株式会社ＮＴＴドコモ

雑談対話 SDK for iOS™ 利用ガイド

株式会社ＮＴＴドコモ

Version : 2.0.0
作成日: 2015/02/24

改訂履歴

日付	区分	項番	変更内容	版数
2014/07/30		All	新規作成	1.0.0
2015/02/24	変更	3.2	以下のライブラリバージョンを変更 ・docomo SDK 共通ライブラリ (1.0.1) ・雑談対話 SDK ライブラリ (2.0.0) ・OAuth SDK ライブラリ (1.1.2)	2.0.0
	追加	4.2	雑談対話（認証あり）の API を追加	2.0.0
	追加	6	OAuth 認証エラーコードを追加	2.0.0
	追加	7.3	雑談対話（認証あり）のサンプルコードを追加	2.0.0

目次

1. はじめに

1.1. 本書の内容と目的

本ドキュメントは雑談対話 API を利用した iOS アプリケーション開発用の iOS SDK の概要を記載したものです。

1.2. 商標

iOS は、Cisco Systems, Inc. の商標または登録商標です。

2. 雑談対話 SDK の概要

本 SDK は、雑談対話 API を利用した iOS アプリケーション開発に必要となる各種ライブラリやサンプルコードを提供します。

3. 動作環境および設定

3.1. サポートバージョン

本 SDK は iOS 7.0 以降で使用可能です。
また本 SDK は iOS 7.0 にて動作確認を行っています。

3.2. ライブラリ

本 SDK 利用には下記のライブラリが必要となります。

ユーザ認証が必要となる API を利用する場合は、OAuth による認証処理が必要となります。OAuth SDK は以下の URL からダウンロードできますので、ご利用ください。

https://dev.smt.docomo.ne.jp/?p=docs.common.index

表 1 ライブラリー覧

ライブラリファイル名	ライセンス	ライブラリ概要
libdocomo-common-ios-sdk-1.0.1.a	-	docomo SDK 共通ライブラリ
libdocomo-dialogue-ios-sdk-2.0.0.a	-	雑談対話 SDK ライブラリ

表 3-2 OAuth SDK ライブラリー覧

ライブラリファイル名	ライセンス	ライブラリ概要
libdocomo-oauth-ios-sdk-1.1.2.a	-	OAuth SDK ライブラリ

3.3. ヘッダファイル

本 SDK 利用には下記のヘッダファイルが必要となります。これらのヘッダファイルを iOS アプリケーション開発プロジェクトにて import してください。

表 3-3 雑談対話 SDK ヘッダーファイル一覧

ヘッダファイル名	ヘッダファイル概要
Dialogue.h	雑談対話問い合わせ処理クラス
DialogueError.h	雑談対話エラー情報クラス
DialogueRequestParam.h	質問文問い合わせリクエスト用データクラス
DialogueResultData.h	回答データクラス
AuthApiKey.h	API キー認証情報クラス
SdkError.h	SDK エラー情報クラス

表 3-4 OAuth SDK ライブラリヘッダファイル一覧

ヘッダファイル名	ヘッダファイル概要
OAuth.h	OAuth 認証実施に必要なヘッダファイル
OAuthToken.h	OAuth 認証成功結果取得に必要なヘッダファイル
OAuthError.h	OAuth 認証失敗結果取得に必要なヘッダファイル

4. 利用可能な API

本 SDK で使用可能な API の処理概要、ガイドラインを示します。

4.1. 雑談対話

- 処理概要
  雑談対話 API は、ユーザの発話(テキスト)に対して返答を行う機能です。
  発話(テキスト)を株式会社 NTT ドコモの提供する雑談対話サーバ上の API に送信することで発話に対する返答を取得できます。

4.2. 雑談対話（認証あり）

- 処理概要
  雑談対話 API は、ユーザの発話(テキスト)に対して返答を行う機能です。
  発話(テキスト)を株式会社 NTT ドコモの提供する雑談対話サーバ上の API に送信することで発話に対する返答を取得できます。

5. API リファレンス

本 SDK の API リファレンスの詳細情報については AppleDoc を参照してください。
各リクエストパラメータに設定可能な値については「雑談対話API インターフェース仕様書」をご確認ください。
「雑談対話API インターフェース仕様書」は以下のURL で確認することができます。
https://dev.smt.docomo.ne.jp/?p=docs.api.page&api_docs_id=3

6. SDK エラーコード一覧

本 SDK で発生したエラー内容はエラーコードを取得することで確認できます。エラーコードと詳細を示します。

表 5 エラーコード一覧

エラーコード	エラー内容	発生例
100501	パラメータ不正	必須パラメータの設定漏れ
100502	認証失敗	認証情報の設定間違い
100503	処理失敗	SDK 内部処理失敗
100504	サーバ接続エラー	サーバ接続失敗
100505	サーバー時利用不可	サーバの輻輳規制時
100506	その他サーバエラー	サーバ側内部処理失敗
100507	レスポンスデータエラー	サーバ側から異なるフォーマットのデータが返された場合
100508	OAuth 認証失敗	アクセストークンの期限切れ
100509	OAuth 認証失敗	アクセストークンフォーマットエラー 認証サーバエラー

