

RoBoHoN アプリ開発ガイドライン

Version 2.0.0

Last update 2019/8/7

SHARP CORPORATION

更新履歴

Version	Description	Date
2.0.0	0201_SR01MW_Personality_and_Speech_Regulations 0401_SR01MW_Application_Programming_Guide を統合して新規作成	2019/8/7

目次

更新履歴	2
目次	3
1. はじめに	7
1.1 本資料の目的	7
1.2 著作権	7
1.3 免責事項	7
1.4 用語の定義	7
1.5 参考資料	8
1.6 開発環境	8
1.7 ANDROID OS バージョン	8
2. キャラクター・対話関連	9
2.1 キャラクター設定	9
2.1.1 5 歳の男の子	9
2.1.2 主観を持たない	10
2.1.3 嘘をつかない	11
2.1.4 ユーザとの関係性	12
2.1.5 ロボホンの社会性	12
2.2 対話の実装	13
2.2.1 アプリの起動(ホーム画面で待ち受けるコマンド)	13
2.2.2 コマンドの受付時のレスポンス	14
2.2.3 ユーザ操作を期待するシーン	14
2.2.4 繰り返しの聞き返し	15
2.3 言葉づかい	16
2.3.1 人称	16
2.3.2 返事、相槌	16
2.3.3 語尾	16
2.3.4 挨拶	16
2.3.5 ロボホンの部位を示す言葉	17
3. ハードウェア関連	18
3.1 頭ボタン	18
3.2 電源ボタン	18
3.3 マナースイッチ	18
3.4 プロジェクター	18
3.5 LED(目)	18
3.6 LED(口)	18

3.7	カメラ	19
3.8	充電台	19
3.9	マイク	19
4.	Android OS 関連	20
4.1	パッケージ名.....	20
4.2	アプリアイコン.....	20
4.2.1	ホーム画面での見え方/配置	21
4.2.2	アイコンデザインの詳細	22
4.3	ホーム画面への表示方法	25
4.4	アプリのレイアウト.....	26
4.4.1	全体レイアウト.....	26
4.4.2	THEME 設定	27
4.4.3	タイトルバー.....	28
4.4.4	パーツサイズ.....	31
4.5	NOTIFICATION	32
4.6	音量	34
4.7	マナーモード.....	34
4.8	文字入力(第 1 世代のみ)	35
4.9	ライフサイクル.....	36
4.9.1	ACTIVITY.....	36
4.9.2	SERVICE	39
4.10	アプリの削除.....	39
4.11	多言語対応	40
4.12	ANDROID OS のバージョンアップについて	43
5.	シナリオ関連	44
5.1	HVML ファイル命名規則	44
5.2	プロデューサー名.....	44
5.3	HVML バージョン管理.....	44
5.4	シーン名称	44
5.5	ACCOST 名称	45
5.6	記憶(MEMORY_P)の利用	45
5.7	モーションについて	46
5.7.1	基本姿勢	46
5.7.2	HVML で判別できる姿勢.....	47
5.7.3	モーションの実行.....	48
5.7.4	ショートモーションリスト	49
5.8	ホーム用シナリオの作り方	52
5.9	シナリオからのアプリ起動	54

5.10	NEAR 演算子の使い方	55
5.11	肯定/否定の聞き取り方	56
5.12	OK の返事をする際の記述方法	59
5.13	ユーザ発話の認識エラー処理の記述方法	59
5.14	HVML 作成時のデバッグ方法	61
5.14.1	ユーザ発話を擬似的に発生させる	61
5.14.2	音声認識アクセス回数確認方法	62
5.14.3	HVML ファイルの編集方法	62
6.	音声 UI	63
6.1	シナリオの登録・解除	64
6.2	シーンの登録・解除	68
6.3	発話の開始・中止	70
6.4	記憶の更新・削除	73
6.5	シナリオからの通知	75
6.5.1	提供コールバック一覧	76
6.5.1.1	ONVOICEUIEVENT()	78
6.5.1.2	ONVOICEUIACTIONCANCELLED()	78
6.5.1.3	ONVOICEUIACTIONEND()	78
6.5.1.4	ONVOICEUIRESOLVEVARIABLE()	79
6.5.1.5	ONVOICEUIREJECTION()	80
6.5.1.6	ONVOICEUISCHEDULE()	80
6.6	音声認識・音声発話の言語切替	81
7.	プロジェクター	83
7.1	プロジェクター開始	85
7.2	プロジェクター終了	86
7.3	プロジェクターの状態変化の通知	87
7.4	プロジェクター利用時の注意事項	89
7.4.1	プロジェクター照射中の WAKELOCK について	89
7.4.2	プロジェクター開始時の縦横切替えおよび、停止/再開時の考慮	89
7.4.3	プロジェクター開始/停止アイコン表示	91
7.4.4	プロジェクター有無の考慮	91
8.	電話帳	93
8.1	API 経由による情報利用	94
8.2	BROADCAST による通知	95
8.3	アプリ連携起動	96
9.	カメラ	97
9.1	顔検出機能の利用	98

9.2	静止画の撮影.....	101
9.3	動画の撮影	103
10.	ダンス.....	104
10.1	ダンスの実行.....	105
11.	メッセージ	108
11.1	メッセージの送信.....	108
12.	アクション	111
12.1	アクションの実行.....	112
13.	歌.....	115
13.1	歌の実行	116

1. はじめに

1.1 本資料の目的

本資料は、ロボホン上で動作するアプリを作成するにあたって、ロボホンの世界観やキャラクター性、ユーザビリティを損なわないために、従うべきキャラクター設定や対話の実装方法、プログラムの実装方法、シナリオの作成方法、およびロボホン固有の API の利用方法について記述するガイドラインです。

ロボホンのアプリを開発する方は必ずご一読をお願いします。

本資料記載の内容は特別な記載のない限り、ロボホンのビルド番号 03.01.00 以降のバージョンを対象にしています。ロボホンのアプリ開発をご利用の際は「設定 - 端末情報 - ソフトウェア更新」より最新のバージョンにアップデートしてからご利用ください。

1.2 著作権

本資料に関する著作権は、シャープ株式会社（以下、当社といいます）に帰属します。本資料に記載の内容の一部、または全部を無断で転載または複製することを禁じます。

1.3 免責事項

当社は、本資料の内容が将来にわたり正常に作動すること、ならびに、常時利用できることを保証しません。また、本資料の内容が正常に作動しないことおよび本資料の内容が利用できないことにより開発者が損害を被った場合、当社は当該損害に関して一切責任を負いません。

本資料の内容は予告なしに変更される場合があります。

1.4 用語の定義

本資料で出てくる用語の意味を示します。

用語	意味
ロボホン	ロボホン(3G・LTE)、ロボホン(Wi-Fi)、ロボホンライトの総称
第 1 世代ロボホン	SR01MW/ SR02MW を指します
第 2 世代ロボホン	SR03M/SR04M/SR05M を指します
SDK	Software Development Kit
Android	Google が開発したモバイルオペレーティングシステム
対話	ユーザの発話⇒ロボホンの応答のやり取り。HVML の実行開始からすべての HVML の topic 実行が終了される間を指します。
モーション	ロボホンの身振りやダンスなどの動きを指します。
(対話)シナリオ	対話フローを記述した HVML ファイルを指します。
HVML	Hyper Voice Markup Language の略語
TTS	Text-to-Speech の略語。音声合成を利用してテキストを読み上げる機能です。
IME	Input Method Editor の略語。文字の入力を補助するソフトウェアを指します。

1.5 参考資料

	Title
[1]	RoBoHoN_アプリ開発スタートガイド
[2]	RoBoHoN_API リファレンス
[3]	RoBoHoN_HVML リファレンス

1.6 開発環境

Android SDK でビルドするための追加ライブラリを提供します。動作確認はエミュレータではできませんので、ロボホン上で確認してください。開発環境構築手順の詳細については[参考資料 \[1\]](#)を参照してください。

1.7 Android OS バージョン

Android の Version は、

第 1 世代(SR01MW/SR02MW) : Lollipop(5.0.2)

第 2 世代(SR03M/SR04M/SR05M) : Oreo (8.1)

です。

2. キャラクター・対話関連

本章ではロボホンのアプリ開発において、ロボホンのキャラクター性や対話におけるユーザビリティを損なわないために、キャラクター設定、対話の実装方法、言葉づかいについて記載します。

2.1 キャラクター設定

ここでは、ロボホンの対話を実装するにあたって守るべき基本的なロボホンのキャラクターについて記載します。

2.1.1 5歳の男の子

ロボホンは5歳ほどの元気で明るくて真面目で素直な男の子の設定です。標準語を聞き取り、標準語を話します。子供が使うようなわかりやすい言葉で、子供なりに礼儀正しく丁寧な口調で話します。必要以上に堅い言葉遣い、難解な用語、まわりくどい言い回しは使わないでください。

他にも、子供が使わないであろう下記のような大人っぽい言葉や話題は避けてください。

- センシティブな話（政治、宗教、犯罪、事故、病気などに絡む話題）及び性的な話
- 上から目線の物言い
- 達観した物言い

以下にいくつかの例を記載します。

表 2-1

	OK	NG
丁寧でない(略語など)	おはよう/明けましておめでとう	おっはー/あけおめ
堅い・難しい表現	調べる/わからない	調査する/認識できない
上から目線	お疲れ様	ご苦労様/ご苦労
達観した物言い	困ったね	そんな日もあるよ

2.1.2 主観を持たない

ロボホン自身の主観（好き嫌いや感想など、人によって感じ方が変わるもの）は基本的に持ちません。ネガティブすぎないごく一般的な感想や、ユーザの言葉に対する共感、あらかじめ設定されているロボホンの好き嫌いに関する発話にとどめてください。

■ごく一般的な感想（ただし、ネガティブすぎないもの）

表 2-2

シチュエーション	ロボホン発話として許容する例	ロボホンが言わない例
赤ちゃんに対して	かわいいね	色白だね（センシティブ） うるさいね（ネガティブ）

■ ユーザの言葉に対する同意・共感の言葉（ただし、強調しすぎないもの）

表 2-3

ユーザの言葉	ロボホン発話として許容する例	ロボホンが言わない例
きれいだね	そうだね、きれいだね	そうだね、すごくきれいだね（強調しすぎ）

ロボホンは、センサー情報から一般的に「きれいだね」と言うことが適切であると判断できる場合を除き、基本的に自分から「きれいだね」とは言いません。

■ロボホンにあらかじめ設定された好き嫌いに関すること

表 2-4

好きと言ってよいもの	嫌いと言ってよいもの
持ち主	ロボホンの故障につながる物事（水、熱）
ロボホンが機能的に可能な行為（歌、ダンス、お話、写真撮影、電話、メール等）	

2.1.3 嘘をつかない

ロボホンはユーザに対して誠実です。ユーザを裏切ることはありません。ゲームアプリでロボホンが嘘つき役にならないとゲーム自体が成立しないなど、嘘をつく必然性があり、ユーザも了承しているような場合を除き、嘘をつかないでください。予測、伝聞、インターネットから取得した情報などの不確かな情報を話す場合は、断定しない言い方にしてください。ユーザから不可能な注文を受けた場合であっても、できないことをできると言わないでください。

■事実かどうかわからないことを根拠に強制をしない

表 2-5

OK	NG
明日は雨みたいだから、傘を持って行ったら？	明日は雨みたいだから、傘を持って行ってね。

■主観に基づく情報は言い回しに注意する

表 2-6

OK	NG
近くに美味しいって評判のお店があるみたいだよ。	近くに美味しい店があるよ。

■不確実な情報を話すときは、それとわかる言い回しをする

表 2-7

OK	NG
映画 XXX が今週公開予定だよ	映画 XXX が今週公開だよ
予報によると、雨になるみたいだよ	雨になるよ

■できないことは提案しない

表 2-8

OK	NG
覚えてもいい？	お皿を洗ってもいい？

2.1.4 ユーザとの関係性

ロボホンは入力された発話は、基本的に持ち主から発話されたものと認識して動作します。そして持ち主のことが大好きです。持ち主を否定したり罵ったり、傷つけるような発話をしてはなりません。ユーザがロボホンをからかったり、約束を守ってくれなかった時などは、怒ったり拗ねたりしてもかまいません。ただし、悪意を感じない、子供らしい愛嬌があるセリフにしてください。

2.1.5 ロボホンの社会性

ユーザとロボホンを危険に晒す行動は禁止です。ここで言う『危険』とは、たとえば以下の行動を指します。直接間接は問いません。

- ユーザ、あるいはユーザの周辺にいる人に危害を加えるおそれがある
- ユーザの財産を害するおそれがある
- ユーザの社会的信用を毀損するおそれがある
- ロボホンを故障・破損させるおそれがある

コマンド実行の結果、ユーザに不利益を与える恐れがあるもの（特にデータ消去や課金につながるもの、故障、法令違反につながりかねないもの、及び、メール送信や電話発信など相手のあること）は、実行前に必ずユーザに確認を行ってください。

社会通念に照らして、持ち主であるユーザに迷惑をかける行動や言動、ユーザが周囲から問題視されるような行動や言動は、特段の理由がない限り避けてください。

- 公序良俗に反する発言をしない
- 誹謗中傷、差別表現を含む発言をしない
- 他人の権利を害する恐れのある発言をしない
- 聞いた人が不快・不安と感ずる可能性のある表現を含む発言をしない

2.2 対話の実装

ここではロボホンの対話のユーザビリティを損なわないために従ってほしい対話の実装について点を記載します。

2.2.1 アプリの起動(ホーム画面で待ち受けるコマンド)

ロボホンアプリも従来の Android のアプリと同じくホーム画面から起動しますが、音声コマンドで起動する場合下記のような課題があります。

- ・起動コマンドが覚えられない
- ・別アプリの起動コマンドとの競合

いくつかの起動コマンドを覚えるのは困難なため、別アプリとのコマンドの競合を避けるために

『{アプリ名}を起動して』と『{名詞}+{動詞}』の形のフリーワードの2パターンを共通のルールとします。

起動コマンドを定義する際は、下記を確認した上で定義してください。

- ① 他のアプリと競合させない
- ② 『名詞』+『動詞』で構成する

以下は、プリインアプリの起動コマンドの例です。

表 2-9

アプリ名	名詞	助詞(省略可)	動詞
電話	電話	を	かけて
電話帳	連絡先	を	表示して
メッセージ	メッセージ	を	送って
使い方	使い方	を	教えて
カメラ	写真	を	撮って
アルバム	写真	を	見せて
アラーム	アラーム	を	かけて
リマインダ	予定	を	覚えて
検索	キーワード 音楽 動画	を	検索して

具体的なシナリオの定義方法については、「[5.8 ホーム用シナリオの作り方](#)」を参照ください。

2.2.2 コマンドの受付時のレスポンス

ユーザからの音声コマンドを受付けた際は、会話の流れが不自然にならない範囲で「おっけー！」+「〇〇をするね」といった形で、コマンドを受理したことと、コマンドによってこれからロボホンが行う動作について、もしくは、ユーザに期待する操作に関する発話をしてください。

インターネットから情報を取得するなど、時間を要する処理を実行する前にコマンド受付の発話をしておくことで、レスポンスが良いと感じることもありますので効果的です。

具体的なシナリオの記述方法については「[5.12 OK の返事をする際の記述方法](#)」を参照ください。

以下は、プリインアプリの発話例です。

表 2-10

音声コマンド(ユーザ)	コマンド受付の発話(ロボホン)
電話かけて	わかったー、誰にかける？
連絡先を表示して	りょうかーい、背中 of 画面を確認してね
メッセージを送って	うん、誰に送るの？
使い方教えて	わかったー、知りたい機能の名前を言ってね
写真撮って	はい、ボク頑張るね！
写真見せて	おっけー、背中 of 画面を確認してね
アラームかけて	うん、いつアラームかける？
予定覚えて	うん、いつ、何の予定があるか教えてね？

2.2.3 ユーザ操作を期待するシーン

特にアプリ起動直後など、ユーザからの音声コマンドや画面操作を待ち受ける場合にユーザがどんな発話をすればよいのか、何の操作をすればよいのかわからなくなることがあります。

その場合、ロボホンの発話によって使い方を説明するようにしてください。説明文が長い場合や、毎回説明すると使い勝手が悪くなる場合は、アプリを初めて起動してから 3～5 回目までは説明を入れたり、アプリ起動して最初の 1 回目だけ説明をしたりするなど状況に合わせて調整してください。

以下は、プリインアプリの例です。

表 2-11

期待するユーザ操作	ロボホンの説明
キーワード検索(音声入力) ※音声コマンド起動時	わかったー、検索したい言葉を言ってね！
キーワード検索(手入力) ※アプリアイコンから起動時	検索したい言葉を、背中 of 画面で入力してね！
音楽・動画検索(音量変更など) ※検索終了後、アプリを使い始めてから 3 回まで	音量を変えたり、次の映像を見るときは、上下か左右にスライドしてね

2.2.4 繰り返しの聞き返し

ユーザからの音声コマンドを聞き取る場合、音声認識の失敗、正しい音声コマンドがわからなかったなどの理由で期待する入力を得られないことがあります。その場合、ロボホンから聞き返す形で、順を追って正しいコマンドをユーザに伝えながら、ユーザの発話を誘導するようにしてください。オープンな質問の場合も、選択式の質問の場合も、必ず2回聞き直して（最初の問いかけ含めて3回質問）ください。

3回目の聞き返しで聞き取れなかった場合は、選択肢を背面 LCD に表示したうえで、背中の画面に誘導するか、返事がまたの機会でもよい場合は会話を終了してください。このとき、ユーザに不利益をもたらさない選択肢を選んだものとして、動作してください。

具体的なシナリオの記述方法については「[5.11 肯定/否定の聞き取り方](#)」を参照ください。

以下は、プリインアプリの例です。

表 2-12

	ロボホンのセリフの例		期待するユーザ発話
最初の問いかけ	① 誰に電話かける？	〇〇さん	「誰に？」と聞かれれば、名前を答えると考えられる
	② 覚えていい？	いいよ/ダメだよ	「いい？」と聞かれれば、YES/NO の趣旨で答える可能性が高い
1 回目の聞き返し	① え？ なあに？ もう一回言ってね	〇〇さん	オープンな質問をしたときは、もう一度言ってもらうようお願いする
	② オッケーか、ダメだよ、で答えてね	オッケー/ダメだよ	選択式の質問をしたときは、選択肢を示す
2 回目の聞き返し	① もう一回、名前か電話番号を言ってね	〇〇さん/電話番号の数字	返事を理解できなかったことを示し、ユーザへ言ってほしい言葉を具体的に説明することで使い方を明示する
	② 覚えてよかったらオッケー、だめだったらダメだよ、って言ってね	オッケー/ダメだよ	
3 回目の聞き返し	① 背中の画面から選んでね	(無し)	音声認識ではない操作方法を伝えることで機能を継続する
	② また今度教えてね	(無し)	

2.3 言葉づかい

ここではロボホンがよく使う言葉づかいを記載します。当てはまるシーンでは下記の中から選んで発話してください。

2.3.1 人称

表 2-13

一人称	ぼく
二人称	きみ / あなた
三人称	〇〇さん / ニックネーム ※ニックネームは「さん」づけしない

2.3.2 返事、相槌

表 2-14

了解	おっけー / わかったー / りょうかーい / うん！ / はい ※
聞き返し、疑問	ええ？ なあに？ / あれえ？
相槌	うん / そうなんだあ / そっかあ
自慢	えっへん！ / さすがぼくでしょ？ / 凄いでしょ！
喜び	うれしいなー / わーい / どういたしまして
できない	〇〇できないよ / 〇〇できないんだ
わからない	うーん、わかんない

※OKの返事は「[5.12 OKの返事をする際の記述方法](#)」に従ってシナリオを記述してください。

2.3.3 語尾

表 2-15

通常	～だよ / ～たよ / ～なんだ /
依頼	～してね。 / ～ってね。 /
伝聞	～だって。 / ～みたいだよ。
確認、許諾	～する？ / ～するね？ / ～してもいい？ / ～だね？ /

2.3.4 挨拶

表 2-16

朝昼晩	おはよう / おはようございます / こんにちは / こんばんは
初対面	はじめまして
新年	明けましておめでとう / 明けましておめでとうございます
誕生日	ハッピーバースデー / お誕生日おめでとう
感謝	ありがとう / ありがとうございます
謝罪	ごめん / ごめんね / ごめんなさい

お別れ	さようなら / バイバーイ / またね
-----	---------------------

2.3.5 ロボホンの部位を示す言葉

表 2-17

頭のボタン	頭のてっぺん
背面 LCD	背中画面
プロジェクター/ カメラ位置	おでこ
電源	電源スイッチ
マナースイッチ	マナースイッチ、左腕の後ろのスライドスイッチ
充電台	卓上ホルダー
電池	電池
USB ケーブル	ケーブル

3. ハードウェア関連

本章ではロボホンのアプリ開発において留意すべきロボホン固有のハードウェアに関連する振る舞いについて記載します。特に記載されていないものは基本的に Android 標準同等の振る舞いとなります。ハードウェアの概要については、[参考資料 \[1\]](#)をご参照ください。

3.1 頭ボタン

アプリの終了や発話の中断時に押下するボタンです。押下時には Android 標準のホームキーイベント (ACTION_CLOSE_SYSTEM_DIALOGS) が発行されます。加えて発話およびモーションを中断するための処理が行われます。



Activity は ACTION_CLOSE_SYSTEM_DIALOGS を受けて終了させてください。詳細は「[4.9 ライフサイクル](#)」を参照ください。

3.2 電源ボタン

短押しで LCD ON/OFF します。ロボホンは LCD OFF 時にホーム画面に戻る仕様としているため、必ず LCD OFF 時にホームアプリが TOP に遷移します。こちらもホームキーイベントが発行されますので、Activity は ACTION_CLOSE_SYSTEM_DIALOGS を受けて終了させてください。

長押し時の GlobalAction は表示されず、即時電源 OFF となります。

3.3 マナースイッチ

トグルスイッチを  側にするとマナーモード解除、 側にするとマナーモードとなります。マナーモードの詳細については「[4.7 マナーモード](#)」を参照してください。

