

# VoiceChat ライブラリ 解説

Ver. 1.3.1

任天堂株式会社発行

このドキュメントの内容は、機密情報であるため、  
厳重な取り扱い、管理を行ってください。

## 目次

1	はじめに .....	6
2	ライブラリの利用シーン .....	7
2.1	1対1の会話 .....	7
2.2	複数人による会話 .....	8
2.2.1	トランシーバーモードによる会話 .....	8
2.2.2	カンファレンスモードによる会話 .....	8
3	ライブラリ概要 .....	9
4	会話セッションの管理 .....	9
4.1	シンプル・セッション・プロトコル .....	9
4.2	フォーマット .....	9
4.3	リクエストメソッドとレスポンスコード .....	9
4.4	イベントコード .....	9
4.5	電話モードにおけるシーケンス .....	9
4.5.1	会話要求 .....	9
4.5.2	会話終了(通常終了) .....	9
4.5.3	話中 .....	9
4.5.4	会話要求中止 .....	9
4.5.5	受信拒否 .....	9
4.6	トランシーバーモードにおけるシーケンス .....	9
4.6.1	トランシーバーセッションへの参加と通知 .....	9
4.6.2	参加要求～終了までのシーケンス .....	9
4.6.3	会話要求(サーバ側) .....	9
4.6.4	NOTIFY メソッドの内容 .....	9
4.7	カンファレンスモードにおけるシーケンス .....	9
4.8	ステートマシン .....	9
4.8.1	電話モードにおける状態遷移 .....	9
4.8.2	トランシーバーモードにおける状態遷移 .....	9
4.8.3	カンファレンスモードにおける状態遷移 .....	9
4.9	キープアライブとタイムアウト .....	9
5	音声ストリーミング処理 .....	9
5.1	概要 .....	9
5.2	音声フォーマット .....	9
5.3	VAD (Voice Activity Detection)の調整 .....	9
5.4	エコーキャンセル .....	9
5.5	適応型ジッタバッファ .....	9
5.6	音声パケットの詳細 .....	9

6	API の解説.....	9
6.1	ライブラリの初期化と終了 .....	9
6.2	VoiceChat データのハンドリング .....	9
6.3	SSP イベント処理 .....	9
6.4	音声データのハンドリング .....	9
6.5	音声の録音・再生とBGM 再生の抑制.....	9
7	デモプログラムの解説.....	9
7.1	プログラムの起動からマッチメイクの完了まで .....	9
7.2	電話モードのテストプログラム.....	9
7.3	トランシーバーモードのテストプログラム.....	9
7.4	カンファレンスモードのテストプログラム.....	9
8	特記事項.....	9
8.1	DWC 関数の呼び出し.....	9
8.2	メモリ使用量.....	9

## コード

コード 6-1	電話モードでの初期化例 .....	9
コード 6-2	受信データのハンドリング .....	9
コード 6-3	VoiceChat メインルーチンの実行 .....	9
コード 6-4	会話要求の例.....	9
コード 6-5	INCOMING イベントの処理.....	9
コード 6-6	CONNECTED イベントの処理 .....	9
コード 6-7	INCOMING イベントの処理.....	9
コード 6-8	サウンド処理の初期化 .....	9
コード 6-9	サウンドコールバック関数の処理.....	9
コード 6-10	音声送受信時に BGM をミュートする .....	9

## 表

表 4-1	SSP フォーマット.....	9
表 4-2	SSP リクエストメソッドコード .....	9
表 4-3	SSP レスポンスコード .....	9
表 4-4	イベントコード.....	9
表 4-5	NOTIFY メソッドの内容.....	9
表 4-6	ライブラリの内部状態.....	9
表 5-1	音声フォーマットとビットレートの関係 .....	9
表 5-2	音声フォーマットと帯域消費量の関係 .....	9
表 6-1	VCTConfig 構造体 .....	9
表 7-1	GAME CONNECTED MODE のメニューの内容 .....	9
表 7-2	クライアントの情報.....	9
表 7-3	電話モードのキー操作 .....	9
表 7-4	サブメニューの内容 .....	9



図 2-1 1対1の会話 .....	7
図 2-2 トランシーバーモードによる会話 .....	8
図 2-3 カンファレンスモードによる会話 .....	9
図 3-1 ライブラリ概要図 .....	9
図 4-1 会話要求 .....	9
図 4-2 会話終了 .....	9
図 4-3 話中 .....	9
図 4-4 会話要求の中止 .....	9
図 4-5 受信拒否 .....	9
図 4-6 トランシーバーモードのシーケンス(クライアント側) .....	9
図 4-7 トランシーバーモードのシーケンス(サーバ側) .....	9
図 4-8 電話モードにおける状態遷移図(送信側) .....	9
図 4-9 電話モードにおける状態遷移図(受信側) .....	9
図 4-10 トランシーバーモードにおける状態遷移図(サーバ側) .....	9
図 4-11 トランシーバーモードにおける状態遷移図(クライアント側) .....	9
図 5-1 音声パケットの内容 .....	9
図 7-1 マッチメイク後の画面 .....	9
図 7-2 電話モード .....	9
図 7-3 トランシーバーモード .....	9
図 7-4 カンファレンスモード .....	9

## 改訂履歴

版	改訂日	改訂内容
1.3.1	2008-04-01	VCTConfig.audioBufferのバイトアライメント記述、およびサウンドコールバックのサンプルコードを修正
1.3.0	2008-02-04	Nitroライブラリバージョンアップに伴うバージョンアップ
1.2.0	2007-09-18	記述不正の修正
1.2.0	2007-01-15	Nitroライブラリバージョンアップに伴うバージョンアップ
1.1.1	2006-11-15	サウンドコールバックのサンプルコード内の不要なforループを削除
1.1.0	2006-09-01	エコーキャンセルに関する記述を追記
1.0.0	2006-06-01	初版作成

# 1 はじめに

VoiceChat ライブラリは、TCP/IP ネットワークを用いて音声通話をするために必要なライブラリー式であり、音声通話のためのライブラリと、着信・受信を制御するライブラリを含んでいます。

本ドキュメントは、ライブラリの基本的な構造と用法について解説するものです。

本ライブラリには音声通話のために、特別なサーバハードウェア・ソフトウェアは必要ありません。ただし通信には NitroWiFi / NitroDWC を利用する必要があります。

version 1.1.0 からエコーキャンセル機能が搭載されています。エコーキャンセルを使用するためには、NitroSystem のサウンド機能を使用することが前提となります。独自のサウンドシステムを使用されている場合、エコーキャンセル機能が使用できないため、トランシーバーモードをお使いください。（電話モード、カンファレンスモードではハウリング等の問題が発生する可能性があります。）

なお、本ライブラリによって可能となる会話は1対1（電話モード）、1 対複数（トランシーバーモード）、および複数人同時（カンファレンスモード）の会話です。

## 2 ライブラリの利用シーン

ゲーム内で利用する際の利用シーン(Use Case)として、VoiceChat ライブラリでは以下のケースを想定しています。

- 1対1の会話(電話モード)
- 複数人による会話(トランシーバーモード)
- 複数人による会話(カンファレンスモード)

### 2.1 1対1の会話

1対1の会話では、通常の電話のように相手呼び出し、呼び出された側がそれに応答する形で会話が始まります。このモードを VoiceChat ライブラリでは「電話モード」と呼びます。

ライブラリでは話中応答や割り込み通話をサポートしていますが、会話自体は1対1であり、複数人による会話は出来ません。

電話モードはリアルタイム性が低いロールプレイングゲームのような謎を解くタイプのゲームにおいて、他のプレイヤーとゲーム内容について会話する、というような場合に利用されることを想定しています。

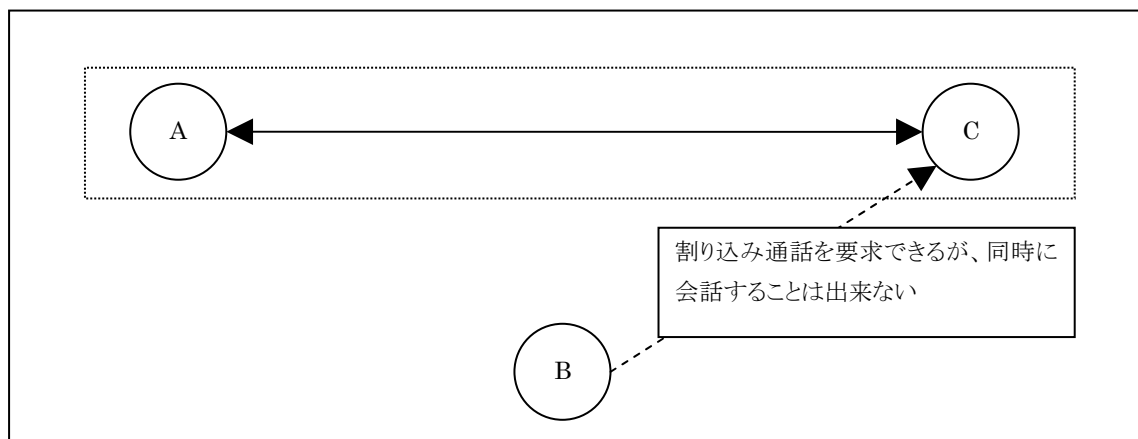


図 2-1 1対1の会話

上図では、AとCが会話中です。AとCが会話中であっても、BはCに会話を要求することが出来ませんが、CがBとの会話を始める場合、Aとの会話を終了しなければいけません。

## 2.2 複数人による会話

複数人による会話は、誰か一人だけが会話することが出来る「トランシーバーモード」と、参加者が同時に会話することが出来る「カンファレンスモード」の2つがあります。

### 2.2.1 トランシーバーモードによる会話

トランシーバーモードでは、同時に複数人が会話することは出来ず、誰か一人だけが話することが出来ます。

トランシーバーモードはリアルタイム性が高いアクションゲームなどにおいて、ごく短い音声のやり取りを行う、というような場合に利用されることを想定しています。トランシーバーモードで会話できる**上限は8人**です。

