks, can be done by using VCT_GetVADStatus as follows:

Code 6-11 VAD Self-Judgment

```
BOOL needMute = FALSE;
VCTVADStatus sts = VCT_VAD_NONE;

VCT_EnableVAD ( FALSE );

sts = VCT_GetVADStatus ( micSrc, length, NULL );
if ( sts == VCT_VAD_DETECT || sts == VCT_VAD_ACTIVE) {
       /* BGM on mute here */
}
/* Audio packet is always transmitted */
VCT_SendAudio ( micSrc, length );
```

VoiceChat Library controls the packet arrival interval by transmitting VAD information while simultaneously transmitting audio packets. This means that code should never be written to control the VCT_SendAudio function calling after VAD self-judgement. The VCT_ReceiveAudio/VCT_SendAudio functions must always be called after streaming.

Code 6-12 Bad Code Sample

```
VCT_EnableVAD ( FALSE );

sts = VCT_GetVADStatus ( micSrc, length, NULL );
if ( sts == VCT_VAD_DETECT || sts == VCT_VAD_ACTIVE) {
       VCT_SendAudio (micSrc, length );
}
```

7 Demo Programs

7.1 Program Initiation to Matchmaking Completion

This demonstration program shows an example process from program initiation to the completion of matchmaking and is similar to the procedure for `dwc_match` in the NITRO-DWC demonstration program.

After running the program, select Login to connect to the access point and log on to an available Wi-Fi connection. Before starting the program, the access point should be registered to connect and create a Wi-Fi Connection.

After logging in, matchmaking can be processed through any type of connection. When the matchmaking process completes, the following screen appears:

Figure 7-1 Post-Matchmaking Screen

```
========================================================
GAME CONNECTED MODE
> Start Phone
  Start Transceiver
  Start Conference
  Close Connections
  Close All Hard
========================================================
```

The following table details the Game-Connected Mode menu options.

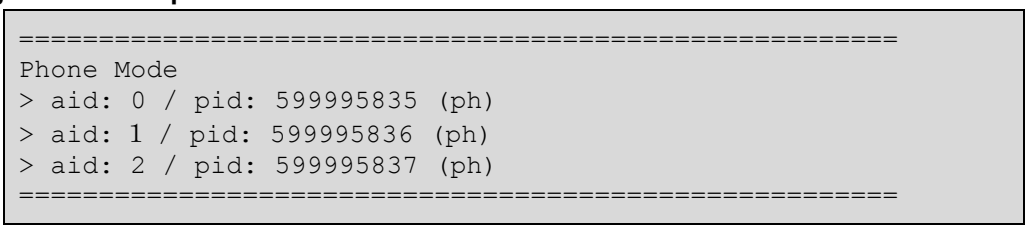
Table 7-1 GAME-CONNECTED MODE Menu

Menu	Description
Start Phone	Start of Telephone Mode
Start Transceiver	Start of Transceiver Mode
Start Conference	Start of Conference Mode
Close Connections	Close of connection
Close All Hard	Force termination of connection

7.2 Telephone Mode Test Program

After selecting the Start Phone option in the GAME CONNECTED mode, the following screen displays to execute the Telephone Mode demonstration program.

Figure 7-2 Telephone Mode



The screen displays a list of clients for which matchmaking successfully completed. In addition to the AID and PID, the following table describes the other client information that can be displayed:

Table 7-2 Client Information

Display	Description
(x)	No mode determined
(ph)	Telephone Mode
(tr)	Transceiver Mode
(talk)	In conversation

The X button selects the client with which to talk; pushing the A button calls this partner. The receiving partner can push the A button to accept or the B button to reject. If the B button is pushed while calling the other partner, it cancels the call.