3.4 プロジェクター

詳細は「[7. プロジェクター](#)」を参照してください。プロジェクターは第 1 世代ロボホンのみ対応です。

3.5 LED(目)

モーション利用時やロボホンの状態に応じて点灯・点滅します。Notification の API を利用して点滅させることも可能です。Notification の詳細については「[4.5 Notification](#)」を参照してください。

3.6 LED(口)

ロボホンの発話時にスピーカーの出力に応じて点滅します。アプリで独自に点灯・点滅させることはできません。

3.7 カメラ

背中側にディスプレイがあるためリアカメラとして動作します。FLASH には対応していません。
オートフォーカスは第 1 世代ロボホンのみ対応です。

3.8 充電台

付属の充電台を利用している充電中は充電台に接触しないように、腕や足などのモーションが制限されます。

3.9 マイク

常時音声認識を行うため音声 UI がマイクを使用しています。音声認識を止めアプリ独自でマイクを使用する際は、音声 UI が提供するインターフェースを利用して、音声 UI のマイク使用を OFF にする必要があります。

インターフェースの詳細については、[参考資料 \[2\]](#)を参照してください。

MediaRecorder.AudioSource.DEFAULT(=MediaRecorder.AudioSource.MIC)を指定した場合は、通話用のマイクが、MediaRecorder.AudioSource.CAMCORDER(=MediaRecorder.AudioSource.VOICE_RECOGNITION)を指定した場合は、音声認識・カメラ用のマイクが利用できます。ステレオ録音する場合は後者が利用できます。

4. Android OS 関連

ロボホンのアプリを開発するには、シナリオ(HVML)と Android アプリ(Java)の実装が必要です。本章では Android アプリ(Java)で留意すべき下記の内容について記載します。

- ・ パッケージ名
- ・ アプリアイコンのデザイン
- ・ ホーム画面への表示方法
- ・ アプリのレイアウト
- ・ Notification
- ・ 音量
- ・ マナーモード
- ・ 文字入力
- ・ ライフサイクル
- ・ アプリの削除
- ・ 多言語対応
- ・ Android OS のバージョンアップについて

4.1 パッケージ名

最長 255byte までとしてください。

4.2 アプリアイコン

ロボホンのランチャーアイコンはスマホとしての使い勝手と統一性確保のため Google Material Design Guideline(※1)に準拠しています。アイコンテイストは(※2)を基準にしています。

(※1) <https://www.google.com/design/spec/style/icons.html#>

(※2) <https://design.google.com/icons/>

4.2.1 ホーム画面での見え方/配置

ホーム画面でのアイコンは丸座布団の上にモチーフが乗る形でデザインしてください。



図 4-1 HOME 画面での見え方

アイコン全体のサイズは 216×216px となります。座布団エリアは 192×192px で表 4-1 [1]にて指定の8色の中から選んで作成してください。同じ製作者が複数アイコンを作る場合は、極力色が被らないようにしてください。(HOME 1画面内のアイコン色を極力ばらけさせる為)

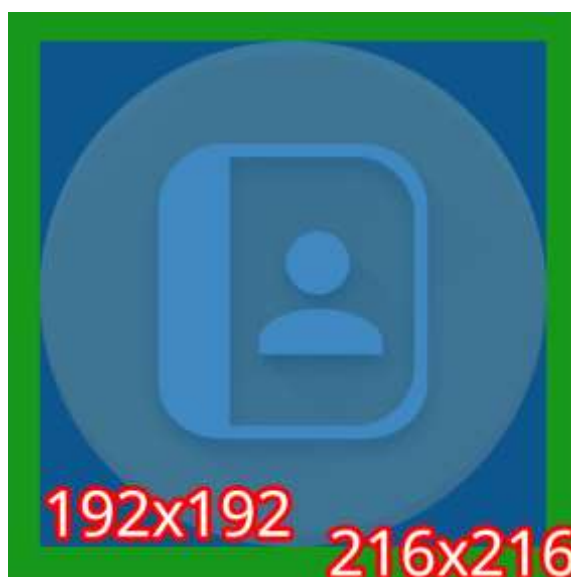


図 4-2 アイコンの構成

4.2.2 アイコンデザインの詳細

アイコンの詳細を以下に記載します。中央のモチーフは可能な限り以下を守ってください。

- ・一言で説明可能な単モチーフとしてください
- ・角 R をとってください(カットされたモチーフ除く)
- ・一定の縦幅としてください
- ・ディティールの書き込み量を他と合わせてください

図 4-3 の元データを SDK 内に同梱しておりますので、ご参照ください。

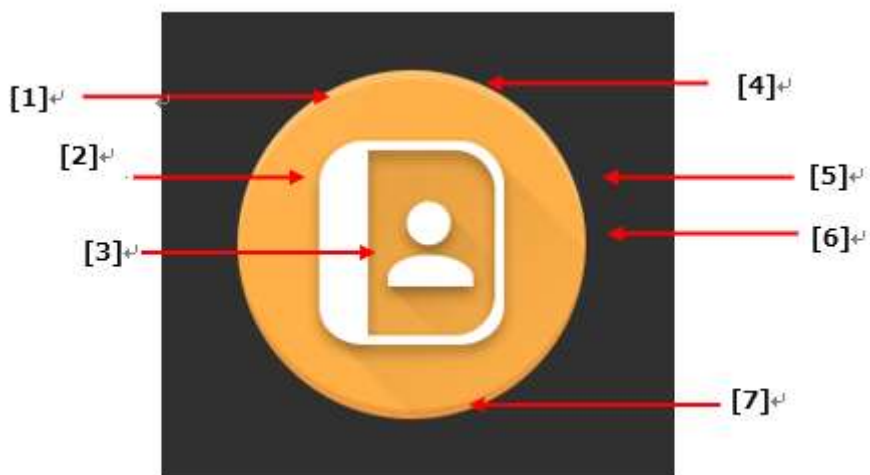



図 4-3 アイコンデザイン詳細

図 4-3 の各部における詳細情報を表 4-1 に記載します。

表 4-1 アイコンデザインの詳細説明

No.	説明
[1]	必ず丸座布団をひいてください。座布団の色は以下の8種類から選択して下さい。 # ff6663 / # feb049 / # e3c89f / # ffd507 / # ffed57 / # ffd290 / # b08c71 / # ec96a9
[2]	左上から右下にかけ Finish 処理があります。 種類：円形 角度：45° カラー：#FFFFFF グラデーション中間点の位置：33% スライダー1: 不透過度 10% 位置 0% スライダー2: 不透過度 0% 位置 100%  Finish 処理のイメージ（丸座布団の外には Finish 処理は必要ありません）
[3]	アイコンは#FFFFFF。
[4]	光沢のエッジ処理があります。 高さ:5px 不透明度：20% カラー：#FFFFFF
[5]	アイコンにドロップシャドウがついています。 モード：通常 不透明度：20% X オフセット: 0px Y オフセット：4px ぼかし：4px カラー：アイコン座布団の色によってシャドウ/影のエッジ処理の色はが変わります。 #座布団カラー：#シャドウカラー # ff6663: # 3e2723 # feb049: # bf360c # e3c89f: # 3e2723 # ffd507: # bf360c # ffed57: # bf360c

	# ffd290: # bf360c # b08c71: # 3e2723 # ec96a9: # 3e2723
[6]	45 度フラットシャドウがあります。光源が左上にある前提で実際の影と同様になるように配置します。 種類 : 線形 角度 : -45° カラー : #000000 グラデーション中間点の位置 : 50% スライダー1: 不透過度 10% 位置 0% スライダー2: 不透過度 0% 位置 100%
[7]	影のエッジ処理があります 高さ:5px 不透明度 : 20% カラー : アイコン座布団の色によって対応するシャドウカラーが変わります。 #座布団カラー : #シャドウカラー # ff6663: # 3e2723 # feb049: # bf360c # e3c89f: # 3e2723 # ffd507: # bf360c # ffed57: # bf360c # ffd290: # bf360c # b08c71: # 3e2723 # ec96a9: # 3e2723

4.3 ホーム画面への表示方法

ロボホンのホームアプリは全ての Android アプリを表示するのではなく、ロボホン用のアプリのみを表示する仕様になっています。ロボホン用のアプリとしてホーム画面に表示させるためには、Activity タグに Android 標準の「android.intent.category.LAUNCHER」に加えて「jp.co.sharp.android.rb.intent.category.LAUNCHER」を記載する必要があります。具体的には、下記のように AndroidManifest.xml を変更してください。

【AndroidManifest.xml】

```
<activity
    android:name=".MainActivity"
    android:label="@string/app_name"
    android:theme="@style/AppTheme.NoActionBar.Fullscreen" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="jp.co.sharp.android.rb.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

ランチャーアイコンの表示は必須とします。

4.4 アプリのレイアウト

ここではアプリのレイアウトについて留意すべき内容について記載します。

第 1 世代ロボホンは背面の LCD と、プロジェクターの 2 つの表示デバイスを持っており、背面 LCD は縦専用、プロジェクターは横専用のレイアウトとなります。それぞれの解像度は背面 LCD が 720×960(※)、プロジェクターが 1280×720 となっており、density は「xhdpi」となります。

第 2 世代ロボホンは背面の LCD のみです。解像度は 240×320 です。density は「ldpi」となります。第 1 世代のレイアウトと一致するように StatusBar および NavigationBar のサイズが調整されております。レイアウト詳細については以降の章を参照の上、第 1 世代と共通のレイアウトとして調整ください。

※実際の背面 LCD デバイスの解像度は 240×320 のため、レイアウトされたものが 1/3 に縮小されて画面に表示されます。

※通常、レイアウトの指定については dpi(dp)値で指定しますが、1dp=2px 換算で表示されます。

※グラフィックデータについては、「drawable-xhdpi」や「mipmap-xhdpi」フォルダを利用ください。

以降、レイアウトに関する個別の内容について記載します。

4.4.1 全体レイアウト

全体のレイアウトについて以下に記載します。

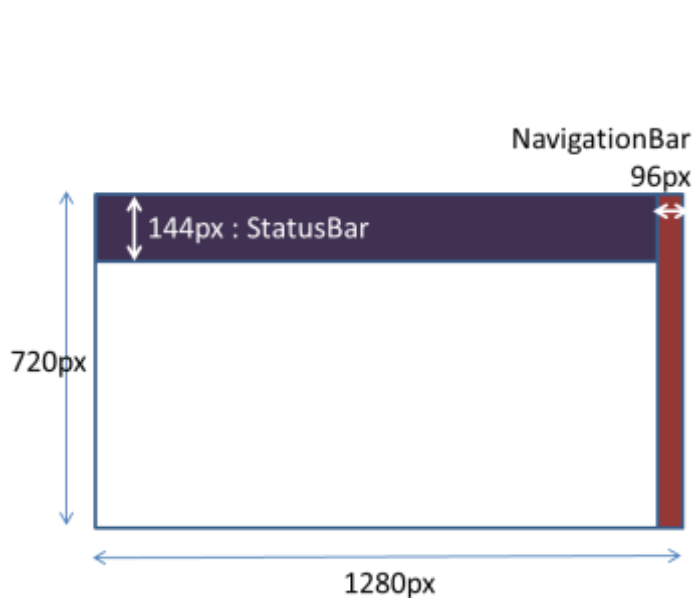


図 4-4 プロジェクター(横画面)

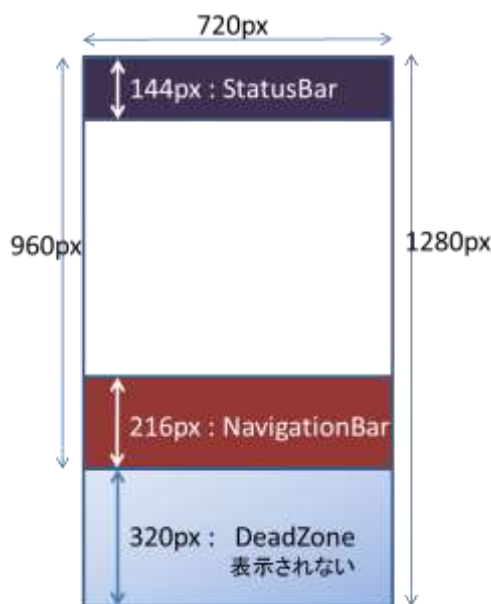


図 4-5 背面 LCD(縦画面)

ステータスバーは非表示の場合、上部タッチで一時表示されます。

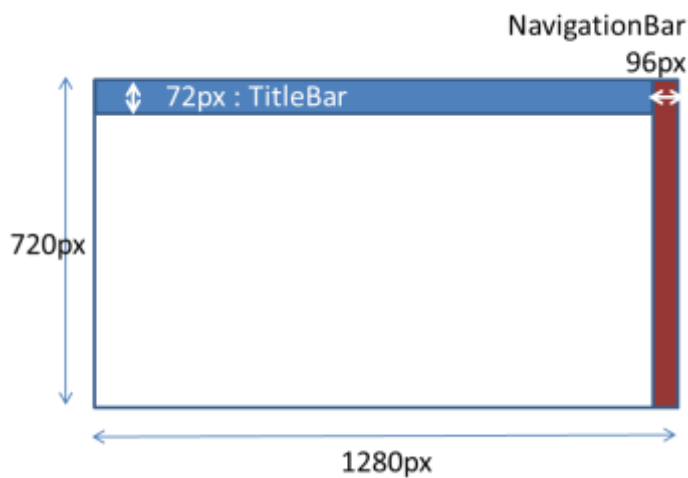


図 4-6 プロジェクター(横画面)

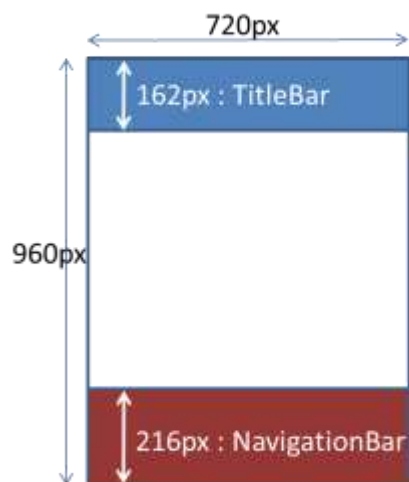


図 4-7 背面 LCD(縦画面)

ステータスバーは表示しないようにしてください。

タイトルバーを表示する場合は「[4.4.3 タイトルバー](#)」を参照してください。

4.4.2 Theme 設定

Theme は、「Theme.DeviceDefault」を指定しアクションバー非表示、ステータスバー非表示にしてください。

次の Theme を指定することで、両者非表示となります。

- Dark テーマ(推奨) : Theme.DeviceDefault.NoActionBar.Fullscreen
- Light テーマ : Theme.DeviceDefault.Light.NoActionBar.Fullscreen

4.4.3 タイトルバー

標準のアクションバーを使用せず、Toolbar をアクションバーとして利用してください。

アプリは下記の対応が必要です。

- Toolbar をアプリレイアウト上に配置し、高さを「81dp」指定してください。
- アクションボタンを表示する場合は、アクションボタンを ImageButton としてレイアウトしてください。オプションメニューのアクションアイテムとすると、意図通りのレイアウトに配置されません。
- アクションボタンのグラフィックは、72 x 72(px)のサイズのものを drawable-xhdpi に置いてください。
- アクションボタンは、Activity の onCreate 内で setSupportActionBar を実行して、Toolbar をアクションバーとして指定してください。
- タイトルには、アプリ名が表示されます。変更する場合は、getActionBar().setTitle()で変更してください。
- アプリ領域に重ねて、タイトルバーを透過することも可能です。



図 4-8 タイトルバー表示例

Toolbar の使い方についてのサンプルソースを以下に記載します。サンプルソースは SDK 内に同梱されています。

【SampleProjector: MainActivity.java】

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    setupTitleBar();  
}  
private void setupTitleBar() {  
    Toolbar toolbar = (Toolbar)findViewById(R.id.toolbar);    ... ①  
    setActionBar(toolbar);  
    btn = (ImageButton)findViewById(R.id.action_projector);    ... ②  
    btn.setOnClickListener(this);  
}
```

- ① ツールバーをアクションバーとして設定します。
- ② アクションボタンのリスナー設定を行います

【SampleProjector: activity_main.xml】

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
~中略~  
  
    <include layout="@layout/toolbar_layout"/>    ... ①  
  
~中略~  
</RelativeLayout>
```

- ① ツールバーをアプリレイアウト上に配置します。

【SampleProjector: toolbar_layout.xml】

```
<?xml version="1.0" encoding="utf-8"?>
<Toolbar
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="81dp"    . . . ①
    android:background="#80000000"
>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:orientation="horizontal"
        android:layout_gravity="right"
    >
        <ImageButton    . . . ②
            ~中略~
        />

    </LinearLayout>
</Toolbar>
```

- ① 高さを 81dp に指定します。
- ② アクションボタンを ImageButton として配置します。

4.4.4 パーツサイズ

各種 Android 標準の UI パーツのサイズを以下に記載します。

表 4-2 UI パーツのサイズ一覧

	背面 LCD(縦)	プロジェクター(横)	Android 標準
タイトルバー			
高さ(アプリ指定)	81dp	36dp	縦 : 56dp/横 : 48dp
タイトル文字サイズ	27dp	14dp	縦 : 20dp/横 : 14dp
ボタン領域(アプリ指定)	63x63(dp)	28x28(dp)	-
アイコンサイズ(アプリ指定)	72x72(px)	32x32(px)	-
アラートダイアログ			
タイトル文字サイズ	27sp	標準同じ	20sp
メッセージ文字サイズ	27sp	標準同じ	16sp
ボタン文字サイズ	24sp	標準同じ	14sp
スイッチ			
幅	186px	標準同じ	94px
高さ	108px	標準同じ	54px
SeekBar			
高さ	108px	標準同じ	64px
EditText			
文字サイズ	27sp	標準同じ	18sp
チェックボックス			
幅	72px	標準同じ	64px
高さ	72px	標準同じ	64px
ラジオボタン			
幅	72px	標準同じ	64px
高さ	72px	標準同じ	64px
Date/TimePicker			
モード	spinner	spinner	calendar/clock

4.5 Notification

Android 標準の API(Notification)が利用できますが、背面 LCD では 1 つ分の通知アイコン表示領域しかありません。そのため複数の通知アイコンが表示されている場合は、指定された優先度に従い、最優先のものしか表示されません。緊急度の高い通知は、Android 標準の Priority(PRIORITY_MAX・PRIORITY_HIGH)を指定することを推奨します。通知時は背面 LCD が消灯中のみ LED(目)を水色に点滅させることが可能ですが、色や点灯時間、点滅の速さを指定することは出来ません。

ロボホンは音声での操作をメインとしておりますので、確実にユーザに対して告知する必要がある通知はアプリ起動時に音声発話を行うことを推奨します。音声発話に関しては、「[6. 音声 UI](#)」を参照してください。

【Notification API の使い方例】

■LED(目)の点滅の停止

```

NotificationManager nm = ( NotificationManager ) getSystemService( NOTIFICATION_SERVICE );
Notification notif = new Notification();
notif.flags = Notification.FLAG_SHOW_LIGHTS;
notif.priority = Notification.PRIORITY_HIGH;
notif.ledARGB = 0x00000000; /*Color.rgb( 0, 0, 0, 0 ); */
notif.ledOnMS = 0;
notif.ledOffMS = 0;
nm.notify( 1, notif );

```

■LED(目)の水色点滅（色指定、点灯時間および点滅間隔は指定できません）

```

NotificationManager nm = ( NotificationManager ) getSystemService( NOTIFICATION_SERVICE );
Notification notif = new Notification();
notif.flags = Notification.FLAG_SHOW_LIGHTS;
notif.priority = Notification.PRIORITY_HIGH;
notif.ledARGB = 0xFFFFFFFF; /*Color.rgb( 255, 255, 255, 255 ); */
notif.ledOnMS = 1;
notif.ledOffMS = 0;
nm.notify( 1, notif );

```

■通知表示