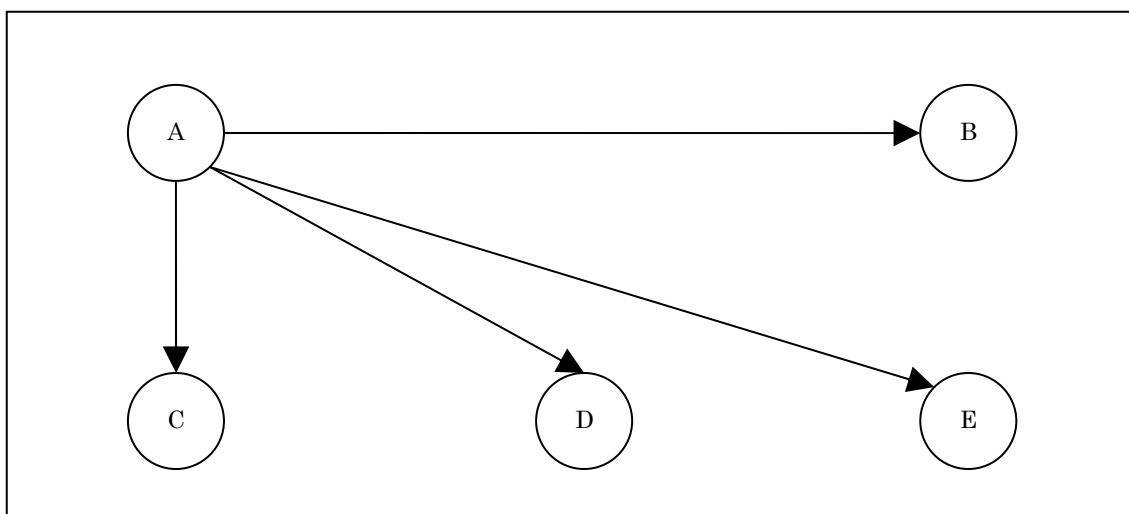


図 2-2 トランシーバーモードによる会話

上図では、AからEの5人がセッションに参加中であり、現在Aが他の4人に向けて話している状態です。この状態ではA以外は会話を開始することができず、またその要求を行っても拒否されます。

なお、このモードでは誰が会話を行うか(現在誰が会話に参加しているか、現在会話要求を出来る状態にあるかどうか)を制御するために親となるマシンが一台必要となります。親となるマシンをどのマシンにするかは、ゲーム側であらかじめ決めておく必要があります。

※VoiceChat ライブラリでは最大8人まで会話可能となっていますが、**実際に会話に参加できる人数の上限はネットワークの帯域利用状況と使用する音声フォーマットのビットレートに依存します**。必要とする帯域に関しては「5.2 音声フォーマット」を参考にしてください。

### 2.2.2 カンファレンスモードによる会話

カンファレンスモードでは、トランシーバーモードと異なり、参加者が同時に会話することが可能になっています。

カンファレンスモードはリアルタイム性が非常に高いアクションゲームなどにおいて、参加者全員が会話権の取得(=キーの操作など)の手続きをすることなく音声のやり取りを行う、というような場合に利用されることを想定しています。カンファレンスモードで会話できる**上限は4人**です



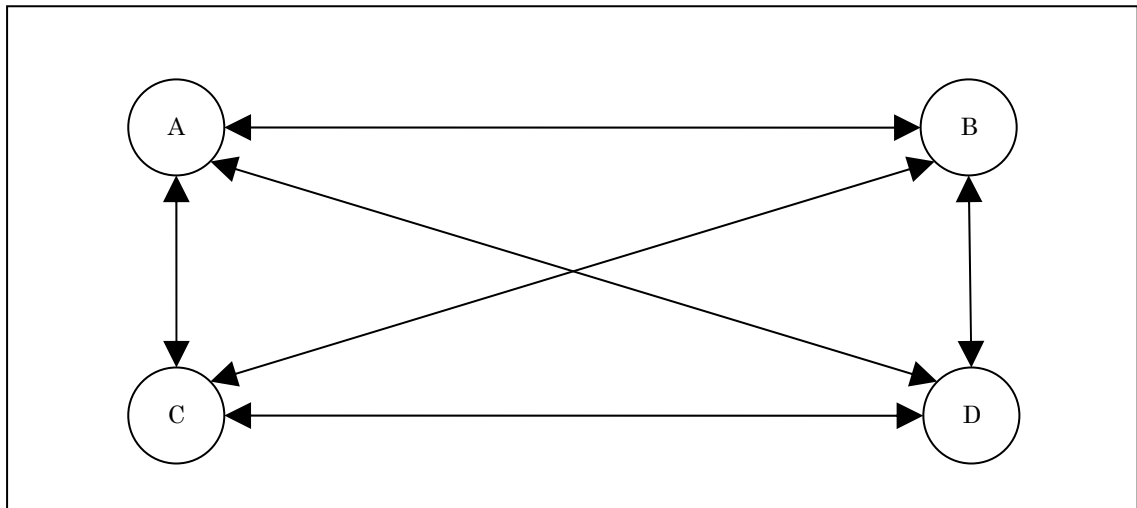


図 2-3 カンファレンスモードによる会話

上図では、A からDの4人がセッションに参加中です。トランシーバーモードと異なり、A が会話中であっても、B,C,D は会話することが出来ます。

なお、このモードでは誰が会話を行うか(現在誰が会話に参加しているか、現在会話要求を出来る状態にあるかどうか)を制御する必要がないため、トランシーバーモードのように親となるマシンを設定する必要はありません。ただし、カンファレンスに参加する端末はゲーム側にて決定していただく必要があります。

※VoiceChat ライブラリでは最大4人まで会話可能となっていますが、**実際に会話に参加できる人数の上限はネットワークの帯域利用状況と利用する音声フォーマットのビットレートに依存します**。必要とする帯域に関しては「5.2 音声フォーマット」を参考にしてください。

### 3 ライブラリ概要

VoiceChat ライブラリは以下の 2 つの機能ブロックを持っています。

- 会話セッションの管理
- 音声ストリーミング処理

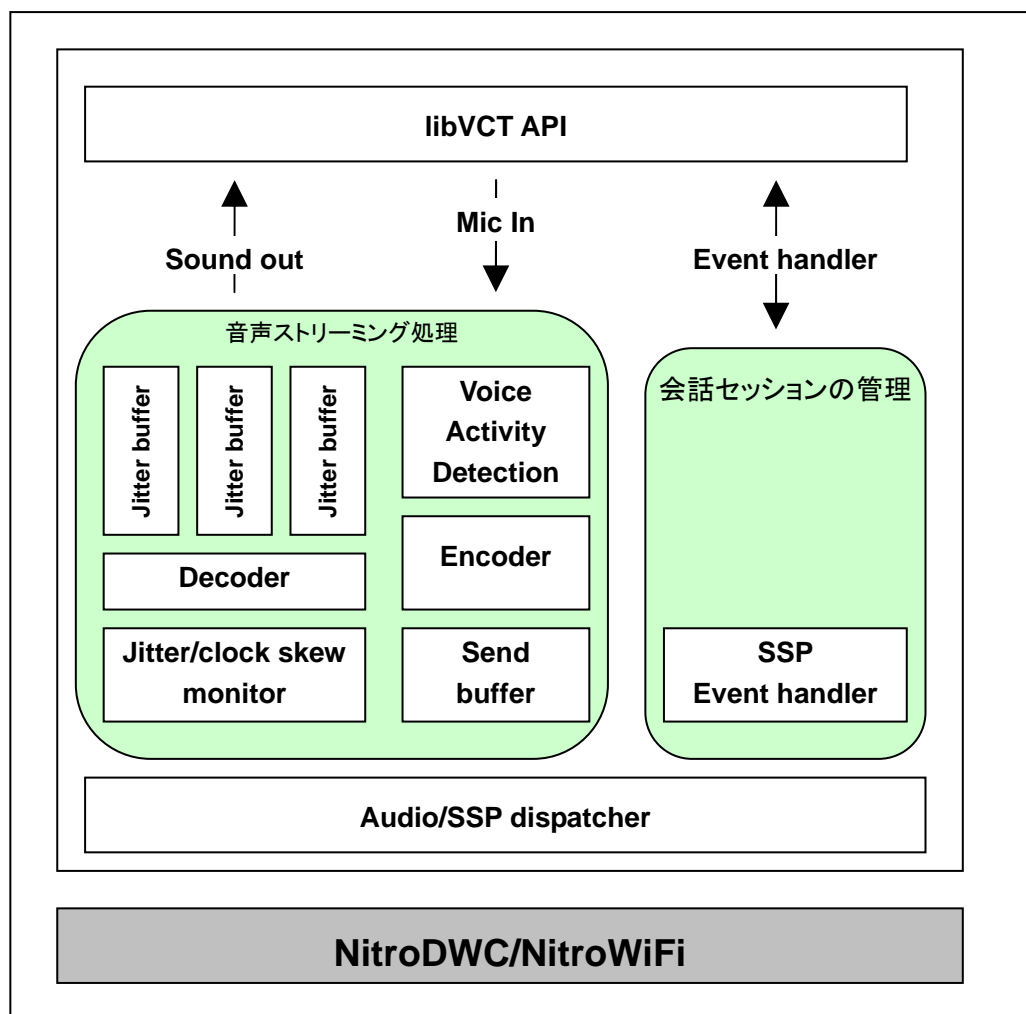


図 3-1 ライブラリ概要図

## 4 会話セッションの管理

### 4.1 シンプル・セッション・プロトコル

NitroDWCによってマッチメイク済みのクライアントに対して、VoiceChatを開始するためのプロトコルを「シンプル・セッション・プロトコル」と呼びます。(以下、SSP と記述します)

SSP は SIP(Session Initiation Protocol)に近いプロトコルですが、DS 上で高速動作が可能となるようカスタマイズされており、処理速度や実装規模の関係からすべてのやり取りがバイナリフォーマットとなっております。

### 4.2 フォーマット

SSP のやり取りはすべてバイナリ形式で行われる独自フォーマットを採用しています。データは 16 バイトでフォーマットは以下のとおりです。なお、データはすべてリトルエンディアンです。

表 4-1 SSP フォーマット

情報	長さ	内容
マジックトークン	4 バイト	'_VCT'(リトルエンディアン)
メソッドタイプ	1 バイト	0x00 または 0xFF
バージョン	1 バイト	バージョン番号(0x10)
メッセージコード	1 バイト	リクエスト・レスポンスのコード
コーデック	1 バイト	利用している音声フォーマット番号
AID	2 バイト	送信・受信 AID
拡張情報	1 バイト	NOTIFY メソッドの拡張情報
会話権 AID	1 バイト	トランシーバーモードで会話権を持つ AID
AID ビットマップ	4 バイト	トランシーバーモードでのクライアントリスト

SSP のやり取りは NitroDWC 関数(具体的には DWC\_SendReliable)を利用して送信されるため、先頭の4バイトに識別用のマジックトークンが付加されています。ゲーム上でやり取りするデータについては、先頭の4バイトが VoiceChat ライブラリと同じにならないようにしてください。

### 4.3 リクエストメソッドとレスポンスコード

SSP がサポートするリクエストメソッドとレスポンスコードは次のとおりです。

表 4-2 SSP リクエストメソッドコード

メソッド名	意味	コード
INVITE	セッションの確立要求(接続要求)	00
BYE	セッションの終了要求(切断)	01
CANCEL	セッション確立要求の取り消し	02
CONTACT	トランシーバー要求	03
NOTIFY	状態情報の通知	04