The following table describes the operation of other buttons:

Table 7-3 Key Operation in Telephone Mode

Button	Operation
Cursor	Move the cursor to up/down and right/left
A	While idling: make a call While receiving: respond to the call
B	While receiving: deny the request to talk While transmitting: cancel the call On conversation: hung up
X/Y	Change audio format
Select	Change menu
R	Transmit rectangle wave

Pushing the Select button changes the display contents of the lower screen. Menus other than AudioInfo cannot control the upper screen because the buttons are used for lower screen menu operation. The demonstration program for Telephone Mode has the following menu:

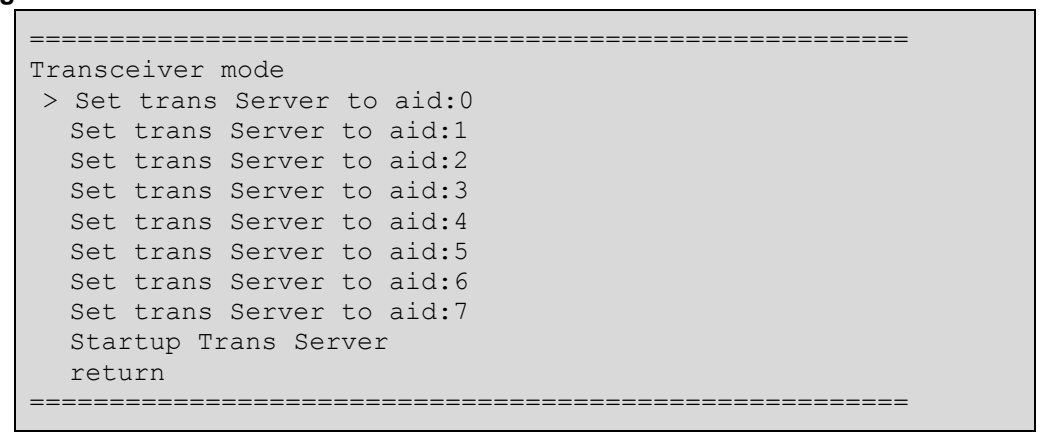
Table 7-4 Sub-Menu

Display	Description
AudioInfo	Display of information of audio streaming
VAD Info	Display of information of VAD and change of VAD settings
Audio Debug	Menu for debugging audio streaming (Debug Build version only)

7.3 Transceiver Mode Test Program

Selecting Start Transceiver in GAME CONNECTED MODE executes the Transceiver Mode demonstration and displays the following screen:

Figure 7-3 Transceiver Mode



In Transceiver Mode, one of the participating machines must operate as a server. Select Startup Trans Server on the server machine, and select the AID of the server machine on the client machines.

Upon setting the server and client machines, pushing the L button initiates conversation.

Other operations are performed using the same methods as operations in Telephone Mode. Transceiver Mode cannot use 8-bit 8KHz Raw and G.711 u-Law formats.

7.4 Conference Mode Test Program

Selecting Start Conference in GAME CONNECTED MODE executes the Conference Mode demonstration and displays the following screen:

Figure 7-4 Conference Mode

Select client AID using the X button and push the A button to add/delete the client to/from the conference. This operation must be made for each client participating in the conference. Upon being added to the conference, any client can participate in the conversation without any further operation.

Other operations are performed using the same methods as operations in Telephone Mode. The X/Y buttons can manually change the audio format, but the program automatically sets the audio format to 4-bit ADPCM (for conferences with up to three participants) or 3-bit ADPCM (for conferences with up to four participants). Conference Mode cannot use 8-bit 8KHz Raw and G.711 u-Law formats.

8 Additional Notes

8.1 Calling the DWC Function

VoiceChat Library calls the NITRO-DWC's transmitting function within the following functions. Extra care is critical when using the DWC function with multiple threads in an application.

- `VCT_Main` / `VCT_HandleData`
- `VCT_Request` / `VCT_Response` / `VCT_Contact` / `VCT_Release`

8.2 Usage of Memory

The VoiceChat Library does not internally call functions to reserve memory. Specifying the buffer and its respective size during VoiceChat Library initialization controls the size of the memory used. Specifying the buffer and its respective size during VoiceChat Library initialization controls the size of the memory used. The Library has a maximum limit of 72 packets (1144 bytes per packet with a total size of approximately 80KB).

The library occupies 5KB of static memory size. The codes in the Library's resident memory use 20KB. Both values are based on a final ROM build.