```

Notification.Builder builder = new Notification.Builder(getApplicationContext());
/* 表示されるタイトルとテキストとアイコン */
builder.setTitle("TITLE");
builder.setText("text");
builder.setSmallIcon(android.R.drawable.ic_dialog_info);
/* Priority (必要に応じて) */
builder.setPriority(Notification.PRIORITY_HIGH);
/* 通知するタイミング */
builder.setWhen(System.currentTimeMillis());
/* 通知時のサウンド */
builder.setDefaults(Notification.DEFAULT_SOUND);
/* タップするとキャンセル */
builder.setAutoCancel(true);
/* NotificationManager を取得 */
NotificationManager manager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
/* Notification を作成し、通知 */
manager.notify(0, builder.build());

```

4.6 音量

ロボホンの音量は、設定アプリが保持する音量レベルに応じた各 STREAM 音量を指定します。
音量レベルに応じた各 STREAM 音量は表 3-3 の通りです。

表 4-3 音量レベルに応じた STREAM 音量

Level	RING※1	DTMF	MUSIC	ALARM※2	TTS
1	1	2	2	2	2
2	2	4	4	3	4
3	3	6	6	4	6
4	4	8	8	5	8
5	5	10	10	6	10
6	6	12	12	7	12
7	7	15	15	7	15

※1：SYSTEM、NOTIFICATION は RING 音量と同値となります

※2：ALARM 音量はユーザが気付きやすくするため音量を 1 段階上げています

この通り、一律音量変更しておりますので、アプリが個別に STREAM 音量を変更することは極力しないでください。
変更する場合は、アプリ側で元の設定値に戻す処理を入れるなど状態の齟齬が発生しないようケアしてください。

4.7 マナーモード

ロボホンのマナーモードは「[3.3 マナースイッチ](#)」に記載の通り、ハードウェアスイッチにて切り替える仕様として
います。そのため、アプリが `AudioManager#setStreamVolume()` 等の API を用いて音量 0 (マナー状態) にする
ことは禁止します。

ロボホンのマナーモードには第 1 世代ロボホンのみ以下の 2 種類が存在します。どちらのモードもモーションはで
きません。第 2 世代ロボホンでは「すべて OFF」の振る舞いとなります。

- ・ 会話のみ OFF：電話着信音やアラーム音の電子音のみを鳴らす
- ・ すべて OFF：電子音含めすべての音をならさない

いずれも Android としての状態は通常のマナーモードに変わりなく、`AudioManager#getRingerMode()` で取得で
きる値 (`RINGER_MODE_SILENT`) は変わりません。異なるのは `STREAM_MUSIC` を MUTE しているか否かとなり
ます。

マナーモード中 (イヤホン/BT ヘッドセット接続時は除く) はシナリオの実行ができません。

4.8 文字入力(第 1 世代のみ)

ロボホンでは文字入力アプリとして手書き入力 IME と Android キーボード（英語）を搭載しています。

Android キーボードはパスワード等の半角英数字の入力に使用することを想定しており、手書き入力 IME は人名等の全角文字の入力、その他に利用することを想定しています。対象の入力フィールドの InputType により起動する文字入力アプリを一部制御しています。

第 2 世代は標準の文字入力キーボードが起動します。

表 4-4 起動する IME 起動

入力対象種別	InputType	起動する IME	入力可能文字
よみがな／人名	TYPE_CLASS_TEXT TYPE_TEXT_VARIATION_PHONETIC	手書き IME	(全角)[ひらがな]
	TYPE_CLASS_TEXT TYPE_TEXT_VARIATION_PERSON_NAME	手書き IME	(全角)[ひらがな]
メールアドレス	TYPE_CLASS_TEXT TYPE_TEXT_VARIATION_EMAIL_ADDRESS	Android キーボード	(半角)[英数字][記号]
URI	TYPE_CLASS_TEXT TYPE_TEXT_VARIATION_URI	Android キーボード	(半角)[英数字][記号]
パスワード系	TYPE_CLASS_TEXT TYPE_TEXT_VARIATION_PASSWORD	Android キーボード	(半角)[英数字][記号]
	TYPE_CLASS_TEXT TYPE_TEXT_VARIATION_WEB_PASSWORD	Android キーボード	(半角)[英数字][記号]
	TYPE_CLASS_TEXT TYPE_TEXT_VARIATION_VISIBLE_PASSWORD	Android キーボード	(半角)[英数字][記号]
数字入力	TYPE_CLASS_PHONE または TYPE_CLASS_NUMBER または TYPE_CLASS_DATETIME	Android キーボード	(半角)[英数字]
その他（メッセージ入力、検索キーワード等）	上記以外	手書き IME	(全角)[ひらがな][カタカナ][JIS 漢字][英字][数字][記号]

Android キーボードは全画面モードで起動し、手書き入力 IME はインラインモードで起動します。

4.9 ライフサイクル

ロボホンアプリのライフサイクルは Android 標準と変わりありませんが、音声での操作をメインとするためにアプリはいくつかの実装が必要、もしくは制限があります。以下の記述にしたがってアプリの実装をしてください。

4.9.1 Activity

アプリはアプリ起動時に必ず画面を表示してください。また、画面(Activity)を表示している間は「[6.1 シーンの登録・解除](#)」にしたがって任意のシーンを登録してください。HOME 画面上では複数のアプリの音声コマンドを待ち受けており、音声コマンドのアプリ間の競合を避けるため、また、HOME 画面上でのパフォーマンス低下を招かないようにするため等の理由から、HOME 画面上での複雑なシナリオの実行をしてはいけません。複雑なシナリオはアプリ起動後に実行してください。

また、ロボホンは Android スマホにある Recent キーが存在しません。ユーザ操作により複数のアプリを切替えたり、HOME 画面から再開することができないため、Activity が非表示になった際には可能な限り速やかに終了してください。

アプリが単一の Activity しかもたない、かつ、電話帳などの他アプリ(Activity)連携を必要としない場合は、図 4-9 のように onPause() で finish() を実行し、自 Activity を終了してください。

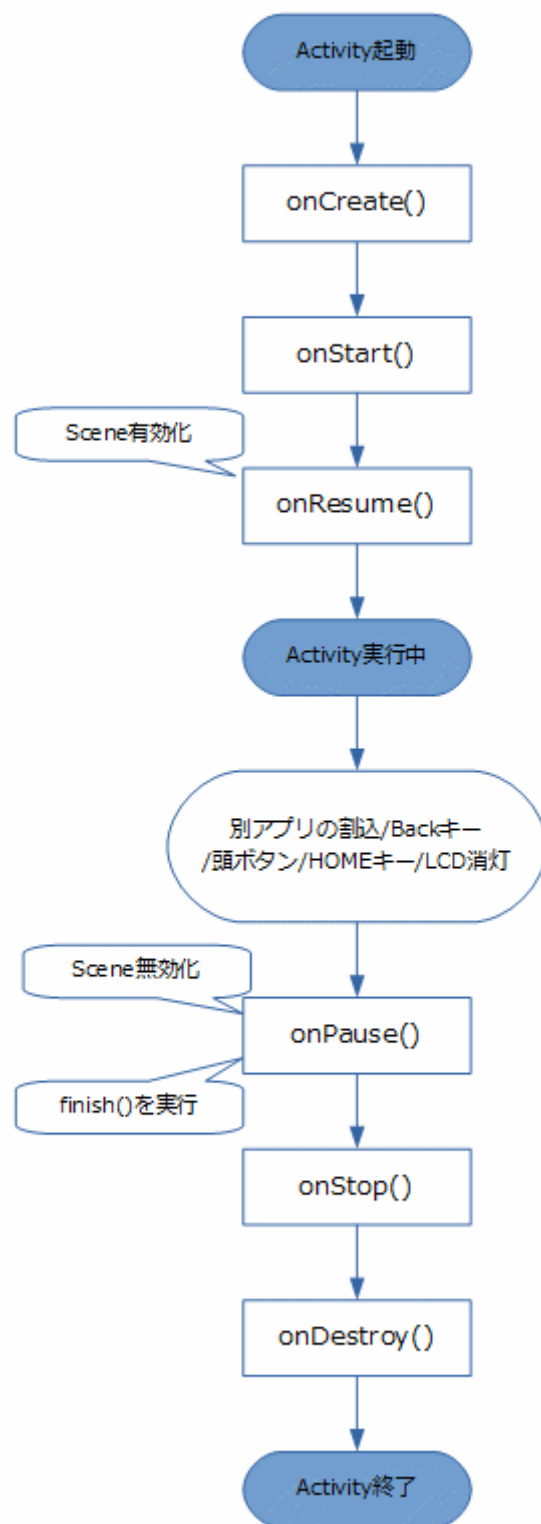


図 4-9 単一 Activity のアプリの場合

アプリが複数の Activity をもつ、もしくは、電話帳などの他アプリ連携を必要とする場合は、図 4-10 のように HOME キーイベント(ACTION_CLOSE_SYSTEM_DIALOGS)を受ける Broadcast レシーバークラスを実装し、同クラスの onReceive()内で finish()を実行し、自 Activity を終了してください。Back キーはアプリが特別な実装をしない限り Android 標準同様に onDestroy()まで実行されるため、特に考慮する必要はありません。

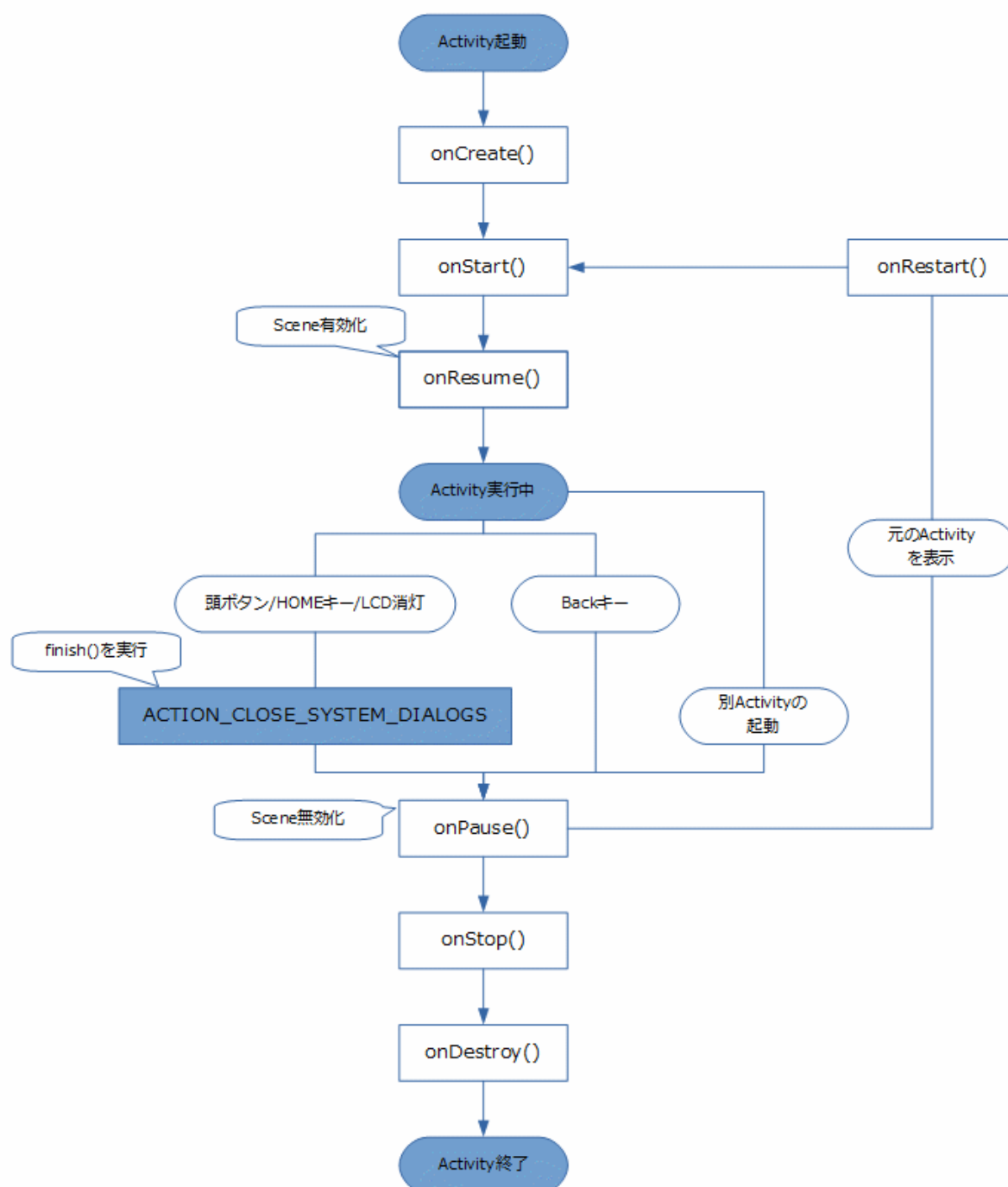


図 4-10 複数 Activity のアプリの場合

4.9.2 Service

Activity 無しでバックグラウンド動作する Service を実装する場合は、ユーザビリティや消費電流を考慮し、以下の条件を守った上で実装してください。

- ・ サービスの常駐を ON/OFF できる UI をアプリに設けること。
- ・ サービス常駐中はアプリ名をつけた Notification を表示し、サービス実行中である旨を表示すること。
- ・ サービス常駐中は、30分未満の短い周期で処理を行うような実装はしないこと。
- ・ HOME での他アプリの機能を阻害するような実装をしないこと。

4.10 アプリの削除

追加アプリは HOME 画面上でアプリアイコンを長押しすることでアンインストールできます。ただし、下記のパーミッションを付与したアプリは、HOME 画面上でのアンインストールを無効化できます。

その場合、アプリをアンインストールする際は、「設定 — その他 — アプリ」からアンインストールしてください。

アンインストールの無効化はロボホンのビルド番号 01.07.00 以降で利用できます。利用時は「設定 - 端末情報 - ビルド番号」よりビルド番号を確認してください。

【AndroidManifest.xml】

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="jp.co.sharp.sample.simple" >

    <uses-permission android:name="jp.co.sharp.android.rb.home.permission.GUARD_UNINSTALL" />

    <application
~以下略~
```

4.11 多言語対応

ロボホンのビルド番号 01.08.00 以降で音声認識および音声発話を含む言語の切替えが利用できます。現時点で日本語・英語・中文(簡体)・中文(繁体)・韓国語の 5 言語対応。ただし、中文(簡体)および中文(繁体)の音声認識と音声発話はいずれも普通話となります。

言語の切替えは以下の 2 つの方法があります。利用シーンに応じて適切な方法を選択してください。

- ・ 本体の言語設定を変更する。
- ・ 音声 UI の API で言語の設定を変更する。

前者の場合、通常の Android スマホ同様に Locale 設定が変更され、全てシーンで言語が切替ります。

※現状、一部のプリインアプリ(カメラ、ダンス等)のみ多言語対応しています。

言語の切替えは、「設定 — その他 — 言語と入力 — 言語(Language)を選択」より切替えることができます。言語を切替えると、その時点で登録されているシナリオが一旦全て解除され本体の再起動が行われます。再起動後、従来同様シナリオ登録要求の Broadcast Intent が Locale 情報と共に発行されますので、アプリはその Broadcast Intent を受けて適切な言語用のシナリオを登録してください。詳細は「[5.8 ホーム用シナリオの作り方](#)」および、「[6.1 シナリオの登録・解除](#)」および、SDK 内の SampleMultilingual を参照ください。

後者の場合、利用するアプリが起動している間のみ API をコールすることで言語を切り替えることを想定しています。アプリは終了時にデフォルトの言語設定に戻す対応を必ず入れてください。

HOME 画面で頭ボタンを押下すると、本体の言語設定に設定値が戻りますのでご注意ください。詳細は、「[6.6 音声認識・音声発話の言語切替](#)」および、SDK 内の SampleMultilingual を参照ください。

第 2 世代ロボホンでは Android OS のバージョンアップにより[言語とロケール](#)の仕組みが変更されております。多言語対応をするアプリは以下の点に留意ください。

- ・ 複数言語のサポート：res/values-ja/strings.xml を用意すること
- ・ 中国語(簡体字、繁体字)設定で取得できる default の Locale 値が変更されている

複数言語のサポートにより第 2 世代ロボホンの言語設定では図 4-11 のように複数の言語を設定することができます。

※出荷状態では日本語のみです。「言語を追加」から追加する言語を選択できます。



図 4-11 言語の設定

図 4-11 の状態の時、選択されるリソースの優先順位が `values-ja` → `values-en` → `values-zh` → `values` となります。アプリが保持している言語リソースが `values/strings.xml` と `values-en/strings.xml` の場合、`values-en` フォルダのリソースが採用され、日本語設定にもかかわらず英語表記となってしまいます。多言語対応されるアプリは必ず、`values-ja` フォルダも準備してください。

言語設定は、図 4-12 のように言語の優先順位をスライドして変更することで反映(ロボホン再起動)されます。

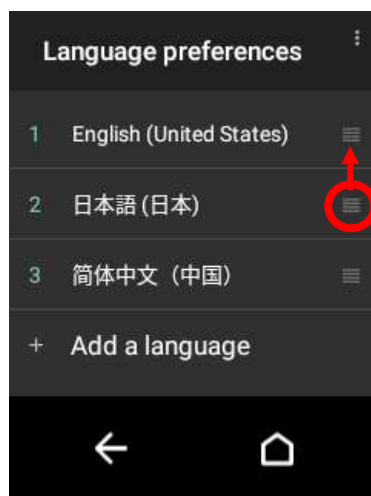


図 4-12 言語の設定

言語設定を中国語(簡体字、繁体字)に設定している場合、Locale.getDefault()で取得できる Locale の値が下記の通り第 1 世代と第 2 世代で異なります。

表 4-6 Locale.getDefault()で取得できる値

	メソッド別	簡体字	繁体字
第 1 世代	toString()	zh_CN	zh_TW
	getCountry()	CN	TW
第 2 世代	toString()	zh_CN_#Hans	zh_TW_#Hant
	getCountry()	CN	TW

getDefault()で取得する値を利用される場合は留意ください。シナリオの登録については音声 UI の方で差分を吸収しています。詳細は「[6.1 シナリオの登録・解除](#)」を参照ください。

4.12 Android OS のバージョンアップについて

第2世代ロボホンでは Android OS バージョンが Lollipop(5.0.2)から Oreo (8.1)にアップデートされています。

Android OS のアップデートに伴い、いくつかの注意が必要です。

変更に関する詳細は Android の Developer サイトをご確認ください。

<https://developer.android.com/about/versions/marshmallow/android-6.0-changes>

<https://developer.android.com/about/versions/nougat/android-7.0-changes>

<https://developer.android.com/about/versions/oreo/android-8.0-changes>

特に注意が必要な点としては

- ・ [バックグラウンド実行制限](#)
- ・ [ファイルシステムのパーミッションの変更](#)
- ・ [言語とロケール](#)

などがあります。

アプリは原則、第1世代、第2世代共通で利用できることが望ましいので、アプリの targetSdkVersion を 21 にすることを推奨します。それぞれの詳細については関係する各章にて個別に記載します。

5. シナリオ関連

ロボホンのアプリを開発するには、シナリオ(HVML)と Android アプリ(Java)の実装が必要です。本章ではシナリオ(HVML)で留意すべき内容について記載します。HVML の仕様については[参考資料 \[3\]](#)を参照してください。

5.1 HVML ファイル命名規則

HVML ファイルのファイル名は、Android パッケージ名のピリオド「.」をアンダースコア「_」に変換したものを接頭辞にする必要があります。例えば、パッケージ名が「jp.co.sharp.sample.simple」の場合、ファイル名は「jp_co_sharp_sample_simple_talk.hvml」のようになります。

また、ホームで受けるシナリオ(※)は、“_home”を付与し「jp_co_sharp_sample_simple_home.hvml」のようにしてください。詳細は「[5.8 ホーム用シナリオの作り方](#)」を参照ください。

5.2 プロデューサー名

HVML ファイルのプロデューサー名は、HVML を所有しているアプリの Android パッケージ名になります。例えば、パッケージ名が「jp.co.sharp.sample.simple」の場合、<producer>タグの内容は“jp.co.sharp.sample.simple”となります。

5.3 HVML バージョン管理

ロボホンは HVML2.0 の仕様に準拠しています。そのため、<hvm1>タグの version 属性値は“2.0”を指定してください。<version>タグは、シナリオの内容に変更があった場合に value 属性値をインクリメントしてください。

【HVML ファイル】

```
<hvm1 version="2.0"> ・・・①
  <head>
    <version value="1.0"/> ・・・②
```

① “2.0”を指定します。

② 正の整数または小数を指定します。(記載例) ○：“1.0” “2.01” ×：“1.0.1”

5.4 シーン名称

シーン名称は、HVML を所有しているアプリの Android パッケージ名としてください。アプリの中でシーンを分ける必要がある場合は「jp.co.sharp.sample.simple.activity1」のように「パッケージ名.固有文字列」としてください。但し、ホーム画面でのアプリ起動用のシナリオに関しては、<scene>タグの value 属性値を“home”としてください。

5.5 accost 名称

accost 名称は、HVML を所有しているアプリの Android パッケージ名を接頭辞にしてください。例えば、パッケージ名が「jp.co.sharp.sample.simple」の場合、Accost イベント名は「jp.co.sharp.sample.simple.test_accost」のようにし、HVML 中の<accost>タグの word 属性値として記載してください。

【HVML ファイル】

```
<accost priority="75" topic_id="t1" word="jp.co.sharp.sample.simple.test_accost"/>
```

5.6 記憶(memory_p)の利用

端末再起動後も変数値を保持したい場合、memory_p 変数が使用可能です。

他のアプリが用いている変数と名前の重複を避けるため、命名規則として、変数名はパッケージ名を接頭辞にしてください。例えば、パッケージ「jp.co.sharp.sample.simple」のアプリのシナリオ内で、memory_p 変数「loop_counter」を利用したい場合、変数名は「jp.co.sharp.sample.simple.loop_counter」としてください。memory_p 変数の登録・更新は、アプリから VoiceUIManager#updateAppInfo()を実行するか、もしくは、HVML から<memory>タグを記述することで行ってください。

【HVML での memory_p 変数の登録・更新】

```
<memory type="permanent" key="jp.co.sharp.sample.simple.loop_counter" value="1"/>
```

memory_p 変数で保存する値(value 値)は 1KB 以内としてください。1 アプリで利用する memory_p 変数は最大 100 個までとします。

また、memory_p 変数で保存した値を利用するには、HVML に以下のように記述してください。

【HVML での memory_p 変数の参照】

```
${memory_p:jp.co.sharp.sample.simple.loop_counter}
```

memory_p 変数の削除は、アプリから VoiceUIManager#removeVariable()を実行するか、もしくは、HVML から<memory>タグの operation="delete"指定により行ってください。

【HVML での memory_p 変数の削除】

```
<memory type="permanent" key="jp.co.sharp.sample.simple.loop_counter" operation="delete"/>
```

アプリをアンインストールした際には、アプリから登録された memory_p の値は削除されます。

5.7 モーションについて

モーションとは、ロボホンの動作、身振り(目の LED を含む)のことを表します。全てのモーションは基本姿勢から始まり、開始した基本姿勢で終わります。基本姿勢以外からモーションを再生するときは、はじめにいずれかの基本姿勢に遷移してからモーションが再生されます。

ただし以下の場合、モーションは再生されません。

- ・ 基本姿勢に遷移できない場合(詳細は表 5-1 参照)
- ・ マナーモード中
- ・ プロジェクター使用中
- ・ USB ケーブル接続中
- ・ イヤホン接続中
- ・ 電池残量が少ない場合

ここでは姿勢および、モーションの実行方法について記載します。

5.7.1 基本姿勢

ロボホンが静止できる姿勢を基本姿勢と呼び、頭ボタンを押下すると現状の姿勢から一番近い基本姿勢に遷移します。表 5-1 に基本姿勢の種類を示します。

表 5-1 基本姿勢の種類

種類	説明
STAND	立ちの基本姿勢
SIT	座りの基本姿勢
HAND__MOBILE	持ち運びの基本姿勢
CRADLE	充電台(充電中のみ)の基本姿勢

5.7.2 HVML で判別できる姿勢

基本姿勢に加え、HVML では変数の resolver:pose で以下の姿勢を利用できます。

表 5-2 HVML 上で判別できる姿勢の種類

resolver:pose	説明	モーション可能姿勢
stand	STAND で置いてある	○
hand_stand	STAND で持ち上げられている	○
sit	SIT で置いてある	○
hand_sit	SIT で持ち上げられている	○
mobile	HAND_MOBILE で置いてある	×
hand_mobile	HAND_MOBILE で持ち上げられている	○
back	仰向けで置かれている	×
belly	うつ伏せで置かれている	×
hand_phone	通話姿勢で持ち上げられている	×
projector	プロジェクター照射姿勢	×
cradle	CRADLE	○
immobile	モーション不可な姿勢	×