表 4-3 SSP レスポンスコード

メソッド名	意味	コード
200 OK	リクエストが成功した	00
400 Bad Request	不正なリクエストを受け取った	01
406 Not Acceptable	サポートされていないメディアタイプが指定されている	02
486 Busy Here	現在クライアントは会話中	03
487 Request Terminated	リクエストが BYE、または CANCEL によって終了させられた	04
603 Decline	辞退。相手側ユーザーが会話を望まない、または応答できない	05

## 4.4 イベントコード

VoiceChat ライブラリでは、SSP のシーケンスに応じて、以下のイベントコードが割り当てられています。ライブラリ利用者は以下のイベントコードをディスパッチすることでリクエスト(あるいはレスポンス)に対して、どのように応対すればよいかをプログラミングします。

表 4-4 イベントコード

イベント名	意味
NONE	何も発生していない
INCOMING	会話要求 (INVITE) を受信した
REJECT	相手側から拒否された (INVITE の拒否、コーデックの違い)
BUSY	相手側は通話中であった
CANCEL	呼び出しのキャンセルを受けた
NOTIFY_FREE	トランシーバーモードにて他の人の会話が終了した
NOTIFY_BUSY	トランシーバーモードにて他の人の会話が開始した
CONNECTED	接続が確立された
RESPONDBYE	BYE リクエストの受信
DISCONNECTED	切断された (BYE に対する 200 OK、または CANCEL に対する 487 Request Terminated)
CONTACT	CONTACT を受信した
TIMEOUT	リクエストがタイムアウトした
ABORT	不正なシーケンスのため、セッションが中止になった

## 4.5 電話モードにおけるシーケンス

以下では電話モードにおけるシーケンスを説明します。電話モードのシーケンスは、続くトランシーバーモード、カンファレンスモードにおいても利用される、基本となるシーケンスです。

リクエストメソッドは太字で、レスポンスコードは斜体で記しています。また、オレンジの四角で囲んだものは、リクエスト、またはレスポンスを受けた際に発生するイベントを示しています。

### 4.5.1 会話要求

会話要求の際は、発信側から INVITE メソッドが発行されます。応答側が 200 OK を返すと、会話を開始します。

200 OK 以降のやり取りはリアルタイムの音声データとなり、SSP とは別のプロトコルによって行われます。

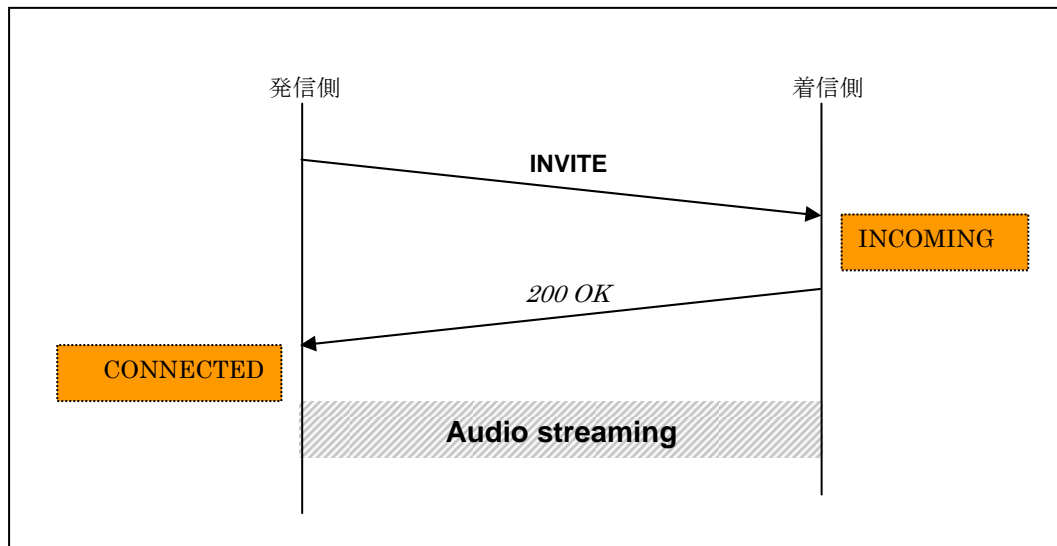


図 4-1 会話要求

#### 4.5.2 会話終了(通常終了)

正常な会話終了シーケンスは上記の通りとなります。終了を要求する側が **BYE** メソッドを発行すると、**BYE** を受けた側は **200 OK** を返します。ここでは通話を開始した側が **BYE** を送っていますが、着信側が **BYE** を送ってもかまいません。

なお、会話の終了要求を拒否することはできません。終了要求を無視しても、セッション自体は削除されます。

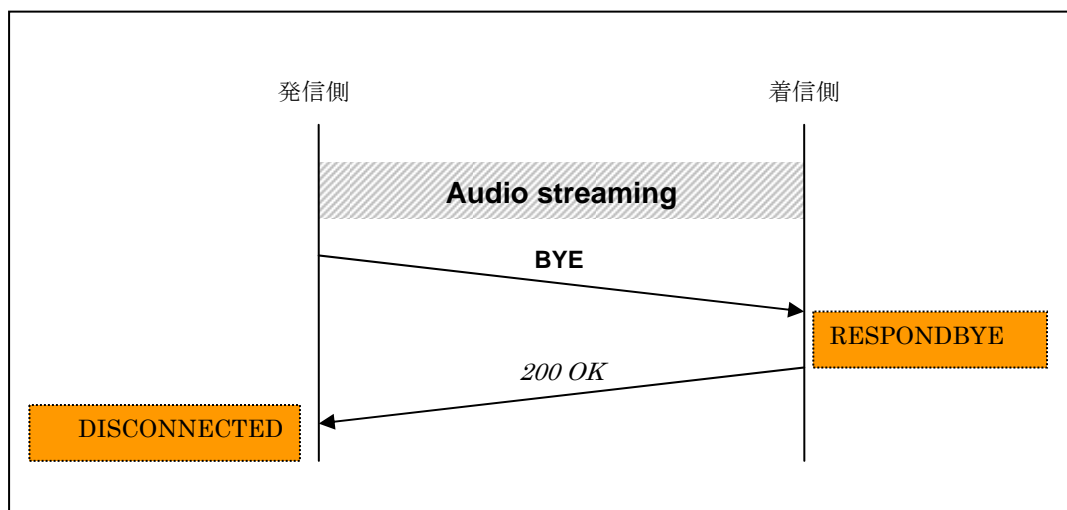


図 4-2 会話終了

#### 4.5.3 話中

**INVITE** 要求を受け取った側の端末が、すでに別の端末と会話中の場合、受信側は要求側に対して **486 Busy Here** を送り返すことで会話中であることを要求側に伝えることが出来ます。なお、実際に会話が始まっていない場合(受信側が別の端末とセッションの確立処理中などの場合)も **486 Busy Here** を返すことが出来ます。

**INVITE** 要求を受け取った側の端末が、**486 Busy Here** を返すのではなく、新しい端末との会話に応じる場合は、ま

ず現在の会話を終了させてから、新しい端末に対して 200 OK を返すことが可能です。

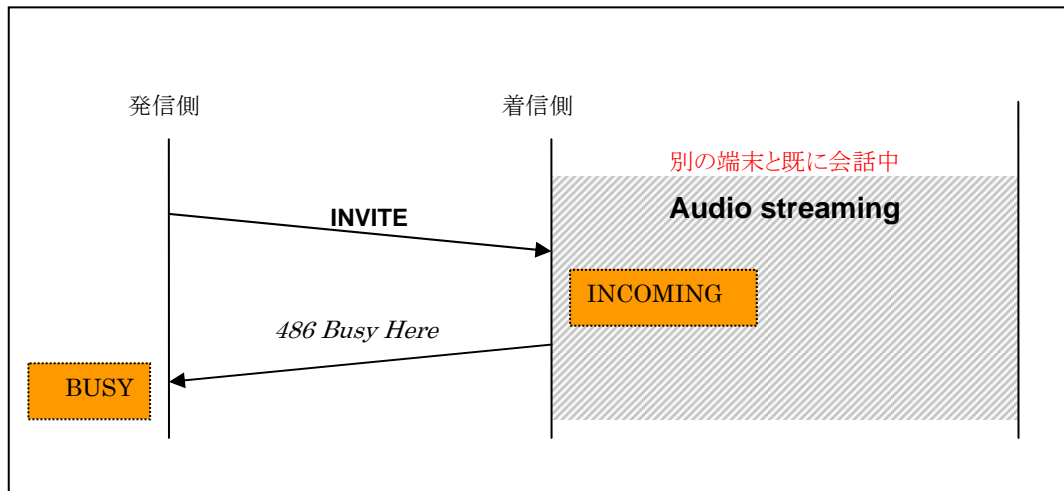


図 4-3 話中

#### 4.5.4 会話要求中止

会話を要求した端末が途中でその要求を中止する場合は、**CANCEL** メソッドを利用します。**CANCEL** メソッドを受けた着信側は **487 Request Terminated** を返信して、要求終了通知に答えます。