※レスポンスデータエラーが発生した場合は SDK の最新バージョンを取得してください。

7. サンプル

本 SDK 使用時のサンプルを示します。

7.1. 初期化処理

本 SDK 使用のため、開発者ポータルから取得した API キーの設定を行います。

```
// 開発者ポータルから取得したAPIキーの設定
[AuthApiKey initializeAuth:"ApiKey"];
```

7.2. 雑談対話要求

発話「こんにちは」を問い合わせるリクエストの例です。
任意の文字列を context に設定することで会話をつなげることができます。

```
//雑談対話要求処理クラスを作成
Dialogue * dialogue= [[Dialogue  alloc] init];
//雑談対話要求リクエストデータクラスを作成してパラメータをset する
DialogueRequestParam * param = [DialogueRequestParam  alloc] init];
param.utt = @"こんにちは";
// context には任意の文字列を設定する。
param.context = @"aaabbbccc111222333";
//雑談対話要求処理クラスにリクエストデータを渡し、受信完了時処理を登録する
DialogueError * sendError = [dialogue request: param
onComplete:^( DialogueResultData *resultData)
{
    // レスポンス受信時の処理を行います
    NSLog(@"回答=%@", resultData.utt);
    //対話を継続するために context には任意の文字列を設定する。
    param.utt = @"いい天気ですね";
    param.context = @"aaabbbccc111222333";
    // 雑談対話を続ける

} onError:^( SdkError * receiveError) {
    NSLog(@"受信エラー=%ld",(long) receiveError.code);
}];
if(sendError){
    NSLog(@"送信エラー=%ld",(long) sendError.code);
}
```

発話「こんにちは」を問い合わせるリクエストの例です。
サーバから受信したcontext を設定することで会話をつなげることができます。

```
// 雑談対話処理データクラス
DialogueRequestParam * param;
Dialogue * dialogue;
DialogueResultData * resultData;
DialogueError * requestError;

/** 初期化処理 */
-(id) init {
    [AuthApiKey initializeAuth: @"xxxxxxxxxx"];
    param = [DialogueRequestParam alloc] init];
    dialogue = [Dialogue alloc] init];
    return self;
}
```

```
/** ボタンを押したらリクエストを送信する処理  */
- (IBAction)dialogueRequest:(id)sender {
    // ユーザーの発話を設定する
    param.utt = @"こんにちは";
    //雑談対話要求処理クラスにリクエストデータを渡し、受信完了時処理を登録する
    DialogueError * sendError = [dialogue request: param
    onComplete: ^( DialogueResultData *resultData)
    {
        // 受信完了時処理へ
        [self receiveDialogueRequest:resultData];
    } onError:^( SdkError  *receiveError) {
        NSLog(@"受信エラーコード=%ld",(long) receiveError.code);
        NSLog(@"受信エラー情報=%@", receiveError.localizedDescription);
    }];
    if (sendError) {
        NSLog(@"送信エラーコード=%ld",(long) sendError.code);
        NSLog(@"送信エラー情報=%@", sendError.localizedDescription);
    }
}
```

```
/** 受信完了時処理 */
-(void) receiveDialogueRequest: (DialogueResultData *) resultData
{
    NSLog(@"回答=%@", resultData.utt);
    // 対話を継続するために context にはサーバから受信した文字列を設定する。
    param.context = resultData.context;
    // サーバー発のしりとりモードを継続するため受信したモードを設定する
    param.mode = resultData.mode;
    // 任意の文字列を会話に設定する
    param.utt = @"しりとりしましょう。";
    // ボタンを押したらリクエストを送信する処理へ
}
```

7.3. 雑談対話（認証あり）

発話「こんにちは」を問い合わせるリクエストの例です。
アクセストークン、context を設定することで会話をつなげることができます。

```
/** ボタンを押したらリクエストを送信する処理  */
- (IBAction)dialogueRequest:(id)sender {
    // ユーザーの発話を設定する
    param.utt = @"こんにちは";
    accessToken =  @"xxxxxxxxxx";
    //雑談対話要求処理クラスにリクエストデータを渡し、受信完了時処理を登録する
    DialogueError * sendError = [dialogue request: param
    accessToken:  accessToken
    onComplete: ^( DialogueResultData *resultData)
    {
        // 受信完了時処理へ
        [self receiveDialogueRequest:resultData];
    } onError:^( SdkError  *receiveError) {
        NSLog(@"受信エラーコード=%ld",(long) receiveError.code);
        NSLog(@"受信エラー情報=%@", receiveError.localizedDescription);
    }];
    if (sendError) {
        NSLog(@"送信エラーコード=%ld",(long) sendError.code);
        NSLog(@"送信エラー情報=%@", sendError.localizedDescription);
```