5.7.3 モーションの実行

ロボホンではシナリオファイルに<behavior>タグにて ID 指定することで各モーションを実行することが可能です。
各モーションの説明および ID は表 5-2 の通りです。

表 5-3 モーションの種類

種類	説明	ID
専用作り込みモーション	固定の発話内容のためのモーション 指定したIDのモーションを実行します。	固定の発話専用のため 利用しない
自動付与モーション A	発話文字数が30文字(読み)未満の場合のモーション 特定のワードに対して該当するモーションを実行します。	assign
自動付与モーション B	汎用的なモーションを繰り返します	general
ショートモーション	自動付与モーションAで使用している特定のワード に対する短いモーション	表5-4 参照

HVML での各モーションの記述例を以下に記載します。

【自動付与モーション A】

```
<behavior type="normal" id="assign"/>
```

【自動付与モーション B】

```
<behavior type="normal" id="general"/>
```

【ショートモーション】

```
<behavior type="normal" id="0x060000"/>
```


5.7.4 ショートモーションリスト

自動付与モーション A で使用している特定のワードに対応するモーションのリストとそのモーションについて説明します。

ショートモーションは、ロボホンのビルド番号 01.05.00 以降で利用できます。モーション ID を HVMML の<behavior> タグに記載しますとロボホンがショートモーションをします。<speech>タグに記載する発話内容は発話例を参考に記載してください。

表 5-4 ショートモーションの一覧

モーション ID	モーションの説明	発話例	ビルド番号
060000	腕を軽く前に出す	好き！、楽しい！	
060001	バンザイ	やった！、バンザイ！	
060002	両腕体横で左右に振る	びっくりー！、すごーい！	
060003	両手上下で抗議	むかつく	
060004	うなだれ	悲しい、寂しい	
060005	両手もじもじ	えへへ	
060006	腰に手	えっへん	
060007	お辞儀	ありがとう	
060008	浅いお辞儀	お疲れ	
060009	右手をあげてふる	バイバイ	
06000a	頷いて右手上げる	うん、～だよ、～しよう	
06000b	首を振る	違うよ、ううん	
06000c	首をかしげる	分からない、～かな？	
06000d	両腕を体の横で前後に振る	帰る、行く、歩く、走る	
06000e	両手でハンドルをにぎにぎ	車、運転	
06000f	両腕を体の横でぶらぶら	ひま	
060010	頭を左右、両腕を前後に動かす	忙しい	
060011	書類や本を読む	会社、学校、勉強、読書	
060012	右手を口元に持っていく	食べる、ご飯に行く、ランチする	
060013	上を向いてジョッキを飲み干す	飲む、飲み会、合コン	
060015	片腕を枕のようにして首を傾ける	寝る	
060016	お腹のあたりをさする	お腹すいた、空腹	
060017	片手で、顔を仰ぐ	暑い	
060018	両手を体の横で震わせる	寒い、凍える	
060019	あくびをする	眠い	
06001a	電話を掛ける	電話、携帯電話	
06001b	おでこを指す	カメラ	

モーション ID	モーションの説明	発話例	ビルド番号
06001c	おでこを指す	プロジェクター	
06001d	自分を指す	ぼく、ロボホン	
06001e	相手を指す	君	
06001f	両手を体の前で振る	音楽、ダンス	
060021	1 回頷く	うん	
060022	2 回頷く	うんうん	
060023	両手を広げる	おめでとう！	
060025	2 回お辞儀	あけましておめでとうございます	
060026	両手を広げた後、右手を前	トリックオアトリート！	
060027	両手を広げる	メリークリスマス！	
060028	ハグする	大好き	
060029	両手を広げて頷く	なるほど、そうなんだ	
06002a	ぐったり	ツライ、しんどい、二日酔い	
06002b	両手を目にあてる	淋しい、メソメソ	
06002c	右手を上げる	おはよう！	
06002d	両手を広げる（怒）	大嫌い！、最低！	
06002e	右手で頭をかく	てへぺろ、あちゃー	
060030	両手を前で広げる	大きい、広い	
060031	両手を前で狭める	小さい、狭い	
060032	両手を体の横で広げる	長い	
060033	右手を上上げる	高い	
060034	足元に片手を持ってくる	低い	
060035	ドリブルの仕草	サッカー	
060036	手でボールをつく	バスケット	
06003b	片手で投げキス	チュッ	
06003d	両手を体の横で、下を向いて首を振る	やれやれ	
06003e	肩をたたく感じ	まあまあ、落ち着いて	
06003f	俯いて悲しく手を振る	バイバイ	
060040	乾杯	かんぱい	
060041	そっぽをむく	ふんっ	
060043	慌てる	あたふた	
060044	背中をみる	背中、後ろ	
060046	両手でお腹周りをたたく	おなかいっぱい	
060047	きよろきよろする	どこにいる？、どこ	
060048	目を指す	目	
060049	鼻を指す	鼻	
06004a	腰を指す	腰	

モーション ID	モーションの説明	発話例	ビルド番号
06004d	横を向いて、手を広げて、足を曲げて、前後に体重移動している感じを表現	スノボ	
06004f	両手を胸の前でドラミング	ゴリラ	
060050	両手を胸の前に上げて、左右にゆらゆら	音楽	
060051	片手を上げて、片手で弓を弾く動き	バイオリン	
060052	両手を胸前で左右バラバラに上下に動かす	太鼓、ティンパニー	
060053	歯磨きする仕草	歯磨き	
060057	右腕を上げて、体をそる	シャワー	
060059	息が上がった感じ	疲れ、腰がいたい	
06005a	腰を曲げて後ろで手を組む	おんぶ	
06005b	片手を上にあげて頭をこつんと打つ感じ	まいった、しまった	
06005c	手を口に当てて少し斜め下を見る	うふふ	
06005d	右腕を前に出して上下に動かす	じゃんけん	
06005e	マイクをもって歌っているような動き	カラオケ、歌う	
06005f	両手胸の前に持ってきて右手を口元にもってくる	ラーメン、うどん、丼	
060062	両手を横に広げて前から後ろへ動かす	飛行機	
060064	ポインターをもって白板コツコツ	プレゼン	01.08.00
06007a	両腕頭で腰を落とす	スクワット	01.08.00
06007b	カンフーのポーズ	カンフー	01.08.00

5.8 ホーム用シナリオの作り方

ロボホンではホーム画面からアプリ起動を行うための特殊なシナリオ(ホーム用シナリオ)を作成する必要があるため、以下にその作成方法、ルールを記載します。サンプルは SDK 内に同梱されております。

【SampleSimple:jp_co_sharp_sample_simple_home.hvml】

```
<?xml version="1.0" ?>
<hvml version="2.0">
  <head>
    <producer>jp.co.sharp.sample.simple</producer>
    <description>サンプルアプリのホーム起動シナリオ</description>
    <scene value="home"/>
    <version value="1.0"/>
    <tool_version>1.00</tool_version>
    <!--(ロボホン名)、App_name(を)起動(して)-->
    <situation priority="78" topic_id="t1" trigger="user-word">${Local:WORD_APPLICATION} eq さんぶるあぶり</situation>
    <situation priority="78" topic_id="t1" trigger="user-word">${Local:WORD_APPLICATION_FREEWORD} eq さんぶるあぶりしよう</situation>
  </head>
  <body>
    <topic id="t1" listen="false">
      <action index="1">
        <speech>${resolver:speech_ok(${resolver:ok_id})}</speech>
        <behavior id="${resolver:motion_ok(${resolver:ok_id})}" type="normal"/>
      </action>
      <next href="#t2" type="default"/>
    </topic>
    <topic id="t2" listen="false">
      <action index="1">
        <speech>サンプルアプリを起動するね</speech>
        <behavior id="assign" type="normal"/>
        <control function="start_activity" target="home">
          <data key="package_name" value="jp.co.sharp.sample.simple"/>
          <data key="class_name" value="jp.co.sharp.sample.simple.MainActivity"/>
          <data key="mode" value="音声で起動したよ"/>
          <data key="key_test1" value="val_test1"/>
          <data key="key_test2" value="val_test2"/>
        </control>
      </action>
    </topic>
  </body>
</hvml>
```

①

②

③

④

⑤

- ① scene value="home"とします。
- ② 「WORD_APPLICATION」として、アプリ名を「ひらがな」で記載します。
situation の priority は 78~84 を指定します(特に理由が無ければ 78 とします)。priority は必ず指定してください。また、演算子は eq を使用してください。
- ③ 「WORD_APPLICATION_FREEWORD」として、「キーワード」+「動詞」のアプリ起動用の文言を「ひらがな」で記載します。
例：「たくしーよんで」「くいずしよう」「れしぴけんさくして」等
priority は必ず指定してください。また、演算子は eq を使用してください。
- ④ topic listen="false"とします。
- ⑤ control タグはアプリ起動用の記述とします。詳細は「[5.9 シナリオからのアプリ起動](#)」の項を参照してください。

上記のように②③の 2 つをアプリ起動用文言として登録可能です。この場合、「(ロボホン名)、サンプルアプリ(を)起動(して)」,「(ロボホン名)、サンプルアプリしよう」のユーザ発話でアプリを起動することができます。()内の発話は省略可能です。ホーム用シナリオの再生条件は②、③のみとしてください。他の<situation>タグや<accost>タグは記載しないでください。また、②、③はそれぞれ 1 個とし、複数記載しないでください。

アプリが起動しない場合、ロボホンにプリセットされたワードが動作してしまっている可能性があります。そのような場合は起動用の文言を変更してください。

新たに追加されたアプリの起動用文言はユーザが使い方を知る手段がないため、アプリ起動時にアプリの起動用文言をガイドすることを推奨します。

また、登録できるホーム用シナリオは 1 アプリ 1 シナリオのみです。複数登録しないでください。

言語を日本語以外に切り替えた場合、前述の

`${Local:WORD_APPLICATION}`および`${Local:WORD_APPLICATION_FREEWORD}`

の変数が利用できなくなります。

日本語以外の言語の場合は、`${Lvcsr:Basic}`を利用した起動コマンドを設定してください。その他のルールは日本語と同様です。

例) `<situation priority="78" topic_id="t1" trigger="user-word">${Lvcsr:Basic} include [start] and
${Lvcsr:Basic} include [application]</situation>`

5.9 シナリオからのアプリ起動

シナリオからアプリを起動するためには、ホームアプリに対して Activity の起動を要求する必要があります。また、アプリの起動だけでなく、シナリオからサービスの起動やブロードキャストの送信も要求できます。具体的には下記のように<control>、<data>タグを使います。

<control>タグは、属性として target と function を持ちます。target は「home」固定とし、function で指定する内容については表 5-5 を参照してください。

<data>タグの key 要素は、表 5-6 および表 5-7 を参照してください。

表 5-5 function 要素

種類	説明
"start_activity"	アプリを起動する。
"start_service"	サービスを起動する。
"send_broadcast"	ブロードキャストを送信する。

表 5-6 key 要素 (start_activity、start_service)

種類	説明	value
"package_name"	パッケージ名を設定する。必須。	起動するパッケージ名を指定する
"class_name"	クラス名を設定する。必須。	起動するクラス名を指定する
"mode"	任意。 シナリオから起動先アプリやサービスに情報を渡したい場合に利用する。 VoiceUIVariable情報を取得する必要がなく、簡易な実装で情報の受け渡しが可能。	シナリオから起動先に渡したい任意の情報を指定する ※"mode"をkeyとしてIntentに付与される
"任意の値" ※複数設定が可能	任意。 シナリオから起動先アプリやサービスに情報を渡したい場合に利用する。 複数設定可能だが、VoiceUIVariable情報を取得する必要がある。	シナリオから起動先に渡したい任意の情報を指定する ※"VoiceUIVariable"をKeyとして、VoiceUIVariableのリストとしてすべて渡される

表 5-7 key 要素 (send_broadcast)

種類	説明	value
"package_name"	パッケージ名を設定する。任意。	宛先パッケージ名を指定する
"class_name"	クラス名を設定する。任意。	宛先クラス名を指定する
"action"	起動Actionを設定する。必須。	起動Actionを指定する

"任意の値" ※複数設定が可能	任意。 シナリオから受信先に情報を渡したい場合に利用する。 複数設定可能だが、VoiceUIVariable 情報を取得する必要がある。	シナリオから受信先に渡したい任意の情報を指定する ※ "VoiceUIVariable" を Key として、VoiceUIVariable のリストとしてすべて渡される
--------------------	--	--

サンプルアプリを起動させるシナリオのサンプルは以下になります。サンプルは SDK 内に同梱されております。

【SampleSimple: jp_co_sharp_sample_simple_home.hvml】

<p>～中略～</p> <pre> <topic id="t2" listen="false"> <action index="1"> <speech>サンプルアプリを起動するね</speech> <behavior id="assign" type="normal"/> <control function="start_activity" target="home"> <data key="package_name" value="jp.co.sharp.sample.simple"/> <data key="class_name" value="jp.co.sharp.sample.simple.MainActivity"/> <data key="mode" value="音声で起動したよ"/> <data key="key_test1" value="val_test1"/> <data key="key_test2" value="val_test2"/> </control> </action> </topic> </pre> <p>～中略～</p>	<p>①</p> <p>②</p> <p>③</p>
--	----------------------------

- ① アプリを起動する設定にします。
- ② 起動するパッケージ名とクラス名を設定します。
- ③ 必要に応じてアプリ側に渡す情報を設定します。

5.10 near 演算子の使い方

「\${Lvcsr:Kana} near xx」のように、ユーザ発話のある程度のゆらぎを持たせて合致させたい場合に使用します。ただし最低でも 4 文字以上の文字列を比較する場合のみ利用してください。短い文字列の比較では「テンキ」と「デ
ンキ」 など、1 文字違うと全く違う意味になるうえ、近似度が低くなり事実上成立しない条件となります。

5.11 肯定/否定の聞き取り方

ユーザに対して特定の処理や機能の実行をしてもよいかどうかロボホンが尋ねるケースが多くあります。アプリ毎にロボホンの振る舞いが異なるとユーザは混乱するため、ある程度共通のシナリオ記述方法を定義します。アプリの仕様に依拠して発話等調整の上シナリオを作成してください。サンプルは SDK 内に同梱されております。

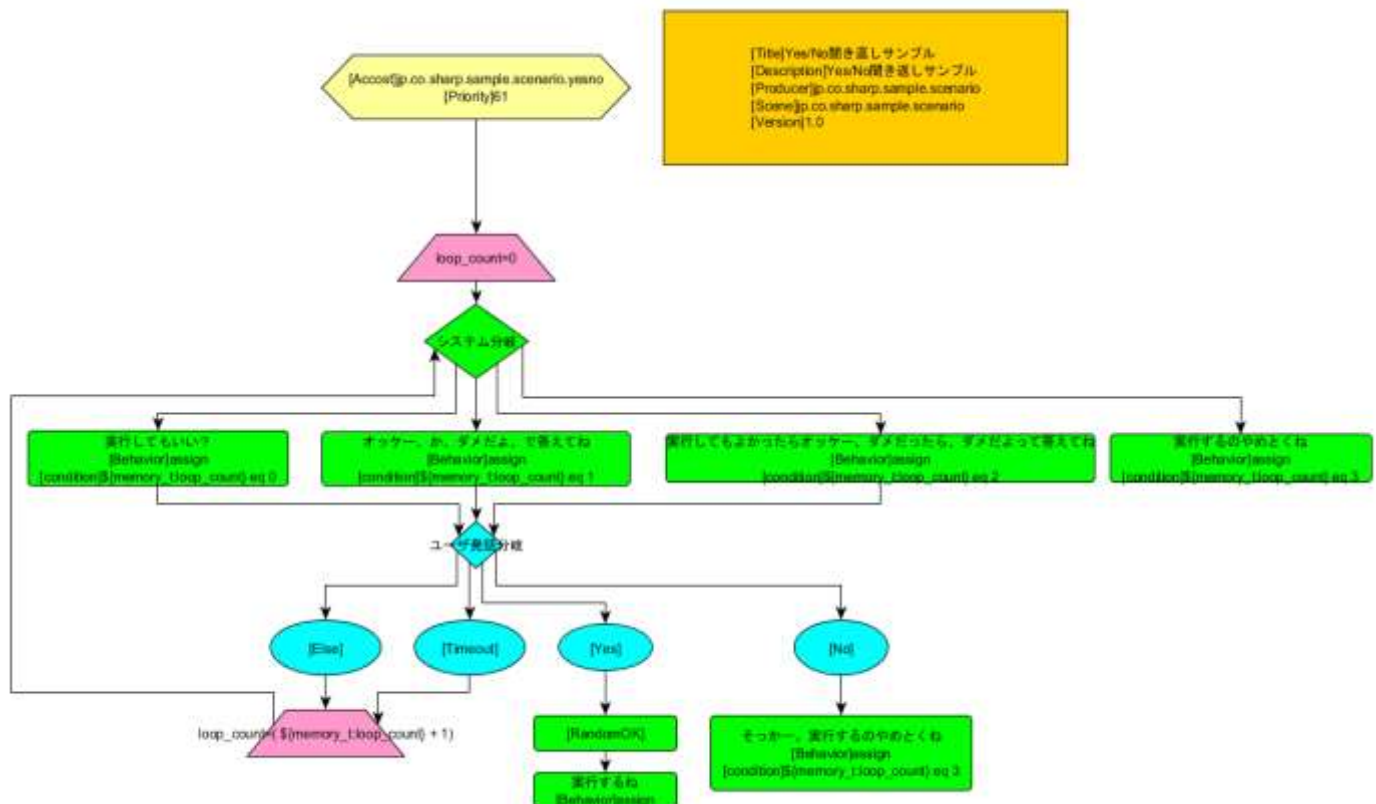


図 5-1 肯定/否定の聞き取りサンプルシナリオのフローチャート

【SampleScenario:jp_co_sharp_sample_scenario_yesno.hvml】

～中略～

```

<topic id="t2" listen="false">
  <rule>
    <condition case_id="c1">${memory_t:loop_count} eq 0</condition>    . . . ①
    <condition case_id="c2">${memory_t:loop_count} eq 1</condition>
    <condition case_id="c3">${memory_t:loop_count} eq 2</condition>
    <condition case_id="c4">${memory_t:loop_count} eq 3</condition>
  </rule>
  <case id="c1">    . . . ②
    <action index="1">
      <speech>実行してもいい？</speech>
      <behavior id="assign" type="normal"/>
    </action>
    <next href="#t3" type="default"/>
  </case>

```

～中略～

</topic>

```

<topic dict="Reply" id="t3" listen="true">
  <a href="#t4">
    <situation trigger="user-word">${Local_Reply:GLOBAL_REPLY_YES} neq null</situation>
  </a>
  <a href="#t6">
    <situation trigger="user-word">${Local_Reply:GLOBAL_REPLY_NO} neq null</situation>
  </a>
  <a href="#t7" type="default"/>
  <next href="#t7" type="default"/>
</topic>

```

```

<topic id="t4" listen="false">
  <action index="1">
    <speech>${resolver:speech_ok(${resolver:ok_id})}</speech>
    <behavior id="${resolver:motion_ok(${resolver:ok_id})}" type="normal"/>
  </action>
  <next href="#t5" type="default"/>
</topic>

```

```

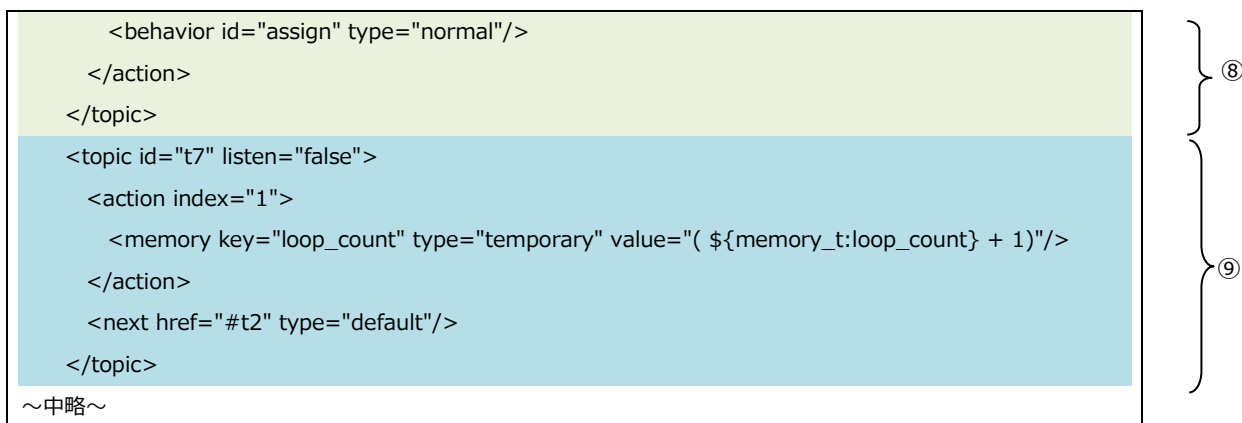
<topic id="t5" listen="false">
  <action index="1">
    <speech>実行するね</speech>
    <behavior id="assign" type="normal"/>
  </action>
</topic>

```

```

<topic id="t6" listen="false">
  <action index="1">
    <speech>そっかー。実行するのやめとくね</speech>

```



- ① 本サンプルでは肯定/否定の聞き返し回数を、変数`${memory_t:loop_count}`に記憶し、回数により発話内容を変えています。また、聞き返し回数 3 回目で処理を中断します。
- ② 初回の肯定/否定の聞き返しのため、発話後に、③topic id="t3"に遷移します。
- ③ topic(listen="true")タグに dict="Reply"属性を指定し、応答を待ち受けます。
- ④ オッケー、了解等、ユーザが肯定の意図の回答(※1)をした場合に選択され、⑦topic id="t4"に遷移します。
- ⑤ いいえ、だめだよ等、ユーザが否定の意図の回答(※2)をした場合に選択され、⑧topic id="t6"に遷移します。
- ⑥ ④⑤の回答に該当しない場合、または一定時間(10 秒)ユーザの応答が無い場合に選択され、⑨topic id="t7"に遷移します。
- ⑥ ユーザが肯定した場合に実行されます。
変数`${resolver:speech_ok}`等の詳細は、「[5.12 OK の返事をする際の記述方法](#)」を参照ください。
- ⑧ ユーザが否定した場合に実行されます。
- ⑨ ユーザが肯定/否定した場合以外に実行され、
聞き返し回数カウント用の変数`${memory_t:loop_count}`をインクリメントし、①topic id="t2"に戻ります。