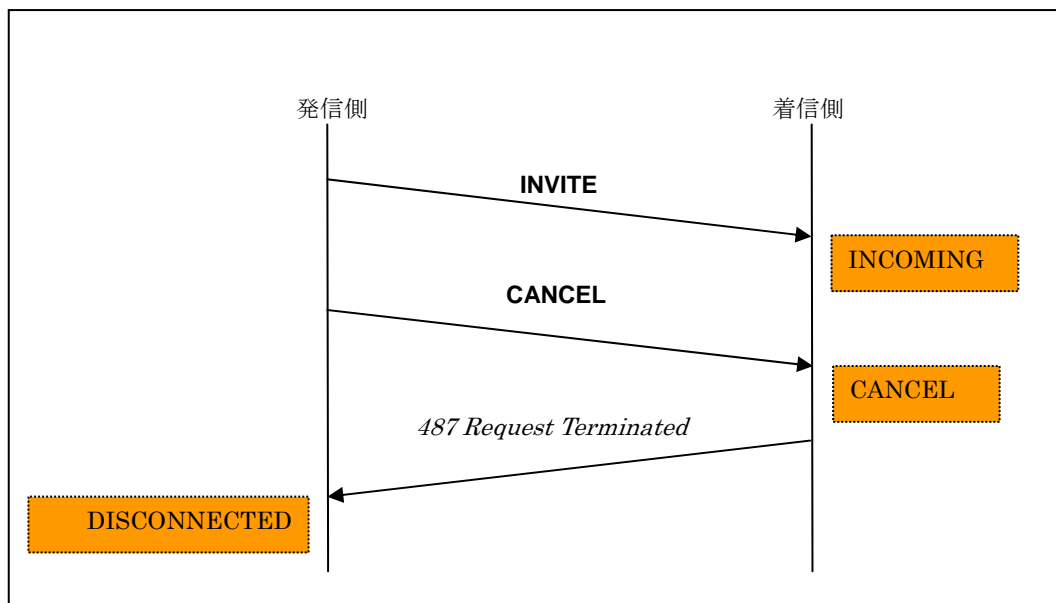


図 4-4 会話要求の中止

#### 4.5.5 受信拒否

会話要求を受けた端末が会話を拒否する操作を行った場合は発信側に対して **603 Decline** を返します。

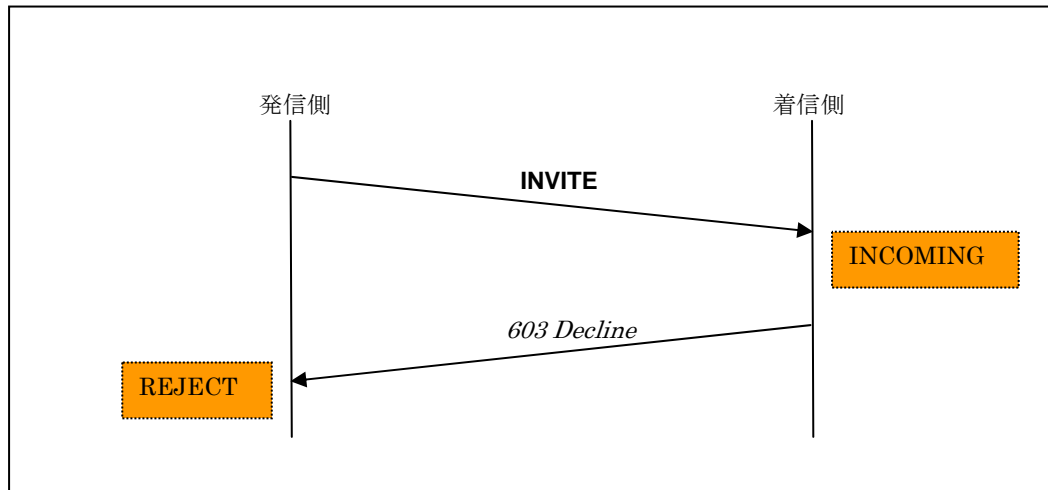


図 4-5 受信拒否

## 4.6 トランシーバーモードにおけるシーケンス

3人以上で会話を行いたい場合、VoiceChat ライブラリをトランシーバーとして利用することが可能です。トランシーバーモードは複数人が会話に参加できますが、同時に話すことが出来るのは一人だけです(半2重通信です)。

### 4.6.1 トランシーバーセッションへの参加と通知

トランシーバーモードでは、複数の人間とのやりとりを行うため、グルーピングのための処理が必要になります。そのためには、まず 1 台、サーバとなるマシンが必要になります。このサーバとなるマシンをどのマシンにするかはゲームの内容に依存することを考慮して、SSP によるグループ管理は実装されていません。ゲーム側で管理してください。

VoiceChat ライブラリでは、セッションの維持やグループ管理は行わず、会話の要求開始、終了のみを扱います。

### 4.6.2 参加要求～終了までのシーケンス

会話権を取得するためのシーケンスは、以下のようになります。

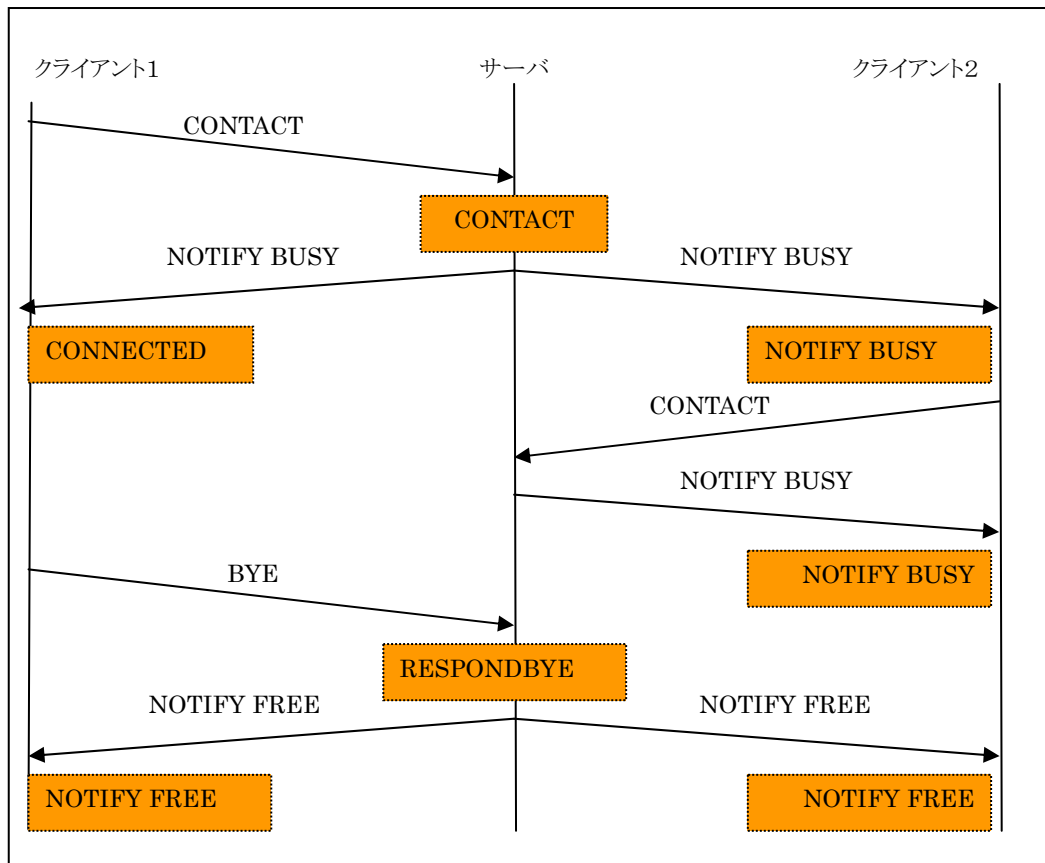


図 4-6 トランシーバーモードのシーケンス(クライアント側)

クライアント1がサーバに対して `CONTACT` メソッドを発行します。サーバは現在誰かが会話中であるかないかにかかわらず `NOTIFY BUSY` をすべてのクライアントに送信しますが、この際 `NOTIFY BUSY` のヘッダに、会話権をもつ端末の AID が付加されます。

`NOTIFY BUSY` を受信したクライアントは AID を見て、自分が会話権を取得したかどうかを判断できます。上記のように会話権を得たクライアントには `CONNECTED` イベントが、そうでないクライアントは `NOTIFY BUSY` イベントが発生し、発言権を得たマシンが音声を送信すると、参加しているすべてのクライアントに音声パケットが送信されます。

その後、クライアント 2 が `CONTACT` メソッドを発行しても、すでに会話中のため、サーバはクライアント 2 に対して、`NOTIFY BUSY` を送信します。ただし実際には、ライブラリ内部ですでにほかの人が会話中であると判断されることが多いため、クライアント 2 の `CONTACT` 要求はネットワーク上のサーバに問い合わせることなく、ライブラリ内部でエラーを返す場合がほとんどとなります。上記のようにネットワークに問い合わせが届くのは、クライアント 1 とクライアント 2 の `CONTACT` がほぼ同時にサーバに届いた場合に発生します。

クライアント 1 から `BYE` リクエストが発行されるか、トランシーバーの会話が一定期間 (30 秒) を経過するとサーバは `NOTIFY FREE` を送信します。この際、自分が発言権を持っている・いないにかかわらず、`NOTIFY FREE` イベントが発生します。

#### 4.6.3 会話要求(サーバ側)

サーバが `CONTACT` を発行した場合、`CONTACT` 要求はネットワークには送出されず、ライブラリ内部で処理された上で、クライアントが発行したのと同様のイベントが発生します。このため、イベントハンドラはサーバ・クライアントで共通のコードで動作させることが可能になっています。



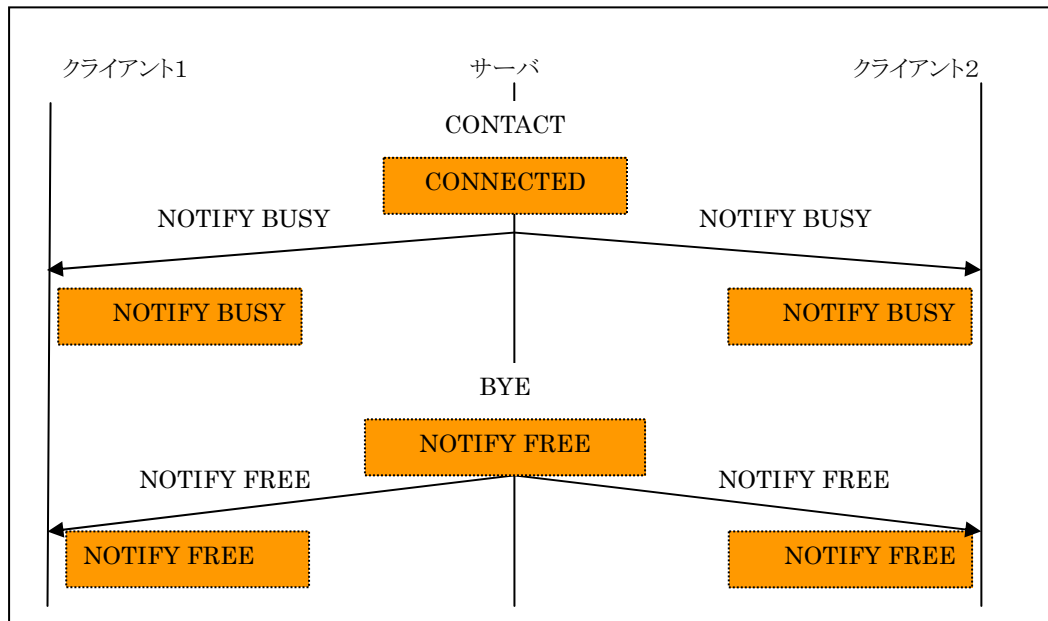


図 4-7 トランシーバーモードのシーケンス(サーバ側)

#### 4.6.4 NOTIFY メソッドの内容

NOTIFY は拡張情報とメッセージ本体に通知の内容が付加されます。その内容は以下のとおりです。

表 4-5 NOTIFY メソッドの内容

コード	意味
0x00	会話が可能になった(NOTIFY_FREE)
0x01	現在ほかの人が会話中(NOTIFY_BUSY)

NOTIFY の内容に応じて、自分が **CONTACT** を発行して会話を出来るかどうかとともに、メッセージボディ上から参加者がどれだけいるか(何台のリモートホストに音声パケットを送信しなければいけないか)などを知ることが出来ます。

## 4.7 カンファレンスモードにおけるシーケンス

カンファレンスモードでは、電話モードやトランシーバーモードと異なり、会話権の制御などを必要としないため、端末間でのやり取りは発生しません。ライブラリ利用者は API 呼び出しで、カンファレンスに参加中の端末を指定し、音声ストリームエンジンを起動するだけで即座に会話を開始することが出来ます。このため、カンファレンスモードでは、SSP のやり取りは発生しません。

なお、トランシーバーモード同様、カンファレンスグループの作成や管理はライブラリ利用者がゲームの内容に合わせて行ってください。

## 4.8 ステートマシン

VoiceChat ライブラリではシーケンスの異常が発生しないよう、ステートマシンによって管理されています。VoiceChat ライブラリがサポートする状態は次のとおりです。

表 4-6 ライブラリの内部状態

ステート名	意味
INIT	初期状態
OUTGOING	INVITE/CONTACT 発行済み
TALKING	接続完了・会話中
CONNECTED	接続完了、他の人が会話中 (トランシーバーモードのみ)
INCOMING	外部よりリクエストを受信
DISCONNECTING	切断処理中

#### 4.8.1 電話モードにおける状態遷移

電話モードにおける INVITE を発行した際の状態遷移図は以下のとおりです。送信は太字で、受信は斜体で記述してあります。

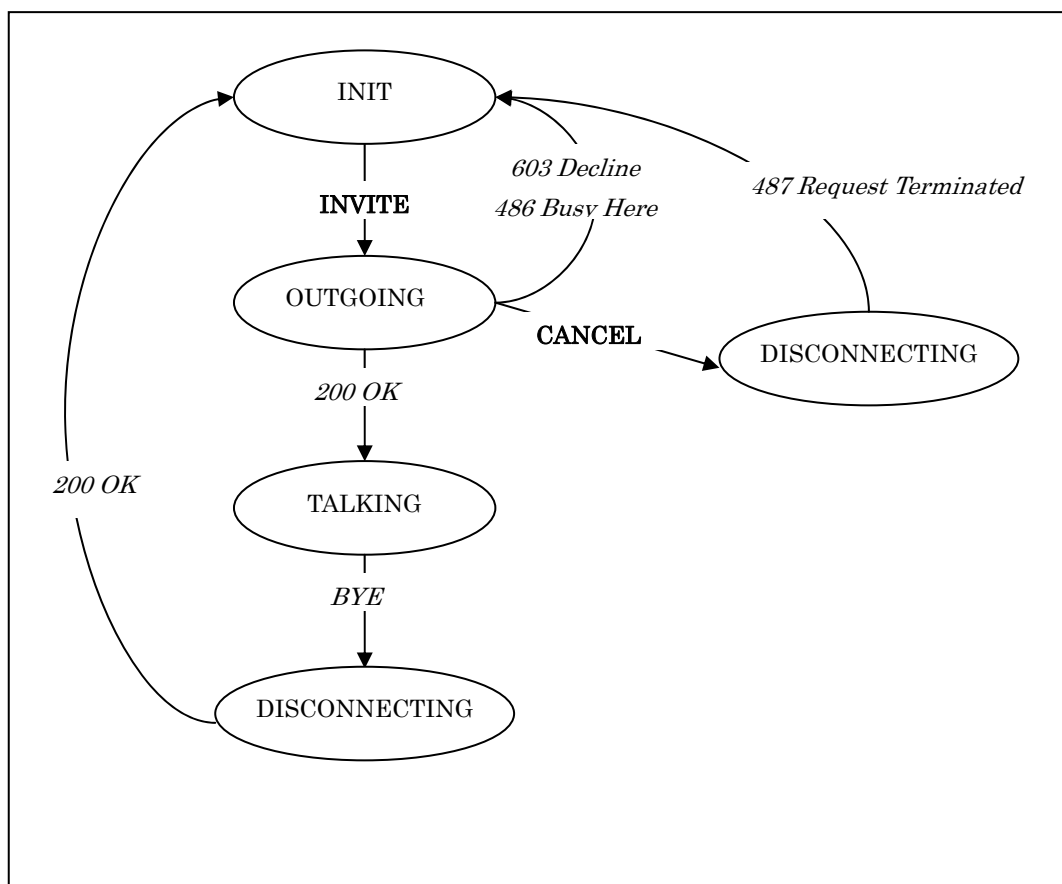


図 4-8 電話モードにおける状態遷移図(送信側)

逆に、INVITE を受けた際の状態遷移図は次のとおりです。

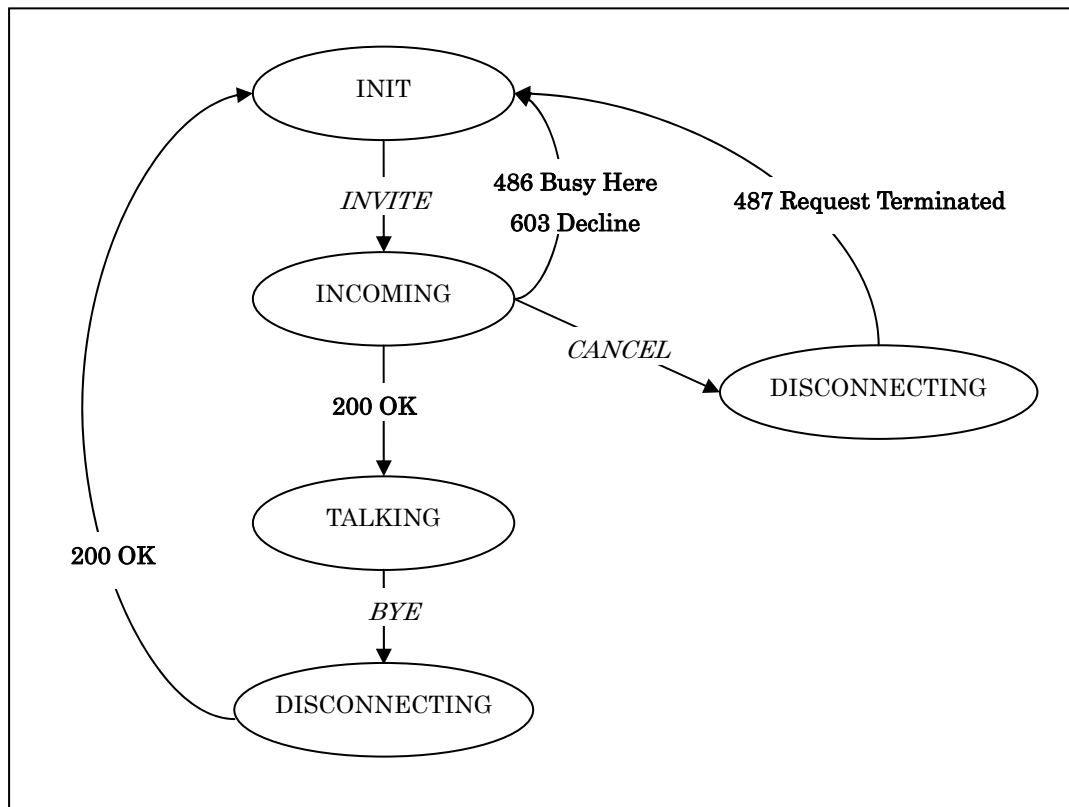


図 4-9 電話モードにおける状態遷移図(受信側)

## 4.8.2 トランシーバーモードにおける状態遷移

以下は、トランシーバーのサーバとなっているマシンの状態遷移図です。

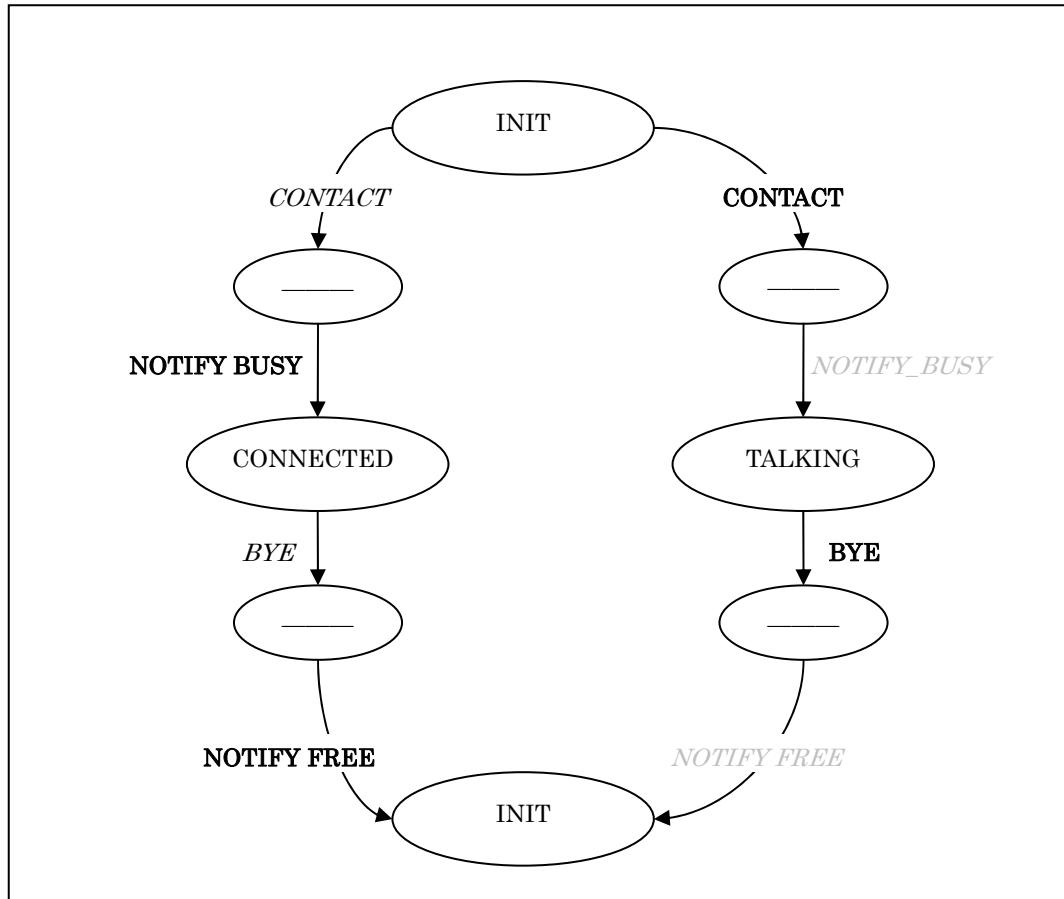


図 4-10 トランシーバーモードにおける状態遷移図(サーバ側)