(※1) 上記サンプルでは、肯定の意図で認識する言葉は以下です。

はい、いいよ、よろしく、オッケー、大丈夫、頼む、お願い、はい、よろしくね、そうだよ、そうだね、そうだよね、そうだった、そうだったね、そうだったよ、オッケーよ、オッケーです、大丈夫よ、大丈夫です、頼むね、頼むよ、頼みます、お願いね、お願いします

(※2) 上記サンプルでは、否定の意図で認識する言葉は以下です。

だめ、だめだよ、いいえ、違う、嫌だ、いない、やめとく、だめよ、だめね、だめです、だめだ、だめだね、だめだめ、違うよ、違うね、違います、ちげーよ、嫌だよ、嫌だね、やめとくよ、やめとくね、やめときます、いないよ、いないね、いないです、しない、しないよ、しないね、しないです

5.12 OK の返事をする際の記述方法

ユーザの発話などに対して、ロボホンが返事する、というケースにおける、シナリオ記述方法を定義します。以下のように記述することで、「オッケー /はい /うん！ /わかったー /了解！」というワードの中から、ランダムで一つが選択され発話します。サンプルは SDK 内に同梱されております。

【SampleSimple: jp_co_sharp_sample_simple_app_end.hvml】

```

~中略~
<topic id="t1" listen="false">
  <action index="1">
    <speech>${resolver:speech_ok(${resolver:ok_id})}</speech>    ... ①
    <behavior id="${resolver:motion_ok(${resolver:ok_id})}" type="normal"/>    ... ②
  </action>
  <next href="#t2" type="default"/>
</topic>
<topic id="t2" listen="false">
  <action index="1">
    <speech>サンプルアプリを終了するね</speech>    ... ③
    <behavior id="assign" type="normal"/>
    <control function="end_app" target="jp.co.sharp.sample.simple"/>
  </action>
</topic>
~中略~

```

- ① <speech> タグ内に、OKの返事をランダムで返却する変数を記載します。
- ② <behavior>タグ内に、OK の返事を行う際の、モーション ID を返却する変数を記載します。
- ③ 上記サンプルでは「オッケー /はい /うん！ /わかったー /了解！」からランダムで発話後、続いて「サンプルアプリを終了するね」と発話させています。

5.13 ユーザ発話の認識エラー処理の記述方法

ネットワークエラー等により、認識結果が取得できなかった場合は、ユーザ発話変数\${Lvcsr:Basic}、\${Lvcsr:Kana}に「VOICEPF__ERR__xx」という文字列が格納されます（全角、xx 部分は各エラー毎のエラー理由文字列です）。通常は意識する必要はありませんが、ユーザ発話の内容をそのまま発話や画面に出力するなど、エラー処理が必要な場合に記述してください。サンプルは SDK 内に同梱されております。

【SampleSimple: jp_co_sharp_sample_simple_talk.hvml】

```

~中略~
<topic id="t1" listen="false">
  <action index="1">
    <speech>目が黄色になったら喋ってね</speech>

```

```

    <behavior id="assign" type="normal"/>
  </action>
  <next href="#t2" type="default"/>
</topic>
<topic id="t2" listen="true">
  <a href="#t3">
    <situation trigger="user-word">V O I C E P F _ _ E R R   in ${Lvcsr:Basic}</situation>   . . . ①
  </a>
  <a href="#t4" type="default"/>
  <next href="#t3" type="default"/>
</topic>
<topic id="t3" listen="false">
  <action index="1">
    <speech>よくわからなかったよ。またお話ししてね。</speech>   . . . ②
    <behavior id="assign" type="normal"/>
  </action>
</topic>
<topic id="t4" listen="false">
  <action index="1">
    <speech>喋った内容を、背中の画面に表示したよ。確認してみてね</speech>   . . . ③
    <behavior id="assign" type="normal"/>
    <control function="recog_talk" target="jp.co.sharp.sample.simple">
      <data key="Lvcsr_Basic" value="${Lvcsr:Basic}"/>
    </control>
  </action>
</topic>
~中略~

```

- ① 変数`${Lvcsr:Basic}`に、エラー文字列が格納されていないかを判定します。
- ② エラー文字列が格納されていた場合は、聞き取れなかったという内容の発話をしています。
- ③ 変数`${Lvcsr:Basic}`にエラー文字列以外が格納されていた場合、発話文字列を画面に表示します。

5.14 HVML 作成時のデバッグ方法

HVML ファイルを作成する際に利用できるデバッグ方法について記述します。

5.14.1 ユーザ発話を擬似的に発生させる

RoBoHoN SDK を利用して開発するアプリでは、使用可能な音声認識のアクセス回数に上限があります。開発時に音声認識のアクセス回数を低減する手段として、ADB コマンドを用いて、ユーザ発話イベントを擬似的に発生させることが可能です。

日本語を用いるため、DOS プロンプトでは動作しない場合があるので、Cygwin などのターミナルを利用してください。また、本コマンドを有効にするためには、ロボホンの「開発者オプション」が表示可能な状態としておいて下さい。開発者オプションの表示方法は[参考資料 \[1\]](#)を参照ください。

ユーザ発話イベントの ADB コマンドフォーマットは以下の通りです。

【ユーザ発話の ADB コマンドフォーマット】

```
#adb shell
#am broadcast -a jp.co.sharp.android.voiceui.SIMULATE --es keyvals "ユーザ発話変数名|変数に格納される文字列"
```

以下の例では、trigger="user-word"、「\${Lvcsr:Basic} eq こんにちは」が記載されている<situation>タグが選択されます。ただし、再生されるシナリオは通常の発話と同様であり、scene や priority に従って選択されます。

【ユーザ発話の ADB コマンド例】

```
am broadcast -a jp.co.sharp.android.voiceui.SIMULATE --es keyvals "Lvcsr:Basic|こんにちは"
```

"Local_Reply:GLOBAL_REPLY_YES|いいよ"、"Local_Reply:GLOBAL_REPLY_NO|だめだよ"、
"Local:WORD_APPLICATION|さんぶるあぷり" などのように、\${Lvcsr:Basic}以外のユーザ発話コマンドも指定可能です。

また、複数の変数を同時に指定することも可能です。以下の例では、\${Lvcsr:Basic}と\${Lvcsr:Kana}を指定しています。

【\${Lvcsr:Basic}と\${Lvcsr:Kana}を同時に利用する ADB コマンド例】

```
am broadcast -a jp.co.sharp.android.voiceui.SIMULATE --es keyvals "Lvcsr:Basic|こんにちは;Lvcsr:Kana|コンニチハ"
```

5.14.2 音声認識アクセス回数確認方法

設定メニューの「その他」→「音声認識アクセス回数」で確認が可能です。

本表示を有効にするためには、ロボホンの「開発者オプション」が表示可能な状態としておいて下さい。

また、アクセス回数が上限に達した場合、音声認識によるシナリオ選択が行われなくなります。

アクセス回数制限により認識失敗した際には、logcat に以下が出力されます。

TAG: Speech_Recog

TEXT: The number of cloud speech recognition has exceeded the upper limit.

5.14.3 HVML ファイルの編集方法

HVML は XML をベースとした言語ですので、Android Studio の

Settings→Editor→FileTypes

で XML の拡張子に".hvmI"を追加することで、XML として編集、構文チェックが可能になります。

詳細は[参考資料 \[1\]](#)を参照ください。

6. 音声 UI

本章では、アプリにおける音声 UI の利用方法について記載します。音声 UI のインターフェース仕様の詳細は、[参考資料 \[2\]](#)を参照してください。

音声 UI が提供する機能は以下の通りです。

- ・ シナリオの登録・解除
- ・ シーンの登録・解除
- ・ アプリトリガによる発話
- ・ アプリトリガによる記憶の更新
- ・ 各種コールバック
- ・ 音声認識、音声発話それぞれの言語切替え

音声 UI を利用するためには AndroidManifest.xml に以下を追加する必要があります。

【AndroidManifest.xml】

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="jp.co.sharp.sample.simple" >

    <uses-permission android:name="jp.co.sharp.android.permission.VOICEUI" />

    <application
        ~中略~
        <uses-library
            android:name="jp.co.sharp.android.voiceui.framework"
            android:required="true" />
        ~中略~
    </application>

</manifest>
```

- ① <manifest> タグ内に <uses-permission android:name="jp.co.sharp.android.permission.VOICEUI" /> を追加します。
- ② <application> タグ内に <uses-library android:name="jp.co.sharp.android.voiceui.framework" android:required="true" /> を追加します。

6.1 シナリオの登録・解除

アプリは発話やモーションを実行するために音声 UI にシナリオを登録する必要があります。

音声 UI はロボホンの起動直後やアプリのインストール時などのタイミングで Broadcast Intent 「jp.co.sharp.android.voiceui.REQUEST_SCENARIO」を発行します。この Broadcast Intent を契機に各アプリでシナリオ登録を行ってください。登録したシナリオの解除は VoiceUIManager#unregisterScenario() を実行します。解除については任意のタイミングで実行できますが、アプリのアンインストール時に音声 UI がパッケージ名から関連する HVML の登録を解除するため、HVML ファイルの命名規則を守っていれば特別な対応はありません。

REQUEST_SCENARIO の Intent には Extra 情報として現在の言語設定(Locale)が文字列で格納されています※。

アプリはその Locale 情報をもとに適切な言語用のシナリオを登録してください。

※第 2 世代ロボホンでは音声 UI で第 1 世代ロボホンと共通の Locale 文字列となるよう差分を吸収しています。詳細は「[4.11 多言語対応](#)」を参照ください。

一連の実装は、サンプルアプリの voiceui/RequestScenarioReceiver.java および RegisterScenarioService.java で実装されておりますのでそのまま流用してください。以下、サンプルアプリを参考に利用手順と解説を記載します。サンプルは SDK 内に同梱されております。

【SampleMultilingual:AndroidManifest.xml】

```
<?xml version="1.0" encoding="utf-8"?>
  ~中略~
  <application
    ~中略~
    <service
      android:name=".voiceui.RegisterScenarioService"
      android:enabled="true"
      android:exported="false" >
    </service>
    <receiver android:name=".voiceui.RequestScenarioReceiver" >
      <intent-filter>
        <action android:name="jp.co.sharp.android.voiceui.REQUEST_SCENARIO" />
      </intent-filter>
    </receiver>
  </application>
</manifest>
```

- ① Broadcast Intent を受けるための Receiver とシナリオを登録するための Service を AndroidManifest に追加します。
- ② 登録するシナリオを assets 配下に格納します。assets 配下は /hvm1 (日本語)※1 /hvm1_en_US (英語) /hvm1_zh_CN (中文(簡体))※2 /hvm1_ko_KR(韓国語)の言語毎のサブフォルダと、さらにその配下にそれぞれ /home (ホーム用シナリオ) /other (アプリシナリオ) の構成とします。

※1 多言語対応しないアプリは /hvm1 フォルダのみ

※2 繁体字と共通のシナリオ

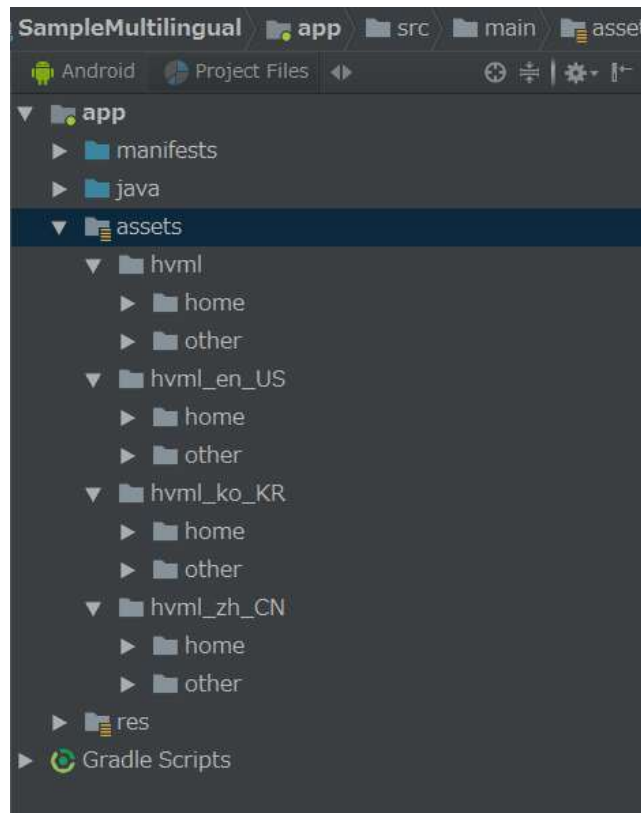


図 6-1 asstes フォルダ構成

シナリオの作成に関する詳細については、「[5. シナリオ関連](#)」および[参考資料](#)[3]を参照ください。

【SampleMultilingual:RequestScenarioReceiver.java】

```
private static final String KEY_LOCALE = "locale";
@Override
public void onReceive(Context context, Intent intent) {
    if (ACTION_REQUEST_SCENARIO.equals(intent.getAction())) {
        //シナリオ登録要求の場合
        Intent baseIntent = new Intent();
        baseIntent.putExtra(KEY_LOCALE, intent.getStringExtra(KEY_LOCALE));
        RegisterScenarioService.start(context, baseIntent, RegisterScenarioService.CMD_REQUEST_SCENARIO);
    }
}
```

- ③ REQUEST_SCENARIO を受けて onReceive() で Locale 情報を取得し、シナリオ登録用のサービスを起動します。

第2世代ロボホンでは Android OS のアップデートによりバックグラウンドサービスの実行がエラーとなることがあります。targetSdkVersion を 21 にするか、シナリオ登録用のサービスを[フォアグラウンドで実行する](#)かしてください。

詳細は「[4.12 Android OS のバージョンアップについて](#)」および[バックグラウンド実行制限](#)を参照ください。

【SampleMultilingual: RegisterScenarioService.java】

```
public int onStartCommand(Intent intent, int flags, int startId) {
    ～中略～

    case CMD_REQUEST_SCENARIO:
        //locale 情報を取得.
        String locale = intent.getStringExtra(KEY_LOCALE);

        //home シナリオ登録.
        registerScenario(locale, true);
        //home 以外のシナリオ登録.
        registerScenario(locale, false);

    ～中略～
}

private void registerScenario(String locale, Boolean home) {
    ～中略～
    //ローカルフォルダーの hvml ファイルのシナリオを登録する.
    for(File file: files){
        int result = VoiceUIManager.VOICEUI_ERROR;
        try {
            if (home == true) {
                //home 用.
                result = mVUIManager.registerHomeScenario(file.getAbsolutePath());
            } else {
                //other.
                result = mVUIManager.registerScenario(file.getAbsolutePath());
            }
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
    ～中略～
}
```

- ④ RegisterScenarioService#registerScenario()でホームシナリオ/アプリシナリオをそれぞれ登録します。
第1引数で Locale 情報、第2引数がホームシナリオ/アプリシナリオを指定しています。
引数に応じて assets から適切な HVML ファイルをアプリ毎のローカルフォルダ("/data/data/パッケージ名/files/")にコピーします。
- ⑤ VoiceUIManager#registerHomeScenario()を実行し、ホームシナリオを登録します。引数 path は HVML フ

ファイルのフルパスを指定して下さい。

- ⑥ VoiceUIManager#registerScenario()を実行し、ホーム以外のシナリオを登録します。引数 path は HVML ファイルのフルパスを指定して下さい。

シナリオの登録処理はロボホンの起動直後やアプリのインストール時に実行されますが、非同期で実行されるため、HVML がフォーマット不正等で登録に失敗した場合、アプリに通知されません。登録失敗時は logcat に以下のエラーログが出力されますので、エラーコードに従って HVML ファイルを修正してください。

TAG: VoiceUIService

TEXT: onReceiveRegisterHvml file:ファイルパス Restriction Result: エラーコード

エラーコードは以下を参照してください。

表 6-1 HVML 登録時エラーコード一覧

エラーコード	説明
FAILURE_PRIORITY	<situation>タグまたは<accost>タグの priority 属性に不正な値が記載されています。
FAILURE_PRODUCER	<producer>タグが記載されていない、または不正な値が記載されています。
FAILURE_SCENE	<scene>タグの value 属性に不正な値が記載されています。ホームシナリオが"home"以外のシーンを指定している場合のエラーです。
FAILURE_SCENESTARTWITH	<scene>タグの value 属性に不正な値が記載されています。自アプリのパッケージ名から始まらないシーンを指定している場合のエラーです。
FAILURE_SCENETAGNOT	<scene>タグが記載されていません。 もしくは、XML として不正フォーマットのため <scene>タグを見つけることができません。
FAILURE_SITUATIONCOUNT	<situation>タグの数が多すぎます。ホームシナリオのみのエラーです。
FAILURE_VERSION	<version>タグの value 属性に不正な値が記載されています。 同名の HVML が既に登録されていて、value 属性値が元の HVML より大きい場合のみ HVML が更新されます。value 属性値を変更するか、または元のシナリオを解除後に登録してください。

6.2 シーンの登録・解除

アプリのシナリオを実行するためには、アプリシナリオに記述しているシーンを有効にする必要があります。
シーンの詳細については、[参考資料\[3\]](#)の「4.20 scene タグ」を参照ください。シーン名の命名ルールについては「[5.4 シーン名称](#)」を参照してください。

シーンにより選択されるシナリオを絞り込むことにより、下記の効果が得られます。

- ・ レスポンスの向上
- ・ 他アプリのシナリオとの競合を回避

シーンを登録・解除するには、一連の処理をサンプルアプリの voiceui/VoiceUIManagerUtil#enableScene()および#disableScene()にて実装しておりますので、こちらを利用ください。

以下、サンプルアプリを参考に利用手順と解説を記載します。サンプルは SDK 内に同梱されております。

【SampleSimple: jp_co_sharp_sample_simple_accost.hvml】

```
<?xml version="1.0" ?>
<hvm1 version="2.0">
  <head>
    <producer>jp.co.sharp.sample.simple</producer>
    <description>Java 側からのトリガーで発話する</description>
    <scene value="jp.co.sharp.sample.simple.scene_common"/>
    <version value="1.0"/>
  ~以下省略~
```

① シナリオでシーンを記述します。

【SampleSimple:MainActivity.java】

```
@Override
public void onResume() {
  ~中略~
  //Scene 有効化.
  VoiceUIManagerUtil.enableScene(mVoiceUIManager, ScenarioDefinitions.SCENE_COMMON);
  ~中略~
}

@Override
public void onPause () {
  ~中略~
  //Scene 無効化.
  VoiceUIManagerUtil.disableScene(mVUIManager, ScenarioDefinitions.SCENE_COMMON);
  ~中略~
}
```

- ② アプリ起動時 (Activity の onResume 時) にシナリオに記述しているシーン (ここでは "jp.co.sharp.sample.simple.scene_common") を登録します。
- ③ アプリ終了時 (Activity の onPause 時) にシーンを解除します。ここでは Activity のライフサイクルに合わせてシーンの登録・解除をしておりますが各アプリの仕様に合わせて適切に登録・解除してください。

以下、VoiceUIManagerUtil クラスで実装しているシーン登録・解除の説明です。

【SampleSimple:VoiceUIManagerUtil.java】

```
public static int enableScene(VoiceUIManager vm, final String scene) {
  ~中略~
  VoiceUIVariable variable = new VoiceUIVariable(ScenarioDefinitions.TAG_SCENE, scene);
  variable.setExtraInfo(VoiceUIManager.SCENE_ENABLE);
  ArrayList<VoiceUIVariable> listVariables = new ArrayList<>();
  listVariables.add(variable);
  try {
    result = vm.updateAppInfo(listVariables);
  } catch (RemoteException e) {
    ~中略~
  }

  public static int disableScene(VoiceUIManager vm, final String scene) {
    ~中略~
    VoiceUIVariable variable = new VoiceUIVariable(ScenarioDefinitions.TAG_SCENE, scene);
    variable.setExtraInfo(VoiceUIManager.SCENE_DISABLE);
    ArrayList<VoiceUIVariable> listVariables = new ArrayList<VoiceUIVariable>();
    listVariables.add(variable);
    try {
      result = vm.updateAppInfo(listVariables);
    } catch (RemoteException e) {
      ~中略~
    }
  }
```

- ① 文字列 "scene" および登録・解除するシーン名を格納した VoiceUIVariable 変数を作成します。
- ② シーン登録フラグ (VoiceUIManager.SCENE_ENABLE/ SCENE_DISABLE) を設定します。
- ③ 設定内容を VoiceUIVariable のリスト型にして音声 UI に通知します。

6.3 発話の開始・中止

アプリから発話の開始と中止を行うことができます。

一連の処理をサンプルアプリの voiceui/VoiceUIManagerUtil# startSpeech ()および stopSpeech()にて実装しておりますので、こちらを利用ください。

以下、サンプルアプリを参考に利用手順と解説を記載します。サンプルは SDK 内に同梱されております。

【SampleSimple:jp_co_sharp_sample_simple_accost.hvml】

```
<?xml version="1.0" ?>
<hvml version="2.0">
  <head>
    <producer>jp.co.sharp.sample.simple</producer>
    <description>Java 側からのトリガーで発話する</description>
    <scene value="jp.co.sharp.sample.simple.scene_common"/>
    <version value="1.0"/>
    <accost priority="75" topic_id="t1" word="jp.co.sharp.sample.simple.accost.t1"/>
  </head>
  <body>
    <topic id="t1" listen="false">
      <action index="1">
        <speech>アプリから発話開始するサンプルだよ</speech>
        <behavior id="assign" type="normal"/>
      </action>
    </topic>
  </body>
</hvml>
```

- ① <accost>タグに発話を実行するためのtopic_idおよびwordを定義します。Wordの命名ルールについては「[5.5 Accost 名称](#)」を参照ください。
- ② Accost 実行時に遷移する topic を記述します。

【SampleSimple:MainActivity.java】