トランシーバーモードでは常に1つの端末のみが会話可能であるため、会話要求の処理はライブラリ内で自動的に行われます。

クライアントから CONTACT 要求を受けた際は、自動的に CONNECTED に移行します。一方、サーバ自身が話す場合はライブラリ内部で TALKING 状態に遷移します(図の右側)。図ではサーバが CONTACT を送信するように書かれていますが、実際にはパケットを送信するのではなく、ライブラリ内部で状態遷移が行われます。

CONNECTED と TALKING の違いは「自分が話せるかどうか」であり、CONNECTED の場合、相手の声を聞くことは出来るが自分は話せない状態、TALKING は自分が話している状態です。

一方、トランシーバーモードのクライアントは以下のようになります。

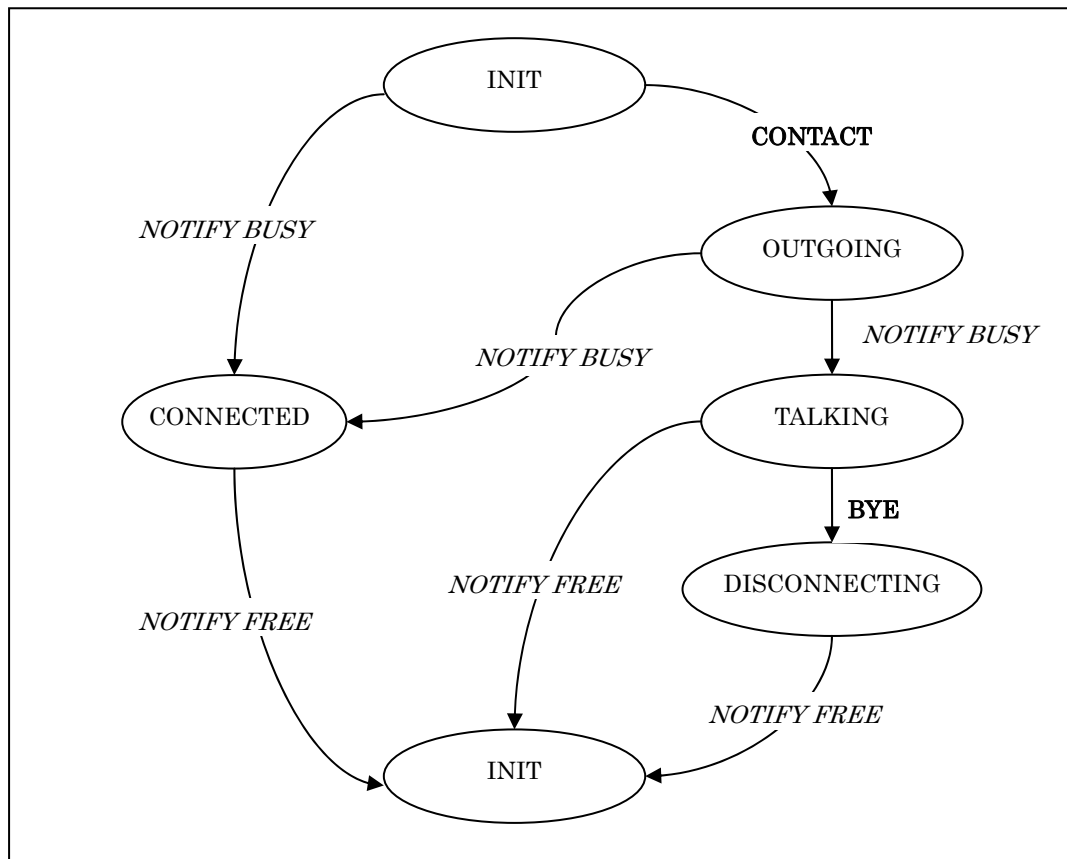


図 4-11 トランシーバーモードにおける状態遷移図(クライアント側)

クライアントマシンは CONTACT リクエストを受け取ることはないかわりに、サーバから NOTIFY を受け取ります。NOTIFY BUSY を受け取ると CONNECTED へ、NOTIFY FREE を受け取ると INIT に遷移します(上図の左側部分に相当)。

CONTACT を発行した場合は、一旦 OUTGOING 状態を経由して NOTIFY BUSY を受け取ります。NOTIFY BUSY には会話可能なクライアントのアドレスが記載されているため、この情報を元に自分が会話可能であれば TALKING に、既に誰かが話している場合は CONNECTED に遷移します。その後、BYE を発行すると、NOTIFY FREE を受けて INIT に戻ります(上図の右側部分に相当)。

TALKING 中でも NOTIFY FREE を受けることに注意してください。トランシーバーモードでは、デフォルトでは会話は 30 秒に制限されており、30 秒を越えると、サーバが NOTIFY FREE を送信して強制的に会話を終了させます。

#### 4.8.3 カンファレンスモードにおける状態遷移

カンファレンスモードではシーケンスが発生しないため、内部状態遷移は「INIT」と「TALKING」しかなく、この2つの間を行き来するだけです。

## 4.9 キープアライブとタイムアウト

SSP では、セッションを維持するためのキープアライブ処理は行われていません。接続が確立されているかどうかは、アプリケーション側で処理してください。

なお、トランシーバーモードでの会話状態の継続は、デフォルトで 30 秒に制限されています。

## 5 音声ストリーミング処理

SSP によってセッションが確立された端末はリアルタイムプロトコルを用いて音声データをやり取りします。

このリアルタイムプロトコルは RTP に近いプロトコルですが、処理速度等を考慮し RTP の全仕様を満たすものではなく、また互換性也没有せん。

### 5.1 概要

やり取りされる音声は現信号が 16Bit/8KHz でサンプリングされたデータです。

VoiceChat ライブラリには VAD (Voice Activity Detection) が搭載されているため、デフォルトでは無音のときは音声パケットが送信されません。なお、無音の際でも音声を送信する・しないはライブラリにて設定可能です。

また、一度に送信されるパケットのデータ量 (= オーディオフレーム長) は 68ms です。

### 5.2 音声フォーマット

VoiceChat ライブラリで利用できる音声フォーマットは以下のとおりです (エンコード前のフォーマットはすべて 16bit/8KHz のサンプリングデータです)。

表 5-1 音声フォーマットとビットレートの関係

フォーマット	ビットレート	音質 (カッコ内は音質の良さの順位)
8bit / 8KHz Raw	64Kbps	良い (2)
G.711 u-Law	64Kbps	良い (1)
4bit ADPCM	32Kbps	普通 (3)
3bit ADPCM	24Kbps	やや悪い (4)
2bit ADPCM	16Kbps	悪い (5)

ビットレートは片側でのレートのため、電話モードにて双方が同時に会話する場合、上記の2倍の無線帯域を利用することになります。また、実際にはオーディオヘッダと IP/UDP ヘッダが加わります。

トランシーバーの受信側は1つの端末からのみ受信するため、トランシーバーセッションの参加台数にかかわらず使用帯域は一定ですが、送信側は、送信する台数に乗じた帯域を使用することに注意してください。また、カンファレンスモードでは、(カンファレンスモードに参加している人数-1) \* 2 倍の帯域が必要です。なお、トランシーバーモード、およびカンファレンスモードでは、消費帯域の問題から 8bit/8KHz Raw フォーマットと G.711 u-Law フォーマットは利用できません。

上記のビットレートは音声データのみであり、実際には IP/UDP ヘッダ、および VoiceChat の音声パケットヘッダ (合計 40 バイト) が付加されます。実際の帯域消費量は以下のとおりとなります。なお、IP データサイズとはペイロードサイズに IP ヘッダ (20 バイト)、UDP ヘッダ (8 バイト)、および VoiceChat のヘッダ (12 バイト) の合計 40 バイトを足した値です。

表 5-2 音声フォーマットと帯域消費量の関係

コーデック	ペイロードサイズ(byte)	IP データサイズ(byte)	IP 帯域幅(bit / second)
8bit Raw/G.711 u-law	544	584	68706
4bit ADPCM	276	316	37176
3bitADPCM	208	248	29176
2bit ADPCM	140	180	21176

VoiceChat ライブラリはストリーミング中の音声フォーマットの変更をサポートしています。トランシーバーモードやカンファレンスモードでは、参加人数に応じて、音声フォーマットを動的に変更させ、帯域と音質のバランスを変更させることが可能です。

### 5.3 VAD (Voice Activity Detection)の調整

VoiceChat ライブラリには、VAD(Voice Activity Detection: 音声発生検出・無音検出)機能が搭載されています。VAD を利用すると、会話していないとき(無音の際)には音声パケットを送出しないよう制御できるため、ネットワークとCPU の負荷を低減できます。

無音を検出した際、すぐに音声パケットの送信を停止すると、会話中のちょっとした息継ぎでもパケット送信を停止するため、聞き取り側は音声途切れ途切れになっているように聞こえることがあります。このようなことから、VoiceChat ライブラリの VAD では「リリース時間」を設定しています。リリース時間はデフォルトで 1020ms (68ms \* 15)となっており、この値は VCT\_SetVADReleaseTime 関数で変更できます。

### 5.4 エコーキャンセル

Version 1.1 から、「エコーキャンセル機能」が搭載されています。エコーキャンセル機能を利用すると、ヘッドセットを利用しなくても、会話を楽しむことが可能になります。ただし、CPU 負荷の関係から、本格的なアコースティックエコーキャンセル機能を提供するのではなく、ハウリングを抑制することを目的としています。

エコーキャンセル機能を利用する場合は、次の点に注意してください。

- マイクのゲインは 160db のみ対応しています。160db 以外のゲイン設定でエコーキャンセルを有効にすると、音声録音処理に悪影響を及ぼすことがあります。
- 音声の再生に2チャンネル以上利用しないでください。
- 6章で説明されている方法で音声録音・再生処理を実装してください。異なる実装方法を用いると、正しくエコーキャンセルされなくなります。
- 本エコーキャンセルは音声のみをキャンセルする機能であり、BGM や効果音はキャンセルできません。これら BGM や効果音が原因でハウリングが発生することはありませんが、本体マイクを通じて相手に聞こえることで耳障りになる可能性がありますので、音声の再生音量よりも音量を絞ることを推奨します。

### 5.5 適応型ジッタバッファ

通常、IP ネットワークにおける音声ストリーミングにおいては、ネットワークのジッタを吸収するために、一度データがバッファリングされます。バッファリングされる時間は、音声再生遅延に影響のない程度の、ある固定長のバッファ(たとえば 50ms)が利用されるのが一般的です。

VoiceChat ライブラリではジッタの多いネットワーク環境でも正しい音声ストリーミングが行われるよう、「適応型ジッタバッファ(Adaptive jitter buffer)」と呼ばれるアルゴリズムが搭載されています。

適用型ジッタバッファの処理はすべてライブラリ内部にて処理されるため、利用者はバッファの状態を気にする必要なく利用することが可能です。

## 5.6 音声パケットの詳細

パケットのデータフォーマットは次のとおりです(横方向 32bit)。

マジックトークン('VCT')						
V	P	X	PH	M	PT	シーケンス番号
タイムスタンプ						
ペイロードデータ(140~544byte)						

図 5-1 音声パケットの内容

SSP パケット同様、音声データも DWC 関数を用いて送受信されるため、先頭4バイトにマジックトークンが付加されています。



## 6 API の解説

本章では、実際のライブラリの利用手順を説明します。なお、ライブラリ利用前に VoiceChat ライブラリのパスを環境変数 `NITROLIBVCT_ROOT` に設定してください。また、アプリケーション・ライブラリを作成する `Makefile` には、以下の共通ビルドツールをインクルードして下さい。

- `include $(NITROLIBVCT_ROOT)/build/buildtools/commondefs`
- `include $(NITROLIBVCT_ROOT)/build/buildtools/modulerrules`

このビルドツールをインクルードすると NITRO-WiFi, NITRO-DWC のビルドツールもインクルードされます。

### 6.1 ライブラリの初期化と終了

すべての VoiceChat ライブラリ関数を使用する前に `VCT_Init` 関数を用いて VoiceChat ライブラリを初期化する必要があります。この関数の呼び出しはモードを切り替えるごとに行う必要があります。

初期化の際、`VCTConfig` 構造体を用いて、使用するモードやオーディオストリームの受信バッファを指定する必要があります。`VCTConfig` 構造体の内容は以下のとおりです。

表 6-1 `VCTConfig` 構造体

変数名	内容
<code>session</code>	<code>VCTSession</code> 構造体を保存する領域を指定します
<code>numSession</code>	<code>session</code> で指定した領域のサイズを指定します
<code>mode</code>	VoiceChat の動作モードを指定します
<code>aid</code>	端末の <code>aid</code> を指定します
<code>audioBuffer</code>	オーディオストリームの受信バッファを指定します (32 バイトアライメント必須)
<code>audioBufferSize</code>	<code>audioBuffer</code> で指定したバッファのサイズを指定します
<code>callback</code>	イベント発生時のコールバック関数を指定します
<code>userData</code>	コールバック関数呼び出し時に渡したいユーザーデータを指定します

電話モードで初期化する場合は次のようになります。

```
static VCTSession sSession[2]; /* VCTSession構造体を保存する場所 */
static u8         stBuffer[8 * VCT_AUDIO_BUFFER_SIZE]; /* 音声受信用バッファ */

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)
```

コード 6-1 電話モードでの初期化例

上記例ではセッションを2つ確保しており、割り込み要求などにも対応できます。ここで確保するセッションを1つにする

と、割り込み要求などがあった際、VoiceChat ライブラリが自動的に相手に **Busy** を返すようになります。なお、トランシーバーモード時の場合は必要なセッション数は1つ、カンファレンスモード時の必要なセッション数は「最大参加人数－1」です。

VoiceChat のモード、自ホストの AID を指定後、音声ストリーミング用のバッファを指定します。音声ストリームバッファは最低でも100ms分は用意しないと、音声再生が途切れ途切れになります。250ms以上が推奨値です。バッファ数の指定はパケットの音声パケットの長さの単位で指定します。上記コード例では  $8 \times \text{VCT\_AUDIO\_BUFFER\_SIZE}$  分を用意していて、これは  $8 \times 68\text{ms} = 544\text{ms}$  分(9154 バイト)のバッファになります。また、カンファレンスモード時は、必要となるバッファ数をクライアントの台数分だけ増やす必要があります。4人参加のカンファレンスの場合において、1クライアントあたり上記例と同じだけのバッファを用意したい場合、 $3 \times (8 \times \text{VCT\_AUDIO\_BUFFER\_SIZE})$  分必要になります。

このバッファ数を増やしたからといって音声再生に遅延が生じるわけではありません。実際のバッファ数はダイナミックジッタバッファがネットワークのジッタを計測した上で動的に決定します。

なお、音声ストリーミングバッファのポインタは 32 バイトにアライメントされている必要があり、audioBufferSize は VCT\_AUDIO\_BUFFER\_SIZE の整数倍である必要があります。

最後にコールバック関数とその関数に与えるパラメータを指定します。

VoiceChat ライブラリの使用を終了する場合は VCT\_Cleanup 関数を呼び出します。別のモードに変更する際には一度 VCT\_Cleanup 関数で終了処理を行った後で VCT\_Init を呼び出す必要があります。

## 6.2 VoiceChat データのハンドリング

VoiceChat ライブラリは NitroDWC の関数を利用して音声や SSP のデータをやり取りします。このため、NitroDWC のデータ受信関数(DWC\_SetUserRecvCallback 関数で指定した関数)にはゲームのデータと VoiceChat のデータの両方を取り扱う必要があります。

VCT\_HandleData 関数は受信したデータが VoiceChat のものかどうかを判断し、VoiceChat のデータであれば必要な処理を行う関数です。受信したデータが VoiceChat のものであるかどうかは、受信データの先頭の4バイトが VoiceChat のマジックトークンであるかどうかで判定しています(4.2フォーマット、5.6音声パケットの詳細を参照)。このため、ゲームでやりとりするデータの先頭4バイトが VoiceChat のマジックトークンと同じにならないよう注意してください。

この関数は次のようにして利用します。

```
/* DWC_SetUserRecvCallback関数で指定したDWC受信コールバック関数 */
void UserRecvCallback(u8 aid, u8* buffer, int size)
{
    BOOL flag;

    flag = VCT_HandleData(aid, buffer, size);
    if (flag == FALSE) {
        /*
         戻り値がFALSEの場合、受信したデータがVoiceChatのデータ
         ではないか、またはVoiceChatライブラリが初期化されていない
         状態です。ここにゲームに必要な処理を実装します。
        */
    }
}
```

コード 6-2 受信データのハンドリング

戻り値が TRUE ならば受信したデータは VoiceChat ライブラリのデータです。この場合、VCT\_HandleData 関数内で

必要な処理、およびコールバック関数が呼び出されます。

また、音声のエンコードと送信処理、およびタイムアウトのチェックなどを行うため、ピクチャーフレーム(1/60 秒)に1回 VCT\_Main を呼び出す必要があります (VCT\_Main は1ピクチャーフレームに必ず1回呼び出すようにしてください)。

```
/* ゲームのメイン関数 */
void GameMain()
{
    while (1) {
        OS_WaitIrq ( TRUE, OS_IE_V_BLANK );
        VCT_Main();
        /*
            その他の処理...
        */
    }
}
```

コード 6-3 VoiceChat メインルーチンの実行

## 6.3 SSP イベント処理

電話モードやトランシーバーモードでは SSP 関数を用いて相手側とセッションの確立を行う必要があります。本節では電話モードにおける会話の要求から終了までを解説します。なお、シーケンスのやり取りは「4.5電話モードにおけるシーケンス」を参考にしてください。

まず、会話を要求する側の端末は VCT\_CreateSession で新しいセッションを作成し、VCT\_Request 関数を用いて相手側端末に会話を要求します。

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

コード 6-4 会話要求の例

端末がイベントを受け取ると、VCT\_HandleData を経由して VCT\_Init で指定したコールバック関数が呼び出されます。静的変数 sSession を用いて、複数のセッションを同時に処理してしまわないようにしています。

新しいセッションの場合は VCT\_Response 関数を用いて VCT\_RESPONSE\_OK(200 OK)を返し、その後 VCT\_StartStreaming 関数で音声ストリームを開始してください。

```
static *sSession = NULL;

/* VCT_Initで指定したコールバック関数 */
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;
    }
}
```

#### コード 6-5 INCOMING イベントの処理

VCT\_RESPONSE\_OK(200 OK)を返したため、INVITEを発行した端末側では CONNECTED イベントが発生します。

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

#### コード 6-6 CONNECTED イベントの処理

ここではイベント発生直後に応答しましたが、実際には INCOMING イベントの際にリングトーンを鳴らす、アニメーションを表示するなどして、ユーザーに会話要求を受け入れるかどうかを選択させることもできます。

この場合はセッションを一旦保存し、後ほどユーザーからのボタン操作の際に、VCTSession 構造体の state メンバの状態を見て、セッションのステートが INCOMING であれば VCT\_RESPONSE\_OK(200 OK)を返す、という処理を行います。

```

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;
    }
}

/* Aボタンが押されたときの関数 */
void onAButtonDown()
{
    if (sSession->state == VCT_STATE_INCOMING) {
        VCT_Response( sSession, VCT_RESPONSE_OK );
        VCT_StartStreaming (sSession );
    }
}

```

コード 6-7 INCOMING イベントの処理

なお、トランシーバーモードでは、CONTACT や BYE リクエストを送るための専用関数、VCT\_Contact と VCT\_Release を利用してください。トランシーバーモード時にこれらのリクエストを VCT\_Request や VCT\_Response 関数は利用できません。

## 6.4 音声データのハンドリング

音声の録音再生・送信受信処理をスムーズに行う、および遅延を最小限に抑えるため、音声データの処理は一定周期で確実に実行されるよう、割り込み処理などを利用して行うことを強く推奨します。

本節では NitroSystem のストリーミング再生を利用した音声データのハンドリング方法を示します。なお、エコーキャンセルを利用する場合、NitroSystem の利用が前提となります。

まず、NNS\_SndStrmSetup を用いて、サウンド処理の初期化を行います。サウンドのフォーマットは符号付きの 16bit、サンプリングレートは 8KHz としてください(マイクの量子化パラメータは MIC\_SAMPLING\_TYPE\_SIGNED\_12BIT としてください)。また、サンプリングコールバックの呼び出し間隔は VoiceChat ライブラリのパケットサイズである 68ms を指定します。

次にマイクの自動サンプリングパラメータを設定します。ただし、マイクの自動サンプリングの開始はすぐには行わずに、**「NNS\_SndStrmSetup で設定したコールバック関数が最初に呼び出された」**タイミングで開始してください。この手順で初期化しないと、マイクのサンプリングと音声のストリーミングの遅延が一定ではなくなるため、エコーキャンセルが正常に働かなくなります。エコーキャンセル機能を利用する場合は、必ずこの手順で初期化を行ってください。