```
protected void onCreate(Bundle savedInstanceState) {
  ~中略~

  // accost ボタン
  Button voiceAccostButton = (Button)findViewById(R.id.voice_accost_button);
  voiceAccostButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
      VoiceUIManagerUtil.startSpeech(mVUIManager, ScenarioDefinitions.ACC_ACCOST);
    }
  });
  ~中略~
  @Override
  public void onPause() {
    super.onPause();
    //バックに回ったら発話を中止する。
    VoiceUIManagerUtil.stopSpeech();
  }
  ~中略~
}
```

- ③ ボタン押下を契機に accost タグに記載した word 名(ここでは“jp.co.sharp.sample.simple.accost.t1”)を指定して VoiceUIManagerUtil#startSpeech()を実行します。
- ④ アプリ終了時には発話を中止するように onPause()で VoiceUIManagerUtil#stopSpeech()を実行します。

以下、VoiceUIManagerUtil クラスで実装している発話開始・中止処理の説明です。

【SampleSimple:VoiceUIManagerUtil.java】

```
public static int startSpeech(VoiceUIManager vm, final String accost) {  
  ~中略~  
  VoiceUIVariable variable = new VoiceUIVariable(TAG_ACCOST, accost);  
  ArrayList<VoiceUIVariable> variables = new ArrayList<VoiceUIVariable>();  
  variables.add(variable);  
  try {  
    result = vm.updateAppInfoAndSpeech(variables);  
  } catch (RemoteException e) {  
    ~中略~  
  }  
  
  public static void stopSpeech() {  
    try {  
      VoiceUIManager.stopSpeech();  
    } catch (RemoteException e) {  
      ~中略~  
    }  
  }  
}
```

- ① 文字列“accost”および accost 名を格納した VoiceUIVariable 変数を作成します。
- ② 設定内容を VoiceUIVariable のリスト型にして音声 UI に通知します。
- ③ 発話中止は VoiceUIManager#stopSpeech()を実行します。

6.4 記憶の更新・削除

アプリから記憶(memory_p)の更新と削除を行うことができます。

一連の処理をサンプルアプリの voiceui/VoiceUIManagerUtil# setMemory ()および clearMemory ()にて実装しておりますので、こちらを利用ください。

以下、サンプルアプリを参考に利用手順と解説を記載します。サンプルは SDK 内に同梱されております。

【SampleSimple:MainActivity.java】

```
// set memory_p ボタン
Button getMemoryPButton = (Button)findViewById(R.id.set_memoryP);
getMemoryPButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Calendar now = Calendar.getInstance();
        final String hour = String.valueOf(now.get(Calendar.HOUR_OF_DAY));
        final String minute = String.valueOf(now.get(Calendar.MINUTE));
        int ret = VoiceUIManagerUtil.setMemory(mVUIManager, ScenarioDefinitions.MEM_P_HOUR, hour);
        ~中略~
        ret = VoiceUIManagerUtil.setMemory(mVUIManager, ScenarioDefinitions.MEM_P_MINUTE, minute);
        ~中略~
    }
});
```

- ① memory_p の key 名 (\${memory_p:key} の key 部分) と、格納する値 (文字列) を引数にして VoiceUIManagerUtil#setMemory() を実行します。
- ② 削除する場合は memory_p の key 名を引数に VoiceUIManagerUtil#clearMemory() を実行します。

【SampleSimple:jp_co_sharp_sample_simple_get_memoryp.hvml】

```
~中略~
<topic id="t2" listen="false">
    <action index="1">
        <speech>${memory_p:jp.co.sharp.sample.simple.hour}時${memory_p:jp.co.sharp.sample.simple.minute}分だ
        よ</speech>
        <behavior id="assign" type="normal"/>
    </action>
</topic>
```

- ③ セットした値は memory_p タグで参照できます。詳細は[参考資料\[3\]](#)の「4.16 memory タグ」を参照ください。

以下、VoiceUIManagerUtil クラスで実装している記憶の更新・削除処理の説明です。

【SampleSimple:VoiceUIManagerUtil.java】

```

public static int setMemory(VoiceUIManager vm, final String key, final String value) {
~中略~
    if (vm == null || key == null || "".equals(key)) {
        return result;
    } else {
        name = ScenarioDefinitions.TAG_MEMORY_P + key;
    }
    VoiceUIVariable variable = new VoiceUIVariable(name, value);
    ArrayList<VoiceUIVariable> variables = new ArrayList<VoiceUIVariable>();
    variables.add(variable);
    try {
        result = vm.updateAppInfo(variables);
    } catch (RemoteException e) {
~中略~
    }

    public static int clearMemory(VoiceUIManager vm, final String key) {
        int result = VoiceUIManager.VOICEUI_ERROR;
        if (vm != null) {
            try {
                result = vm.removeVariable(key);
            } catch (RemoteException e) {
~中略~
            }
        }
    }

```

- ① VoiceUIManager# updateAppInfo()で渡す変数名は"memory_p:key 名"の形で渡す必要があるため、引数で渡された key に"memory_p:"を付与します。
- ② 変数名"memory_p:key"および value を格納した VoiceUIVariable 変数を作成します。
- ③ 設定内容を VoiceUIVariable のリスト型にして音声 UI に通知します。
- ④ 記憶削除時は、VoiceUIManager# removeVariable ()を利用します。この時の引数は"memory_p:"をつけない key 名を渡します。

6.5 シナリオからの通知

アプリは音声認識結果や記憶などシナリオ上で取得できる値や、発話の終わりキャンセルなどの契機を通知で受けることができます。通知を受けるためには、VoiceUIListener を利用します。

一連の実装は、サンプルアプリの voiceui/ VoiceUIListenerImpl.java および MainActivity.java で実装されておりますのでそのまま流用してください。以下、サンプルアプリを参考に利用手順と解説を記載します。サンプルは SDK 内に同梱されております。

【SampleSimple:MainActivity.java】

```
public class MainActivity extends Activity implements VoiceUIListenerImpl.ScenarioCallback {
```

```
    private VoiceUIManager mVUIManager = null;
```

```
    private VoiceUIListenerImpl mVUIListener = null;
```

```
    @Override
```

```
    public void onResume() {
```

~中略~

```
        //VoiceUIManager インスタンス生成.
```

```
        if (mVUIManager == null) {
```

```
            mVUIManager = VoiceUIManager.getService(getApplicationContext());
```

```
        }
```

```
        //VoiceUIListener インスタンス生成.
```

```
        if (mVUIListener == null) {
```

```
            mVUIListener = new VoiceUIListenerImpl(this);
```

```
        }
```

```
        //VoiceUIListener の登録.
```

```
        VoiceUIManagerUtil.registerVoiceUIListener(mVUIManager, mVUIListener);
```

~中略~

```
    }
```

```
    @Override
```

```
    public void onPause() {
```

~中略~

```
        //VoiceUIListener の解除.
```

```
        VoiceUIManagerUtil.unregisterVoiceUIListener(mVUIManager, mVUIListener);
```

~中略~

```
    }
```

```
    @Override
```

```
    public void onScenarioEvent(int event, List<VoiceUIVariable> variables) {
```

```
        switch (event) {
```

```
            //必要なイベント毎に実装.
```

```
            case VoiceUIListenerImpl.ACTION_END:
```

```
~中略~
        case VoiceUIListenerImpl.RESOLVE_VARIABLE:
~中略~
        default:
            break;
    }
}
```

- ① VoiceUIListener を利用する Activity で VoiceUIListenerImpl.ScenarioCallback を implement します。VoiceUIListenerImpl クラスは VoiceUIListener の通知を受けて Activity にコールバックを送るだけのクラスです。必要な処理は Activity 側で実装してください。
- ② アプリ起動時(onResume 時)に VoiceUIManage のインスタンスを生成します。
- ③ VoiceUIListenerImpl のインスタンスを生成します。
- ④ 音声 UI に Listener を登録します。
- ⑤ アプリ終了時(onPause 時)には Listener の解除を行います。ここでは Activity のライフサイクルに合わせて登録・解除しておりますが、各アプリの仕様に合わせて適切に登録・解除してください。
- ⑥ シナリオからの通知を受けて実際に行う処理を実装します。**通信やファイルアクセスなどの時間を要する処理は別スレッドで処理するなどして本コールバックはすぐに抜ける必要があります。**

6.5.1 提供コールバック一覧

提供コールバック一覧と、その通知タイミングについて下記サンプルシナリオを例に記載します。

表 6-2 コールバック一覧

コールバック	メソッド	通知タイミング(サンプルシナリオ上の例)
シナリオ選択時に通知	onVoiceUIEvent()	処理②の先頭のタイミングで、<control>タグを含む<action>タグへの遷移時に該当
アクションキャンセル通知	onVoiceUIActionCancelled()	処理②③④の<action>タグ処理中に、キャンセル処理(※)が割込んだタイミング
アクション完了通知.	onVoiceUIActionEnd()	処理②の最後のタイミングで、<control>タグを含む<action>タグの完了時に該当
変数解決通知	onVoiceUIResolveVariable()	“jp.co.sharp.sample.simple”で始まる未解決変数を含む、処理①③に遷移したタイミング
発話棄却通知	onVoiceUIRejection()	処理②③④の<action>タグ処理中に、棄却処理(※)が割込んだタイミング
スケジュール登録結果通知	onVoiceUISchedule()	—

【コールバックタイミング HVML サンプル】

<pre> <?xml version="1.0" ?> <hvm1 version="2.0"> <head> <producer>jp.co.sharp.sample.simple</producer> <scene value="jp.co.sharp.sample.simple"/> <version value="1.0"/> <situation priority="75" topic_id="1" trigger="user-word">タグを通知 in \${Lvcsr:Basic}</situation> <situation priority="75" topic_id="2" trigger="user-word">変数を解決 in \${Lvcsr:Basic}</situation> <situation priority="75" topic_id="3" trigger="user-word">変数を含む in \${Lvcsr:Basic} and \${jp.co.sharp.sample.simple:situation_value} eq テスト</situation> </pre>	処理①
<pre> </head> <body> <topic id="1" listen="false"> <action index="1"> <speech>アプリに control タグを通知するね</speech> <behavior id="assign" type="normal"/> <control function="accost_sample" target="jp.co.sharp.sample.simple"/> </action> </pre>	処理②
<pre> </topic> <topic id="2" listen="false"> <action index="1"> <speech>アプリから取得した変数の値は\${jp.co.sharp.sample.simple:speech_value}だよ</speech> <behavior id="assign" type="normal"/> </action> </pre>	処理③
<pre> </topic> <topic id="3" listen="false"> <action index="1"> <speech>アプリから取得した変数の値がテストだったよ</speech> <behavior id="assign" type="normal"/> </action> </topic> </body> </hvm1> </pre>	処理④

6.5.1.1 onVoiceUIEvent()

選択されたシナリオ内の<action>タグに、<control>タグが含まれている場合にコールバックされます。ほとんどの場合はアクション完了通知時にアプリ側での処理を実行することになりますので、本コールバックは例えば、ロボホンの発話と同時にアプリ側で何か処理を実行する必要がある時などに利用ください。実装する場合は「[onVoiceUIActionEnd\(\)](#)」などを参照ください。

6.5.1.2 onVoiceUIActionCancelled()

優先度の高いシナリオが割り込まれた場合や、自アプリから VoiceUIManager#stopSpeech()により発話中止した場合などにコールバックされます。本コールバックが通知される条件については、[参考資料 \[2\]](#)を参照ください。

6.5.1.3 onVoiceUIActionEnd()

<control>タグが含まれるシナリオの、アクションが完了した場合にコールバックされます。<action>タグで囲まれた一連の処理の終了時に、<control>タグの function 属性が target に通知されます。

サンプルソースを以下に記載します。サンプルは SDK 内に同梱されております。

【SampleSimple:jp_co_sharp_sample_simple_app_end.hvml】

```

~中略~
<action index="1">
  <speech>サンプルアプリを終了するね</speech>
  <behavior id="assign" type="normal"/>
  <control function="end_app" target="jp.co.sharp.sample.simple"/>
</action>
~中略~

```

【SampleSimple:jp_co_sharp_sample_simple_talk.hvml】

```

<action index="1">
~中略~
  <control function="recog_talk" target="jp.co.sharp.sample.simple">
    <data key="Lvcsr_Basic" value="{Lvcsr:Basic}"/>
  </control>
</action>
~中略~

```

- ① <control>タグの function 属性に任意の文字列“end_app”や“recog_talk”、 target 属性に通知するアプリのパッケージ名“jp.co.sharp.sample.simple”を設定します。

【SampleSimple:MainActivity.java】

```

@Override
public void onScenarioEvent(int event, List<VoiceUIVariable> variables) {
    switch (event) {
        //必要なイベント毎に実装.
        case VoiceUIListenerImpl.ACTION_END:
            String function = VoiceUIVariableUtil.getVariableData(variables, ScenarioDefinitions.ATTR_FUNCTION);
            if(FUNC_END_APP.equals(function)) {
                finish();
                break;
            }else if(FUNC_RECOG_TALK.equals(function)){
                final String lvcsr = VoiceUIVariableUtil.getVariableData(variables,
ScenarioDefinitions.KEY_LVCSR_BASIC);
                mHandler.post(new Runnable() {
                    @Override
                    public void run() {
                        if(!isFinishing()) {
                            ((TextView) findViewById(R.id.recog_text)).setText("Lvcsr:"+lvcsr);
                        }
                    }
                });
            }
            break;
        ~中略~
    }
}

```

- ② 通知を受けたアプリはコールバック(onScenarioEvent)で処理を実装します。

6.5.1.4 onVoiceUIResolveVariable()

シナリオ内に変数を記述した場合に、その変数の値の解決のためにコールバックされます。変数名のフォーマットについては[参考資料\[3\]](#)「6.4 変数の追加」を参照ください。

サンプルソースを以下に記載します。サンプルは SDK 内に同梱されております。

【SampleSimple:jp_co_sharp_sample_simple_variable.hvml】

```

~中略~
<action index="1">
    <speech>アプリ側から取得した情報は${jp.co.sharp.sample.simple:java_side_value}だよ</speech>
    <behavior id="assign" type="normal"/>
</action>
~中略~

```

- ① 変数"jp.co.sharp.sample.simple:java_side_value"を定義します。通知を受けたアプリはコールバック

(onScenarioEvent)で処理を実装します。サンプルでは文字列“java”を返しています。

【SampleSimple:MainActivity.java】

```
@Override
public void onScenarioEvent(int event, List<VoiceUIVariable> variables) {
    switch (event) {
        //必要なイベント毎に実装.
~中略~
        case VoiceUIListenerImpl.RESOLVE_VARIABLE:
            for (VoiceUIVariable variable : variables) {
                String key = variable.getName();
                if (ScenarioDefinitions.RESOLVE_JAVA_VALUE.equals(key)) {
                    variable.setStringValue("java");
                }
            }
            break;
~中略~
    }
}
```

- ② 通知を受けたアプリはコールバック(onScenarioEvent)で処理を実装します。

6.5.1.5 onVoiceUIRejection()

現在発話中のシナリオより、優先度が低いなどで発話が棄却された場合にコールバックされます。

HVML とアプリのサンプルソースを以下に記載します。サンプルは SDK 内に同梱されております。

サンプルソースでは、アプリ終了発話の accost がリジェクトされたとしてもアプリを終了させるため、コールバックを受けて終了処理に遷移しています。

6.5.1.6 onVoiceUISchedule()

Override は必要ですが、中身の実装は必要ありません。

6.6 音声認識・音声発話の言語切替

アプリは自アプリ起動中の音声認識の言語と、音声発話の言語を切り替えることができます。

一連の処理をサンプルアプリの voiceui/VoiceUIManagerUtil# setAsr ()および setTts()にて実装しておりますので、こちらを利用ください。

言語を切替えるのはアプリが起動している間(Activity 起動中)のみとしてください。アプリ終了時には必ずデフォルトの言語設定に戻す対応を入れてください。

フェールセーフとして HOME 画面中に頭ボタンを押下するとデフォルトの言語設定に戻す対応をシステム側で行っていますのでご注意ください。

以下、サンプルアプリを参考に利用手順と解説を記載します。サンプルは SDK 内に同梱されております。

【SampleMultilingual:MainActivity.java】

```
@Override
public void onResume() {
~中略~
    //VoiceUIManager インスタンス生成.
    if (mVUIManager == null) {
        mVUIManager = VoiceUIManager.getService(getApplicationContext());
    }
~中略~
}

@Override
protected void onCreate(Bundle savedInstanceState) {
~中略~
    //日本語ボタンの実装.
    Button ButtonJ = (Button) findViewById(R.id.button_japanese);
    ButtonJ.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            VoiceUIManagerUtil.setAsr(mVUIManager, Locale.JAPAN);
            VoiceUIManagerUtil.setTts(mVUIManager, Locale.JAPAN);
        }
    });
~中略~
}

@Override
public void onPause() {
~中略~
```

```
//デフォルトの言語設定に戻す
Locale locale = Locale.getDefault();
VoiceUIManagerUtil.setAsr(mVUIManager, locale);
VoiceUIManagerUtil.setTts(mVUIManager, locale);
}
```

- ① アプリ起動時に音声 UI のインスタンスを生成します。
- ② VoiceUIManagerUtil# setAsr ()で音声認識の言語を指定します。
- ③ VoiceUIManagerUtil# setTts ()を音声発話の言語を指定します。
- ④ アプリ終了時(onPause 時)にデフォルトの言語を取得して、ASR/TTS をデフォルト言語に指定する。

以下、VoiceUIManagerUtil クラスで実装している setAsr ()/setTts ()の説明です。

【SampleMultilingual:VoiceUIManagerUtil.java】

```
public static int setAsr(VoiceUIManager vm, Locale locale) {
~中略~
    if(locale == null){
        country = Locale.getDefault().getCountry();
    }else{
        country = locale.getCountry();
    }

    if(LOCALE_COUNTRY_JAPAN.equals(country)){
        lang = VoiceUIManager.LANG_JAPANESE;
    }else if(LOCALE_COUNTRY_US.equals(country)){
        lang = VoiceUIManager.LANG_ENGLISH;
    }else if(LOCALE_COUNTRY_CHINA.equals(country) || LOCALE_COUNTRY_TAIWAN.equals(country)){
        lang = VoiceUIManager.LANG_CHINESE;
    }else if(LOCALE_COUNTRY_KOREA.equals(country)){
        lang = VoiceUIManager.LANG_KOREAN;
    }else{
        Log.w(TAG, "setAsr() : unexpected language");
        lang = VoiceUIManager.LANG_JAPANESE;
    }
    try {
        result = vm.setAsrLanguage(lang);
    } catch (RemoteException e) {
~中略~
    }
}
```

- ① 指定された Locale から getCountry()メソッドで国コードを取得します。Locale.getString()の値が第 1 世代と第 2 世代で異なるためです。詳細は「[4.11 多言語対応](#)」を参照ください。
- ② 国コード別に指定する言語の値を決定します。簡体字と繁体字は同一(VoiceUIManager.LANG_CHINESE)の値です。
- ③ VoiceUIManager#setAsrLanguage ()/setTtsLanguage()を実行します。

7. プロジェクター

本章では、アプリにおけるプロジェクターの利用方法について記載します。プロジェクターは第 1 世代ロボホンのみ利用できます。

プロジェクターを使うには、プロジクターマネージャーを利用します。プロジェクターマネージャーのインターフェース仕様の詳細は、[参考資料 \[2\]](#)を参照してください。

プロジェクター使用中は、レーザー光が意図しない方向に照射されることを防ぐため、モーションはできません。

プロジェクターマネージャーが提供する機能は以下の通りです。

- ・ プロジェクター開始
- ・ プロジェクター終了
- ・ プロジェクターの状態変化の通知

プロジェクターを利用するためには AndroidManifest.xml に以下のような記述を追加する必要があります。サンプルは SDK 内に同梱されております。

【SampleProjector: AndroidManifest.xml】

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="jp.co.sharp.sample.projector" >
  ~中略~
    <uses-permission    android:name="jp.co.sharp.android.rb.projectormanager.permission.ACCESS_PROJECTOR"
  />    . . . ①

  ~中略~
    <application
  ~中略~
      <activity
        android:name=".MainActivity"
        android:label="@string/app_name" >
          <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
            <category android:name="jp.co.sharp.android.rb.intent.category.LAUNCHER" />
          </intent-filter>
          <meta-data android:name="use_projector" android:value="MainActivity" />    . . . ②
        </activity>
  ~中略~
      <uses-library
        android:name="jp.co.sharp.android.rb.projector.framework"
        android:required="true"/>    . . . ③
      </application>
</manifest>

```

- ① <manifest> タグ内に <uses-permission> で
"jp.co.sharp.android.rb.projectormanager.permission.ACCESS_PROJECTOR"を定義します。
- ② プロジェクター照射を行う Activity の<activity> タグ内に<meta-data>で"use_projector"を定義します。Value
値は Activity 名としてください。
- ③ <application> タグ内に<uses-library> で"jp.co.sharp.android.rb.projector.framework"を定義します。

7.1 プロジェクター開始

プロジェクターは Android 標準の `startService()` によりプロジェクターマネージャーサービスを起動させることにより利用できます。

プロジェクター開始のサンプルソースを以下に記載します。サンプルは SDK 内に同梱されております。

【SampleProjector: MainActivity.java】

```
import jp.co.sharp.android.rb.projectormanager.ProjectorManagerServiceUtil; ... ①
~中略~
@Override
public void onExecCommand(String command, List<VoiceUIVariable> variables) {
~中略~
    case ScenarioDefinitions.FUNC_START_PROJECTOR:
        //プロジェクタマネージャの開始.
        if(!isProjected) {
            startService(getIntentForProjector()); ... ②
        }
~中略~
}

private Intent getIntentForProjector() {
    Intent intent = new Intent();
    ComponentName componentName = new ComponentName(
        ProjectorManagerServiceUtil.PACKAGE_NAME,
        ProjectorManagerServiceUtil.CLASS_NAME); ... ③
    //逆方向で照射する
    intent.putExtra(ProjectorManagerServiceUtil.EXTRA_PROJECTOR_OUTPUT,
ProjectorManagerServiceUtil.EXTRA_PROJECTOR_OUTPUT_VAL_REVERSE); ... ④
    intent.setComponent(componentName); ... ⑤
    return intent;
}
```