マイクのサンプリングレートは NitroSDK で定義されているサンプリングレート(MICSamplingRate)を利用せず、 $(NNS\_SND\_STRM\_TIMER\_CLOCK / 8000) * 64$  を指定してください。こうすることで、ストリーム再生のクロックと確実に一致したレートでマイクのサンプリングを行うことができます。

```

/* サウンド関連のパラメータ */
#define SAMPLING_RATE      VCT_AUDIO_FRAME_RATE    // 8kHz
#define SAMPLING_TIME      VCT_AUDIO_FRAME_LENGTH  // 68ms
#define SAMPLING_BYTE      2                      // 16bit

/* NNS_SndStrm用のバッファを確保 */
#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 NNSndStrm sSndStream;
static MICAutoParam sMicParam;
static BOOL sFirstCallback = TRUE;

/*
   サウンドストリームとマイクの初期化
*/
void InitSound()
{
    u32      length;
    u8       cArray[1] = {7};    /* 7チャンネルを利用 */

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

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

    /* マイクの自動サンプリングのパラメータを設定 */
    sMicParam.type      = MIC_SAMPLING_TYPE_SIGNED_12BIT;
    sMicParam.buffer     = sRecBuffer;
    sMicParam.size      = length * 2;
    sMicParam.rate      = (u32)((NNS_SND_STRM_TIMER_CLOCK / SAMPLING_RATE) * 64;
    sMicParam.loop_enable = TRUE;
    sMicParam.full_callback = NULL;
    sMicParam.full_arg   = NULL;
    sFirstCallback = TRUE;

    /* ストリーム再生の開始 */
    NNS_SndStrmSetup(
        &sSndStream,
        NNS_SND_STRM_FORMAT_PCM16,
        sPlayBuffer,
        length * 2,
        NNS_SND_STRM_TIMER_CLOCK / SAMPLING_RATE,
        2,
        SndCallback,
        sRecBuffer);
}

```

#### コード 6-8 サウンド処理の初期化

コールバック関数の実装は以下の通りとなります。静的変数 `sFirstCallback` を確認して、一番最初の呼び出し時に(ストリーミング開始と同時に) `MIC_StartAutoSamplingAsync` でマイクの自動サンプリングを開始しています。その後、マイクの現在位置を取得した上で、VCT 関数を用いて送受信処理を行います。

```
/*
 サウンドコールバック関数
 */

static void micCallback(MICResult result, void *arg)
{
#pragma unused(result, arg)
}

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

    /* stsがNNS_SND_STRM_CALLBACK_SETUPの時は、バッファをクリアする必要あり */
    if ( sts == NNS_SND_STRM_CALLBACK_SETUP )
    {
        MI_CpuClearFast( buffer[0], length );
        return;
    }

    /* コールバックが最初に呼び出されたときにマイクのサンプリングを開始する */
    if ( sFirstCallback ) {
        MIC_StartAutoSamplingAsync(&sMicParam, micCallback, NULL);
        sFirstCallback = FALSE;
    }

    /* 現在のマイクサンプリング位置を検索 */
    micSrc = (u8*)arg;
    micAddr = MIC_GetLastSamplingAddress();
    offset = (u32)((u8*)micAddr - micSrc);
    if ( offset < length )
    {
        micSrc = micSrc + length;
    }

    /* 音声データの送受信 */
    VCT_SendAudio( micSrc, length )
    VCT_ReceiveAudio( buffer[0], length, NULL );

    /* キャッシュのフラッシュとクリア */
    for (ch = 0; ch < nChannels; ++ch) {
        DC_FlushRange(buffer[ch], length);
    }
    DC_InvalidateRange(micSrc, length);

    return;
}
```

コード 6-9 サウンドコールバック関数の処理

VCT\_SendAudio/VCT\_ReceiveAudio は割り込み関数内で呼び出されることを想定しているため、音声のエンコード・デコード、およびパケットの送受信処理などはこの関数内では行っていない。音声のエンコードと送信処理は

VCT\_Main 関数内にて、また受信処理とデコード処理は VCT\_HandleData 関数内にて行われます。

## 6.5 音声の録音・再生と BGM 再生の抑制

VCT\_ReceiveAudio の戻り値を利用すると、実際に再生すべき音声があるかどうかを判定できます。VCT\_ReceiveAudio は再生すべき受信音声がある場合 (バッファを音声データで埋めた場合) は TRUE を、再生すべき受信音声がない場合 (バッファをクリアした場合) は FALSE を返します。これを利用して「相手が話しているときは BGM をミュート、ないしは音量を落とす」といったプログラミングをすることが可能です。

また、VAD をオンにしている場合、VCT\_Main 内で VAD 判定処理が行われますが、VCT\_GetVADInfo 関数を用いると直近の VAD の判定結果を取得することが出来ます。

これらの API を利用することで「自分が話しているときは BGM をミュート、もしくは音量を落とし、相手側に不要な音の送信を抑える」という処理が可能になります。

コード 6-3、およびコード 6-9 を BGM ミュートに対応させたコードは以下のようになります。

```
static BOOL sNeedMute = FALSE;

/* サウンドコールバック部 */
{
    /* VCT_ReceiveAudioの戻り値を保存 */
    sNeedMute = VCT_ReceiveAudio( buffer[0], length, NULL );
    VCT_SendAudio (micSrc, length);
}

/* メインルーチン部 */
{
    VCTVADInfo info;
    VCT_Main();
    VCT_GetVADInfo(&info);
    if (sNeedMute || info.activity) {
        /* ここで BGM のミュート処理を行う */
    }
}
```

コード 6-10 音声送受信時に BGM をミュートする



## 7 デモプログラムの解説

### 7.1 プログラムの起動からマッチメイクの完了まで

デモプログラムは NitroDWC のデモプログラム”dwc\_match”とほぼ同様の手順にてマッチメイクの完了までを行います。

起動後、StartIP/Loginを選択することでAPへの接続とニンテンドー Wi-Fi コネクションへのログインを行います。なお、あらかじめ接続先APの登録とWi-Fi Connection IDの作成を行ってください。

ログイン後、任意の接続タイプでマッチメイク処理を行ってください。マッチメイクが完了すると、以下の画面が表示されます。

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

図 7-1 マッチメイク後の画面

メニューの意味は次のとおりです。

表 7-1 GAME CONNECTED MODE のメニューの内容

メニュー	内容
Start Phone	電話モード開始
Start Transceiver	トランシーバーモード開始
Start Conference	カンファレンスモード開始
Close Connections	接続解除
Close All Hard	強制切断

## 7.2 電話モードのテストプログラム

GAME CONNECTED MODE で Start Phone を選択すると以下のような画面が表示され、電話モードのデモを実行することができます。

```
=====
Phone Mode
> aid: 0 / pid: 599995835 (ph)
> aid: 1 / pid: 599995836 (ph)
> aid: 2 / pid: 599995837 (ph)
=====
```

図 7-2 電話モード

画面上にはマッチメイクが完了したクライアントの一覧が表示されます。Aid の他に pid と、以下のいずれかのクライアントのモードが表示されます。

表 7-2 クライアントの情報

表示	内容
(x)	モード未設定
(ph)	電話モード
(tr)	トランシーバーモード
(talk)	会話中

十字ボタンを利用して、会話したい端末を選択した上で、A ボタンを押すことで、相手呼び出すことができます。呼び出された相手側は A ボタンで受信、B ボタンで拒否ができます。また、呼び出し側が呼び出し中に B ボタンを押すことで呼び出しのキャンセルを行えます。

その他のボタン操作は以下のとおりです。

表 7-3 電話モードのキー操作

ボタン	内容
カーソル	カーソル上下左右移動
A	アイドル中: 電話をかける 着信中: 電話を取る
B	着信中: 会話要求を拒否 発信中: 発信をキャンセル 会話中: 電話を切る
X/Y	音声フォーマットを切り替え
Select	メニュー切り替え
R	矩形波を送信

Select ボタンを押すと、下画面の内容を切り替えて表示できます。電話モードのデモには以下のメニューがあります。なお、AudioInfo 以外のメニューではボタンが下画面のメニュー操作に利用されるため、上画面の操作は出来ません。

表 7-4 サブメニューの内容

表示	内容
AudioInfo	オーディオストリーミングの情報を表示
VAD Info	VAD の情報表示と、設定の変更
Audio Debug	オーディオストリーミングデバッグメニュー(デバッグビルド版のみ)

## 7.3 トランシーバーモードのテストプログラム

GAME CONNECTED MODE で Start Transceiver を選択すると以下のような画面が表示され、トランシーバーモードのデモを実行することができます。

```
=====
Transceiver mode
> Set trans Server to aid:0
  Set trans Server to aid:1
  Set trans Server to aid:2
  Set trans Server to aid:3
  Set trans Server to aid:4
  Set trans Server to aid:5
  Set trans Server to aid:6
  Set trans Server to aid:7
  Startup Trans Server
  return
=====
```

図 7-3 トランシーバーモード

トランシーバーモードではどれか一台がサーバになる必要があります。サーバになるマシン上で”Startup Trans Server”を選択し、残りのクライアントマシン上では、サーバに設定したマシンの aid を選択してください。

サーバ・クライアントを設定すると、L ボタンで会話を開始することができます。

その他の操作は電話モードと同様です。なお、トランシーバーモードでは 8bit 8KHz Raw フォーマットと G.711 u-Law フォーマットは利用できません。

## 7.4 カンファレンスモードのテストプログラム

GAME CONNECTED MODE で Start Conference を選択すると以下のような画面が表示され、カンファレンスモードのデモを実行することができます。

```
=====
Conference mode
> Add aid:0
  Add aid:1
  Add aid:2
  Add aid:3
  Add aid:4
  Add aid:5
  Add aid:6
  Add aid:7
  return
=====
```

図 7-4 カンファレンスモード

十字ボタンでカーソルを移動して A ボタンを押すことで指定した aid のクライアントをカンファレンスに追加・削除することができます。この作業はカンファレンスに参加するすべてのクライアントで実行する必要があります。カンファレンスに追加後は、特別な操作なしに会話をすることが可能です。

その他の操作は電話モードと同様です。なお、X/Y ボタンにてオーディオフォーマットを手動で変更できますが、プログラム上では参加者が3人までは 4bit ADPCM、4 人で 3bit ADPCM と自動的に変更します。また、カンファレンスモードでは 8bit 8KHz Raw フォーマットと G.711 u-Law フォーマットは利用できません。

## 8 特記事項

### 8.1 DWC 関数の呼び出し

---

VoiceChat ライブラリでは、以下の関数内で NitroDWC の送信関数を呼び出しています。アプリケーション内部でマルチスレッドを用いて DWC 関数を利用する場合は注意してください。

- VCT\_Main / VCT\_HandleData
- VCT\_Request/VCT\_Response/VCT\_Contact/VCT\_Release

### 8.2 メモリ使用量

---

VoiceChat ライブラリは、内部でメモリの確保関数を呼び出すことはありません。VoiceChat ライブラリ初期化時にバッファとサイズを指定することで利用するメモリの量を調整できます。なお、ライブラリが制限する最大値は 72 パケット(1パケットあたり1144バイト、合計約 80KB)です。

また、ライブラリが占有している静的メモリ領域は 10KB、ライブラリの常駐部分のコードが占有するサイズは 21KB(いずれも FINAL ROM ビルドでの値)です。

© 2006 Nintendo

任天堂株式会社の許諾を得ることなく、本書に記載されている内容の一部あるいは全部を無断で複製・複写・転写・頒布・貸与することを禁じます。