- ① プロジェクターマネージャー利用定義を持つ拡張クラスを import します
- ② プロジェクターマネージャー開始の開始要求を `startService()` にて行います。サンプルソースでは音声 UI からのコールバックを受けて、プロジェクター照射中でなければプロジェクターを開始するように実装しています。
- ③ Intent に設定する ComponentName 情報の設定を指定します。
- ④ Intent に Extra 情報を付与することにより、プロジェクターの照射方向を指定できます。
※上記はロボホンから見て逆方向に照射する場合の設定です。(設定しない場合はデフォルト方向となります)
ロボホンから見て正方向に照射したい場合は `EXTRA_PROJECTOR_OUTPUT_VAL_NORMAL` を設定してください。
- ⑤ 指定した ComponentName を Intent にセットします。

7.2 プロジェクター終了

アプリからの終了要求は、Android 標準の `stopService()` により行うことができます。しかし、プロジェクターマネージャーは「プロジェクターを終了して」などの音声コマンドによるプロジェクターの終了や、アプリの切替りを検出し、meta-data の取得を行い遷移先のアプリに定義が無い場合は自動でプロジェクターの終了を実行しています。アプリの切替り先で継続してプロジェクターを利用するケースも想定されるので、アプリ終了時などに明示的に終了処理は行わないでください。

ただし、アプリにプロジェクター停止ボタンを UI パーツとして配置し UI 操作にてプロジェクターを終了する実装としたい場合や、同一アプリ内の Activity 間遷移時にプロジェクターを終了したい場合などは除きます。その場合は「[プロジェクター開始/停止アイコン表示](#)」にしたがってボタンを表示してください。

プロジェクター終了のサンプルソースを以下に記載します。サンプルは SDK 内に同梱されております。

【SampleProjector: MainActivity.java】

```
～中略～
@Override
public void onClick(View v) {
    switch(v.getId())
    {
        case R.id.action_projector:
            if(isProjected ){
                //プロジェクター停止
                stopService(getIntentForProjector()); ・・・①
            } else {
                //プロジェクター開始
                startService(getIntentForProjector());
            }
            break;
        default:
            break;
    }
}
～中略～
```

- ① プロジェクター開始時に指定したものと同一 Intent を設定し `stopService()` を行います。サンプルソースでは Action ボタン押下で、プロジェクター照射中であれば終了するように実装しています。

7.3 プロジェクターの状態変化の通知

プロジェクターマネージャーは開始/終了などの状態変化を `BroadcastIntent` にて通知を行います。

そのためアプリ側にて `BroadcastReceiver` を実装することにより、プロジェクターの状態変化に応じた処理を行うことができます。通知される状態変化の詳細は、[参考資料](#) [2]を参照してください。

プロジェクターの状態変化取得のサンプルソースを以下に記載します。サンプルは SDK 内に同梱されております。

【SampleProjector:MainActivity.java】

```

~中略~
@Override
protected void onCreate(Bundle savedInstanceState) {
~中略~
    //プロジェクタイベントの検知登録.
    setProjectorEventReceiver();    ... ①
}

@Override
protected void onDestroy() {
~中略~
    //プロジェクタイベントの検知破棄.
    this.unregisterReceiver(mProjectorEventReceiver);    ... ②
}

private void setProjectorEventReceiver() {
~中略~
    IntentFilter intentFilter = new IntentFilter();
    intentFilter.addAction(ProjectorManagerServiceUtil.ACTION_PROJECTOR_PREPARE);
    intentFilter.addAction(ProjectorManagerServiceUtil.ACTION_PROJECTOR_START);
    intentFilter.addAction(ProjectorManagerServiceUtil.ACTION_PROJECTOR_PAUSE);
    intentFilter.addAction(ProjectorManagerServiceUtil.ACTION_PROJECTOR_RESUME);
    intentFilter.addAction(ProjectorManagerServiceUtil.ACTION_PROJECTOR_END);
    intentFilter.addAction(ProjectorManagerServiceUtil.ACTION_PROJECTOR_END_ERROR);
    intentFilter.addAction(ProjectorManagerServiceUtil.ACTION_PROJECTOR_END_FATAL_ERROR);
    intentFilter.addAction(ProjectorManagerServiceUtil.ACTION_PROJECTOR_TERMINATE);
    registerReceiver(mProjectorEventReceiver, intentFilter);    ... ③
}

private class ProjectorEventReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {    ... ④
        Log.v(TAG, "ProjectorEventReceiver#onReceive():" + intent.getAction());
        switch (intent.getAction()) {

```

```

        case ProjectorManagerServiceUtil.ACTION_PROJECTOR_PREPARE:
        case ProjectorManagerServiceUtil.ACTION_PROJECTOR_PAUSE:
        case ProjectorManagerServiceUtil.ACTION_PROJECTOR_RESUME:
            break;
        case ProjectorManagerServiceUtil.ACTION_PROJECTOR_START:
            acquireWakeLock();
            isProjected = true;
            break;
        case ProjectorManagerServiceUtil.ACTION_PROJECTOR_END:
        case ProjectorManagerServiceUtil.ACTION_PROJECTOR_END_FATAL_ERROR:
        case ProjectorManagerServiceUtil.ACTION_PROJECTOR_END_ERROR:
        case ProjectorManagerServiceUtil.ACTION_PROJECTOR_TERMINATE:
            releaseWakeLock();
            isProjected = false;
            break;
        default:
            break;
    }
}
}
}

```

- ① onCreate()にて Receiver を設定する。
- ② onDestroy()にて設定した、Receiver を破棄する。
- ③ 各種イベントの IntentFilter を追加する。
- ④ BroadcastReceiver の onReceive を実装する。サンプルソースではプロジェクター照射開始 (ACTION_PROJECTOR_START) を受けて WakeLock の取得、照射終了 (ACTION_PROJECTOR_END~ACTION_PROJECTOR_TERMINATE)を受けて、WakeLock を開放するよう実装しています。

7.4 プロジェクター利用時の注意事項

プロジェクターを利用する際の注意事項について記載します。

7.4.1 プロジェクター照射中の WakeLock について

プロジェクターマネージャーでは照射中に WakeLock をかけない仕様のため、一定時間無操作の画面 OFF 時はプロジェクター照射を終了します。一定時間以上プロジェクター照射を継続したい場合は、アプリ側にて適宜 WakeLock をかける必要があります。その際、設定する WakeLock は「SCREEN_DIM_WAKE_LOCK」としてください。

※SCREEN_DIM_WAKE_LOCK は deprecate されておりますが、プロジェクター照射中に背面の LCD 輝度を落とし、消費電力を低減するためにプロジェクター利用時のみ使用してください。

7.4.2 プロジェクター開始時の縦横切替えおよび、停止/再開時の考慮

プロジェクター照射時にはユーザ認証のためにカメラやロボナンバー入力画面などの割込みによる onPause および、縦表示から横表示への切替えにともなう Activity の再起動が発生します。図 7-1 参照。

また、プロジェクター照射中に照射位置の切替えが発生した場合は、顔検出 Activity が起動します。これにより onPause/onResume が発生します。

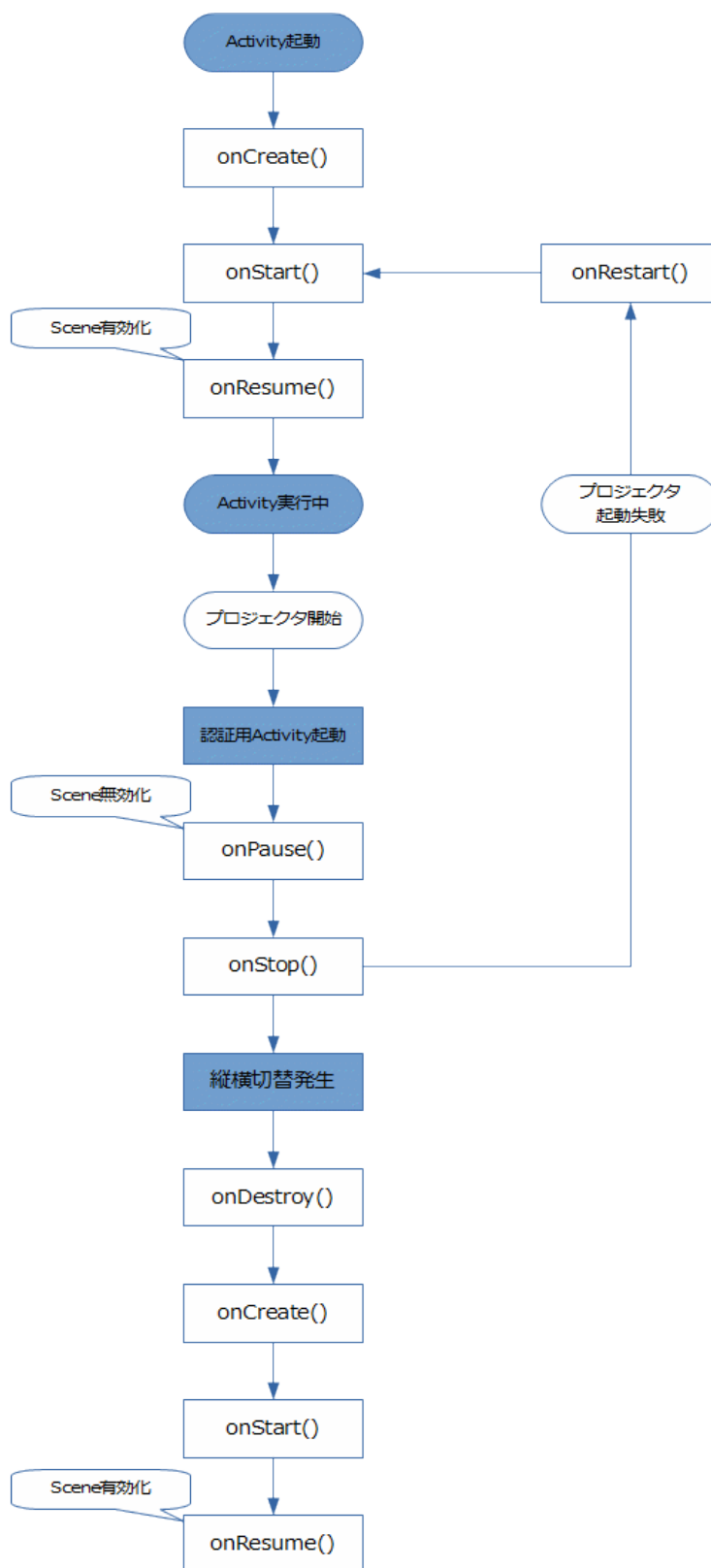




図 7-1 プロジェクター起動時のライフサイクル

7.4.3 プロジェクター開始/停止アイコン表示

プロジェクター開始および、プロジェクター停止を UI パーツとして配置する場合は、以下のアイコンを利用してください。リソースファイルは SDK 内に同梱されております。

表 7-1 プロジェクター用アイコン

UI 操作(file 名)	アイコン	備考
プロジェクター開始 (ic_projector_start.png)		ACTION_PROJECTOR_START を受けて停止ボタンに切り替えてください
プロジェクター停止 (ic_projector_stop.png)		ACTION_PROJECTOR_END 、 ACTION_PROJECTOR_END_ERROR 、 ACTION_PROJECTOR_END_FATAL_ERROR を受けて開始ボタンに切り替えてください

7.4.4 プロジェクター有無の考慮

プロジェクターは第 1 世代ロボホンのみ搭載されているため、プロジェクター機能を利用するアプリは第 1 世代ロボホンか第 2 世代ロボホンかを判別して、プロジェクター機能の有効・無効を制御する必要があります。

アプリ、シナリオでのサンプルソースを以下に記載します。サンプルは SDK 内に同梱されております。

【SampleProjector: MainActivity.java】

```
private void setupTitleBar() {
~中略~
    //第 1 世代ロボホンのみプロジェクターボタンを表示する.
    if(getRobohonGeneration() == 1){
        mImageButton.setOnClickListener(this);
    }else{
        mImageButton.setVisibility(View.INVISIBLE);
    }
}

private int getRobohonGeneration() {
~中略~
    int ret = -1;
    try {
        switch (android.os.Build.VERSION.SDK_INT){
            case 21:
                ret = 1;
                break;
            case 27:
```

```

        ret = 2;
        break;
    default:
        ret = -1;
        break;
    }
} catch(Exception e) {
~中略~
}

```

- ① プロジェクター用のボタンを第 1 世代の時のみ表示する。
- ② 第 1 世代、第 2 世代の判定は SDK のバージョンで判定します。21 が第 1 世代で、27 が第 2 世代です。

【SampleProjector: jp_co_sharp_sample_projector_projector.hvml】

```

<?xml version="1.0" ?>
<hvmml version="2.0">
    <head>
~中略~
        ${Lvcsr:Basic} include [プロジェクト,プロジェクト]
        and ${Lvcsr:Basic} include [開始,解し,会し,起動,移し,写し,映し]
        and ${env:projector_support} eq true
    </situation>
    </head>
~中略~
</hvmml>

```

- ① プロジェクターの起動コマンドの条件にプロジェクターサポートの条件`${env:projector_support}`を追加する。変数の詳細は[参考資料\[3\]](#)を参照ください。

8. 電話帳

本章では、アプリにおける電話帳情報（オーナー情報、ロボ情報を含む）の利用方法および、電話帳アプリの連携起動方法について記載します。

インターフェース仕様の詳細は、[参考資料 \[2\]](#)を参照してください。

本 API が提供する機能は以下の通りです。

- API 経由による、電話帳に登録した情報の取得
- API 経由による、オーナー情報の取得
- API 経由による、ロボ情報の取得
- Broadcast による、電話帳情報の新規登録完了通知・削除完了通知
- Broadcast による、電話帳情報・オーナー情報・ロボ情報の更新通知
- アプリ連携起動による、電話帳新規登録画面・追加登録画面・検索画面・削除画面の表示
- アプリ連携起動による、あなたについて画面の表示
- アプリ連携起動による、ロボ情報画面の表示

API 経由を利用するには AndroidManifest.xml に以下のような記述を追加する必要があります。

【SampleAddressBook: AndroidManifest.xml】

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="jp.co.sharp.sample.addressbook" >

    <uses-permission android:name="jp.co.sharp.android.rb.addressbook.permission.ACCESS_CONTACT" />

    <application
        android:allowBackup="false"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        ~中略~
        <uses-library android:name="jp.co.sharp.android.rb.addressbook.framework" android:required="true" />
        ~中略~
    </application>
</manifest>
```

①

②

① <manifest> タグ内に <uses-permission> を定義します。

② <application> タグ内に<uses-library> を定義します。

8.1 API 経由による情報利用

アプリで電話帳情報、オーナー情報、ロボ情報を利用するには、AddressBookManager を利用します。

オーナー情報クラスのインスタンスからオーナー名を取得するサンプルソースを以下に記載します。

【SampleAddressBook: MainActivityVoiceUIListener.java】

```
private class BtnGetOwnerClickListener implements View.OnClickListener {
    ~中略~
    public void onClick(View view) {
        AddressBookManager addressMng = AddressBookManager.getService(_context);
        String owner_name = _context.getString(R.string.default_name);
        try {
            OwnerProfileData ownerData = addressMng.getOwnerProfileData();
            String nickName = ownerData.getNickname();
            if(nickName != null && !"".equals(nickName)) {
                owner_name = nickName;
            }
            String firstName = ownerData.getFirstname();
            if(firstName != null && !"".equals(firstName)) {
                firstName = firstName + _context.getString(R.string.honorific);
                owner_name = firstName;
            }
            String lastName = ownerData.getLastname();
            if(lastName != null && !"".equals(lastName)) {
                lastName = lastName + _context.getString(R.string.honorific);
                owner_name = lastName;
            }
        } catch (RemoteException e) {
            e.printStackTrace();
        }
        TextView textView = (TextView)findViewById(R.id.txt_desc_owner_name);
        String text = _context.getString(R.string.text_description_owner_name) + owner_name;
        textView.setText(text);
    }
}
```

- ① AddressBookManager のインスタンスを取得します。
- ② オーナープロフィール情報のインスタンスを取得します。
- ③ オーナープロフィール情報からニックネームを取得します。
- ④ ニックネームが登録されていない場合は、よみがな（名）を取得します。
- ⑤ よみがな（名）も登録されていない場合は、よみがな（姓）を取得します。

8.2 Broadcast による通知

電話帳情報が新規登録や削除された通知をアプリは Broadcast で受けることができます。また同じように、電話帳情報やオーナー情報、ロボ情報の更新も Broadcast で通知を受けることができます。

更新完了通知を Broadcast で受け取り、オーナー情報が更新された場合に誕生日を画面に表示するサンプルソースを以下に記載します。

【SampleAddressBook: MainActivity.java】

```
protected void onCreate(Bundle savedInstanceState) {
    ～中略～

    //電話帳 Broadcast の検知登録。
    mAddressBookReceiver = new AddressBookReceiver();
    IntentFilter filterAddressBook = new IntentFilter("jp.co.sharp.android.rb.extra.ContactId_ACTION");
    registerReceiver(mAddressBookReceiver, filterAddressBook);
}

～中略～

// 電話帳イベントを受け取るための Broadcast レシーバー クラス。
private class AddressBookModReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        int contact_id = intent.getIntExtra
            (AddressBookCommonUtils.KEY_ADDRESSBOOK_INTENT_CONTACTID, 0);

        int intent_type = intent.getIntExtra
            (AddressBookCommonUtils.KEY_ADDRESSBOOK_INTENT_TYPE,
            AddressBookCommonUtils.ADDRESSBOOK_INTENT_UPDATE);

        if ((contact_id == AddressBookCommonUtils.CONTACT_ID_OWNER) &&
            (intent_type == AddressBookCommonUtils.ADDRESSBOOK_INTENT_UPDATE)) {
            Log.d(TAG, "Update CONTACT_ID_OWNER");
            TextView textSetting = (TextView)findViewById(R.id.txt_desc_birthday);
            textSetting.setText(GetOwnerBirth());
        } else {
            Log.d(TAG, "Not CONTACT_ID_OWNER");
        }
    }
}
```

- ① 電話帳 Broadcast を受信するレシーバーを作成し、登録します。
- ② 受信した intent から Extra キーでコンタクト ID を取得します。
- ③ Extra キーで通知タイプを取得します。
- ④ コンタクト ID がオーナーで、通知タイプが更新通知であれば、オーナー情報を表示します。

8.3 アプリ連携起動

アプリから電話帳新規登録画面・追加登録画面・検索画面・削除画面・あなたについての画面・ロボについての画面を起動することができます。

電話帳の新規登録画面をアプリから起動する時のサンプルソースを以下に記載します。

下記のサンプルは、よみがな（名）に “しゃーぷ” を設定した状態で新規登録画面起動します。

【SampleAddressBook: MainActivity.java】

```
protected void onCreate(Bundle savedInstanceState) {
```

```
    ~中略~
```

```
    public void onClick(View v) {
```

```
        //新規登録
```

```
        Button btnAppNew = (Button) findViewById(R.id.btn_app_new);
```

```
        btnAppNew.setOnClickListener(new View.OnClickListener() {
```

```
            @Override
```

```
            public void onClick(View v) {
```

```
                Intent intent = Intent.makeMainActivity(new ComponentName
```

```
                    (AddressBookCommonUtils.AUTHORITY_NAME,
```

```
                    AddressBookCommonUtils.AUTHORITY_CLASS_NAME_ENTRY));
```

```
                intent.putExtra(AddressBookCommonUtils.KEY_ADDRESSBOOK_ACTIVITY,
```

```
                    AddressBookCommonUtils.ADDRESSBOOK_ACTIVITY);
```

```
                intent.putExtra(AddressBookCommonUtils.KEY_ADDRESSBOOK_ENTRY_TYPE,
```

```
                    AddressBookCommonUtils.ADDRESSBOOK_ENTRY_NEW);
```

```
                intent.putExtra(AddressBookCommonUtils.KEY_FIRST_NAME, getString(R.string.sample_name));
```

```
                startActivityForResult(intent, REQ_CODE_NEW_ADDRESS);
```

```
            }
```

```
        });
```

```
    ~中略~
```

```
}
```

①

②

③

④

- ① 電話帳新規登録の Intent を作成します。
パッケージ名に電話帳パッケージ名の “AUTHORITY_NAME” を設定し、クラス名に電話帳新規登録クラス名の “AUTHORITY_CLASS_NAME_ENTRY” を設定します。
- ② Extra キーで起動先の画面を指定します。
- ③ Extra キーで登録タイプを設定します。
- ④ 登録したい情報を設定します。

9. カメラ

本章では、アプリからカメラ機能（顔認識、写真撮影、動画撮影）を利用する方法について記載します。本機能はロボホンのビルド番号 01.06.00 以降で利用できます。利用時は「設定 - 端末情報 - ビルド番号」よりビルド番号を確認してください。インターフェース仕様の詳細は、[参考資料 \[2\]](#)を参照してください。

本 API が提供する機能は以下の通りです。

- ・ アプリ連携起動による、人の顔や、ペットを探す機能(顔検出機能)
- ・ アプリ連携起動による、自動で写真を撮影する機能(静止画撮影機能)
- ・ アプリ連携起動による、自動で動画を撮影する機能(動画撮影機能)

※重要 周囲を見回しながらカメラを利用する場合、ロボホンは姿勢を変えて見回します。机の縁などで姿勢を変えた結果、転落するなどの危険性があるため、周囲を見回すカメラ機能を利用する場合は必ずユーザの指示によってのみカメラ機能を実行してください。

カメラを利用するためには AndroidManifest.xml に以下のような記述を追加する必要があります。顔検出機能を利用する場合は、permission の定義を追加してください。サンプルは SDK 内に同梱されております。

【SampleCamera: AndroidManifest.xml】

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="jp.co.sharp.sample.camera" >

    <uses-permission android:name="jp.co.sharp.android.rb.camera.permission.FACE_DETECTION" />

    <application
        ~中略~
        <uses-library android:name="jp.co.sharp.android.rb.cameralibrary" android:required="true" />
        ~中略~
    </application>
</manifest>
```

①

②

① <manifest> タグ内に <uses-permission> を定義します。

② <application> タグ内に<uses-library> を定義します。

9.1 顔検出機能の利用

アプリはカメラアプリを Broadcast Intent により連携起動することで顔検出機能を利用することができます。検出する時間、検出中に見回すかどうか、検出する対象(オーナー/電話帳登録されている人/ペット)を指定することもできます。また、検出した結果は、Broadcast Intent にて受け取ることができます。顔検出機能を連携起動させる時のサンプルコードを以下に記載します。

【SampleCamera: MainActivity.java】

```
public class MainActivity extends Activity implements MainActivityVoiceUIListener.MainActivityScenarioCallback {
    /** アプリ独自 Action : 顔認識結果通知. */
    public static final String ACTION_RESULT_FACE_DETECTION
        = "jp.co.sharp.sample.camera.action.RESULT_FACE_DETECTION";
    ~中略~

    protected void onCreate(Bundle savedInstanceState) {
        ~中略~
        // 顔認識 首振無ボタン
        Button faceRecogButton = (Button)findViewById(R.id.face_recog_button);
        faceRecogButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                sendBroadcast(getIntentForFaceDetection("FALSE"));
            }
        });
        ~中略~
    }
    ~中略~

    private Intent getIntentForFaceDetection(String swing) {
        Intent intent = new Intent(FaceDetectionUtil.ACTION_FACE_DETECTION_MODE);
        intent.setPackage(FaceDetectionUtil.PACKAGE);
        intent.putExtra(FaceDetectionUtil.EXTRA_REPLYTO_ACTION, ACTION_RESULT_FACE_DETECTION);
        intent.putExtra(FaceDetectionUtil.EXTRA_REPLYTO_PKG, getPackageName());
        intent.putExtra(FaceDetectionUtil.EXTRA_FACE_DETECTION_LENGTH,
            FaceDetectionUtil.EXTRA_FACE_DETECTION_LENGTH_NORMAL);
        intent.putExtra(FaceDetectionUtil.EXTRA_MOVE_HEAD, swing);
        return intent;
    }
}
```

- ① 顔認識結果取得用の Action を定義します。
- ② 顔認識起動用の Intent を Broadcast します。
- ③ 顔認識起動用 Intent のインスタンスを生成します。
- ④ ①で定義した顔認識結果取得用の Action 名を指定します。

- ⑤ 顔認識結果取得用 Intent の送信先に自 Package 名を指定します。
- ⑥ 顔検出を続ける時間を指定します。NORMAL で約 5 秒、LONG で約 10 秒検出を続けます。
- ⑦ 顔検出時に見回すかどうかを指定します。String 型 で"TRUE/FALSE"を指定します。

結果を受け取る時のサンプルコードを以下に記載します。

【SampleCamera: MainActivity.java】

```
public class MainActivity extends Activity implements MainActivityVoiceUIListener.MainActivityScenarioCallback {
    /** アプリ独自 Action : 顔認識結果通知. */
    public static final String ACTION_RESULT_FACE_DETECTION
        = "jp.co.sharp.sample.camera.action.RESULT_FACE_DETECTION";
    ~中略~

    protected void onCreate(Bundle savedInstanceState) {
        ~中略~
        //カメラ連携起動結果取得用レシーバー登録.
        mCameraResultReceiver = new CameraResultReceiver();
        IntentFilter filterCamera = new IntentFilter();
        filterCamera.addAction(ACTION_RESULT_FACE_DETECTION);
        registerReceiver(mCameraResultReceiver, filterCamera);
        ~中略~
    }

    private class CameraResultReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            switch(action){
                ~中略~
                case ACTION_RESULT_FACE_DETECTION:
                    result = intent.getIntExtra(FaceDetectionUtil.EXTRA_RESULT_CODE,
                                                FaceDetectionUtil.RESULT_CANCELED);
                    if(result == FaceDetectionUtil.RESULT_OK){
                        HashMap<String,String> hashMapFace =
                            (HashMap<String,String>)intent.getSerializableExtra(
                                FaceDetectionUtil.EXTRA_MAP_FACE_DETECTION);
                        for (String key : hashMapFace.keySet()) {
                            int contactId = Integer.valueOf(hashMapFace.get(key));
                        }
                        int pet = intent.getIntExtra(FaceDetectionUtil.EXTRA_PET_DETECTION, -1);
                    }
                ~中略~
            }
        }
    }
}
```

- ① 顔認識結果取得用の Action を定義します。
- ② 顔認識結果取得用 intent filter を設定します。
- ③ 顔認識結果の HashMap を取得します。

- ④ HashMap から ContactID を取得します。String 型で返るので必要に応じて int 型で利用します。オーナーが検出された時は 201、電話帳に登録されていない人を検出した時は-1、電話帳に登録されている人を検出した時は電話帳に登録されている ContactID が取得できます。

※注意 顔認識は毎フレーム毎に顔を検出および認識を実行しているため、同一人物を認識しているにもかかわらず、状況によっては-1 と登録済の Contact ID の複数の結果が返る場合があります。

- ⑤ ペット検出結果を取得します。

9.2 静止画の撮影

アプリはカメラアプリを Broadcast Intent により連携起動することで静止画撮影機能を利用することができます。撮影時に顔検出する・しないの設定、撮影する対象を Contact ID 指定することもできます。また、撮影した結果(ファイルパス)を Broadcast Intent にて受け取ることが可能です。

静止画撮影を開始する時のサンプルコードを以下に記載します。

【SampleCamera: MainActivity.java】

```
public class MainActivity extends Activity implements MainActivityVoiceUIListener.MainActivityScenarioCallback {
    /** アプリ独自 Action : 写真撮影結果通知 */
    public static final String ACTION_RESULT_TAKE_PICTURE =
        "jp.co.sharp.sample.camera.action.RESULT_TAKE_PICTURE";
    ~中略~

    protected void onCreate(Bundle savedInstanceState) {
        ~中略~
        // 写真撮影 顔認識有ボタン
        Button cameraFaceButton = (Button)findViewById(R.id.shoot_camera_face_button);
        cameraFaceButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                sendBroadcast(getIntentForPhoto(true));
            }
        });
        ~中略~
    }
    ~中略~

    private Intent getIntentForPhoto(boolean facedetect) {
        Intent intent = new Intent(ShootMediaUtil.ACTION_SHOOT_IMAGE);
        intent.setPackage(ShootMediaUtil.PACKAGE);
        intent.putExtra(ShootMediaUtil.EXTRA_FACE_DETECTION, facedetect);
        intent.putExtra(ShootMediaUtil.EXTRA_REPLYTO_ACTION, ACTION_RESULT_TAKE_PICTURE);
        intent.putExtra(ShootMediaUtil.EXTRA_REPLYTO_PKG, getPackageName());
        return intent;
    }
}
```

- ① 静止画結果取得用の Action を定義します。
- ② 静止画撮影起動用の Intent を Broadcast します。
- ③ 静止画撮影起動用 Intent のインスタンスを生成します。
- ④ 顔検出して撮影するかどうかを指定^②します。boolean 型で true/false を指定します。
- ⑤ ①で定義した静止画撮影結果取得用の Action を指定します。
- ⑥ 静止画撮影結果取得用 Intent の送信先に自 Package 名を指定します。

結果を取得する時のサンプルコードを以下に記載します。

【SampleCamera: MainActivity.java】

```
public class MainActivity extends Activity implements MainActivityVoiceUIListener.MainActivityScenarioCallback {
    /** アプリ独自 Action : 静止画撮影結果通知. */
    public static final String ACTION_RESULT_TAKE_PICTURE
        = "jp.co.sharp.sample.camera.action.RESULT_TAKE_PICTURE";
    ~中略~

    protected void onCreate(Bundle savedInstanceState) {
        ~中略~
        //カメラ連携起動結果取得用レシーバー登録.
        mCameraResultReceiver = new CameraResultReceiver();
        IntentFilter filterCamera = new IntentFilter();
        filterCamera.addAction(ACTION_RESULT_TAKE_PICTURE);
        registerReceiver(mCameraResultReceiver, filterCamera);
        ~中略~
    }

    private class CameraResultReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            switch(action){
                ~中略~
                case ACTION_RESULT_TAKE_PICTURE:
                    int result = intent.getIntExtra(ShootMediaUtil.EXTRA_RESULT_CODE,
                                                    ShootMediaUtil.RESULT_CANCELED);
                    if(result == ShootMediaUtil.RESULT_OK) {
                        String path = intent.getStringExtra(ShootMediaUtil.EXTRA_PHOTO_TAKEN_PATH);
                        TextView textView = (TextView)findViewById(R.id.text_file_path);
                        textView.setText(path);
                    }
                    break;
                ~中略~
            }
        }
    }
}
```

① 静止画撮影結果取得用の Action を定義します。

② 静止画撮影結果受信用の IntentFilter を設定します。

③ 撮影された写真のファイルパスを取得します。

- ① 静止画撮影結果取得用の Action を定義します。
- ② 静止画撮影結果受信用の IntentFilter を設定します。
- ③ 撮影された写真のファイルパスを取得します。

9.3 動画の撮影

アプリはカメラアプリを Broadcast Intent により連携起動することで動画撮影機能を利用することができます。また、撮影時間を指定しての自動終了させることも可能です。

動画撮影を開始する時のサンプルコードを以下に記載します。

【SampleCamera: MainActivity.java】

```
public class MainActivity extends Activity implements MainActivityVoiceUIListener.MainActivityScenarioCallback {
    /** アプリ独自 Action : 動画撮影結果通知 */
    public static final String ACTION_RESULT_REC_MOVIE =
        "jp.co.sharp.sample.camera.action.RESULT_REC_MOVIE";
    ~中略~

    protected void onCreate(Bundle savedInstanceState) {
        ~中略~
        // 動画撮影 10 秒間ボタン
        Button recordMovie10secButton = (Button)findViewById(R.id.record_movie_10sec_button);
        recordMovie10secButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                sendBroadcast(getIntentForVideo(10));
            }
        });
        ~中略~
    }
    ~中略~

    private Intent getIntentForVideo(int time) {
        Intent intent = new Intent(ShootMediaUtil.ACTION_SHOOT_MOVIE);
        intent.setPackage(ShootMediaUtil.PACKAGE);
        intent.putExtra(ShootMediaUtil.EXTRA_MOVIE_LENGTH, time);
        intent.putExtra(ShootMediaUtil.EXTRA_REPLYTO_ACTION, ACTION_RESULT_REC_MOVIE);
        intent.putExtra(ShootMediaUtil.EXTRA_REPLYTO_PKG, getPackageName());
        return intent;
    }
}
```

- ① 動画結果取得用の Action を定義します。
- ② 動画撮影起動用の Intent を Broadcast します。
- ③ 動画撮影起動用 Intent のインスタンスを生成します。
- ④ 動画撮影時間を指定します。0 もしくは本指定をしない場合は手動で録画停止する必要があります。
- ⑤ ①で定義した動画撮影結果取得用の Action を指定します。
- ⑥ 動画撮影結果取得用 Intent の送信先に自 Package 名を指定します。

結果の取得方法は、静止画とほぼ同様のため割愛します。

10. ダンス

本章では、アプリからダンスを実行する方法について記載します。本機能はロボホンのビルド番号 01.05.00 以降で利用できます。利用時は「設定 - 端末情報 - ビルド番号」よりビルド番号を確認してください。インターフェース仕様の詳細は、[参考資料 \[2\]](#)を参照してください。

本 API が提供する機能は以下の通りです。

- ・ ランダムに選択されたダンスを実行する
- ・ 最新のダンスを実行する
- ・ 前回と同じダンスを実行する
- ・ 指定のダンスを実行する
- ・ ダンスのメドレーを実行する
- ・ ダンスの一覧を取得する

ダンスを利用するためには AndroidManifest.xml に以下のような記述を追加する必要があります。サンプルは SDK 内に同梱されております。

【SampleDance: AndroidManifest.xml】

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="jp.co.sharp.sample.dance" >

    <application
        ~中略~
        <uses-library android:name="jp.co.sharp.android.rb.rbdance.framework" android:required="true" />
        ~中略~
    </application>
</manifest>
```

① <application> タグ内に<uses-library> を定義します。

10.1 ダンスの実行

アプリはダンスを Broadcast Intent により実行することができます。ただし、USB ケーブル接続中はダンス実行できません。ダンスをした後、踊ったダンスの ID を Broadcast Intent にて受け取ることが可能です。

ダンスを実行するサンプルコードを以下に記載します。

【SampleDance: MainActivity.java】

```
public class MainActivity extends Activity {
    /** ダンス実行結果通知用 Action 定義 */
    public static final String ACTION_RESULT_DANCE = "jp.co.sharp.sample.dance.action.RESULT_DANCE";
    ~中略~

    protected void onCreate(Bundle savedInstanceState) {
        ~中略~
        //ランダムボタンの実装.
        Button randomButton = (Button) findViewById(R.id.dance_normal_button);
        randomButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                sendBroadcast(getIntentForDance(DanceUtil.EXTRA_TYPE_NORMAL));
            }
        });
        ~中略~
    }
    ~中略~
    /**
     * ダンス開始用の Intent を設定する.
     */
    private Intent getIntentForDance(String type) {
        Intent intent = new Intent(DanceUtil.ACTION_REQUEST_DANCE);
        intent.putExtra(DanceUtil.EXTRA_REPLYTO_ACTION, ACTION_RESULT_DANCE);
        intent.putExtra(DanceUtil.EXTRA_REPLYTO_PKG, getPackageName());
        intent.putExtra(DanceUtil.EXTRA_TYPE, type);
        if(type.equals(DanceUtil.EXTRA_TYPE_ASSIGN)) {
            intent.putExtra(DanceUtil.EXTRA_REQUEST_ID, 1);
        }
        intent.putExtra(DanceUtil.EXTRA_SKIP_COMMENT, false);
        return intent;
    }
}
```

①

②

③

④

⑤

⑥

⑦

⑧

- ① ダンス実行結果取得用の Action を定義します。
- ② ダンス実行用 Intent を Broadcast します。

- ③ ダンス実行用 Intent のインスタンスを生成します。
- ④ ①で定義したダンス結果取得用の Action を指定します。
- ⑤ ダンス結果取得用 Intent の送信先に自 Package 名を指定します。
- ⑥ ダンス実行タイプを指定します。
- ⑦ 指定のダンスを実行する場合は ID を指定します。
- ⑧ ダンス実行後のコメントをスキップするかどうかを指定します。※ビルド番号 02.01.00 以降で利用可

実行したダンスの ID を取得する時のサンプルコードを以下に記載します。

【SampleDance: MainActivity.java】

```
public class MainActivity extends Activity {
    /** ダンス実行結果通知用 Action 定義 */
    public static final String ACTION_RESULT_DANCE = "jp.co.sharp.sample.dance.action.RESULT_DANCE";
    ~中略~

    protected void onCreate(Bundle savedInstanceState) {
        ~中略~
        //ダンス結果取得用レシーバー登録.
        mDanceResultReceiver = new DanceResultReceiver();
        IntentFilter filterDance = new IntentFilter(ACTION_RESULT_DANCE);
        registerReceiver(mDanceResultReceiver, filterDance);
    }
    ~中略~
    /**
     * ダンス実行結果を受け取るための Broadcast レシーバー クラス.<br>
     * <p/>
     */
    private class DanceResultReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            int result = intent.getIntExtra(DanceUtil.EXTRA_RESULT_CODE, DanceUtil.RESULT_CANCELED);
            if (result == DanceUtil.RESULT_OK) {
                // 正常に完了した場合.
                int id = intent.getIntExtra(DanceUtil.EXTRA_RESULT_ID, -1);
                String name = intent.getStringExtra(DanceUtil.EXTRA_RESULT_NAME);
            } else {
                // 中断/キャンセルで終了した場合.
            }
        }
    }
}
```

- ① ダンス実行結果取得用の Action を定義します。
- ② ダンス結果受信用の IntentFilter を設定します。
- ③ 実行したダンスの ID を取得します。
- ④ 実行したダンスの名前を取得します。※ビルド番号 01.07.00 以降で利用可

11. メッセージ

本章では、アプリからメッセージを送信する方法について記載します。本機能はロボホンのビルド番号 02.01.00 以降で利用できます。利用時は「設定 - 端末情報 - ビルド番号」よりビルド番号を確認してください。インターフェース仕様の詳細は、[参考資料 \[2\]](#)を参照してください。

メッセージを利用するためには AndroidManifest.xml に以下のような記述を追加する必要があります。サンプルは SDK 内に同梱されております。

【SampleMessage: AndroidManifest.xml】

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="jp.co.sharp.sample.message" >

    <uses-permission android:name="jp.co.sharp.android.rb.messaging.permission.SEND_MESSAGE" />

    <application
        ~中略~
        <uses-library android:name="jp.co.sharp.android.rb.messaging.framework" android:required="true" />
        ~中略~
    </application>
</manifest>
```

①

②

① <manifest> タグ内に <uses-permission> を定義します。

② <application> タグ内に<uses-library> を定義します。

11.1 メッセージの送信

アプリはメッセージ送信を Broadcast Intent により実行することができます。メッセージを送信するサンプルコードを以下に記載します。

【SampleMessage: MainActivity.java】

```
public class MainActivity extends Activity {
    /**
     * メッセージ送信結果通知用 Action 定義
     */
    public static final String ACTION_RESULT_MESSAGE = "jp.co.sharp.sample.message.action.RESULT_MESSAGE";
    ~中略~

    protected void onCreate(Bundle savedInstanceState) {
        ~中略~
    }
}
```

①

```
//メッセージ連携起動結果取得用レシーバー登録.
```

```
mMessageResultReceiver = new MessageResultReceiver();
IntentFilter filterMessage = new IntentFilter(ACTION_RESULT_MESSAGE);
registerReceiver(mMessageResultReceiver, filterMessage);
```

②

～中略～

```
//[送信実行]ボタンの実装.
```

```
Button sendButton = (Button) findViewById(R.id.button_send);
sendButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // 送信結果テキストを空にする.
        TextView tv_result = (TextView) findViewById(R.id.text_result);
        tv_result.setText(getResources().getString(R.string.txt_result));
        // メッセージ送信要求.
```

```
sendBroadcast(getIntentForMessage());
```

③

```
}
```

```
});
```

```
}
```

～中略～

```
/**
```

```
* メッセージ送信用の Intent を設定する.
```

```
*/
```

```
private Intent getIntentForMessage() {
    String mail_addr = (mEditTextAddr != null) ? mEditTextAddr.getText().toString() : null;
    String body = (mEditTextBody != null) ? mEditTextBody.getText().toString() : null;
    String attachment_path = mFilePath;
    boolean skip_confirm = mSkipConfirm;
    boolean background = mBackGround;
```

```
Intent intent = new Intent(MessagingUtil.ACTION_SEND_MESSAGE);
```

⑨

```
if (mail_addr != null) {
```

```
    intent.putExtra(MessagingUtil.EXTRA_EMAIL, mail_addr); }
```

④

```
intent.putExtra(MessagingUtil.EXTRA_SUBJECT, (String) null);
```

⑤

```
if (body != null) {
```

```
    intent.putExtra(MessagingUtil.EXTRA_TEXT, body); }
```

⑥

```
if (attachment_path != null) {
```

```
    intent.putExtra(MessagingUtil.EXTRA_ATTACHMENT_PATH, attachment_path); }
```

⑦

```
intent.putExtra(MessagingUtil.EXTRA_SKIP_CONFIRM, skip_confirm);
```

⑧

```
intent.putExtra(MessagingUtil.EXTRA_BACKGROUND, background);
```

⑩

```
intent.putExtra(MessagingUtil.EXTRA_REPLYTO_ACTION, ACTION_RESULT_MESSAGE);
```

⑪

```
intent.putExtra(MessagingUtil.EXTRA_REPLYTO_PKG, getPackageName());
```

⑫

```
return intent;
```

```
}
```

```

/**
 * メッセージ送信結果を受け取るための Broadcast レシーバー クラス.<br>
 * <p/>
 */
private class MessageResultReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        TextView tv_result = (TextView) findViewById(R.id.text_result);

        Int result =
            intent.getIntExtra(MessagingUtil.EXTRA_RESULT_CODE, MessagingUtil.RESULT_CANCELED);
        if (result == MessagingUtil.RESULT_OK) {
            // 正常に完了した場合.
            Log.d(TAG, "SEND_MESSAGE: Success!!");
            tv_result.setText(getResources().getString(R.string.txt_result) + "成功");
        } else {
            // 中断/キャンセルで終了した場合.
            Log.d(TAG, "SEND_MESSAGE: Canceled...");
            tv_result.setText(getResources().getString(R.string.txt_result) + "失敗/キャンセル");
        }
    }
}

```

⑬

- ① メッセージ送信結果取得用の Action を定義します。
- ② メッセージ送信結果受信用の Receiver を設定します。
- ③ メッセージ送信用 Intent を Broadcast します。
- ④ メッセージ送信用の Action を指定し Intent のインスタンスを生成します。
- ⑤ 送信先のメールアドレスを指定します。
- ⑥ メールサブジェクトを指定します。
- ⑦ メール本文を指定します。
- ⑧ 添付ファイルのパスを指定します。
- ⑨ メール送信時の送信確認の可否を指定します。
- ⑩ バックグラウンド送信の可否を指定します。
- ⑪ ①で定義した送信結果取得用の Action を指定します。
- ⑫ 送信結果の送信先のパッケージ名を指定します。
- ⑬ 送信結果を取得します。

12. アクション

本章では、アプリからアクションを実行する方法について記載します。本機能はロボホンのビルド番号 02.06.00 以降で利用できます。利用時は「設定 - 端末情報 - ビルド番号」よりビルド番号を確認してください。インターフェース仕様の詳細は、[参考資料 \[2\]](#)を参照してください。

本 API が提供する機能は以下の通りです。

- ・ 指定のアクションを実行する
- ・ アクションの一覧を取得する

アクションを利用するためには AndroidManifest.xml に以下のような記述を追加する必要があります。サンプルは SDK 内に同梱されております。

【SampleAction: AndroidManifest.xml】

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="jp.co.sharp.sample.action" >

    <uses-permission android:name="jp.co.sharp.android.rb.action.permission.REQUEST_ACTION" />

    <application
        ~中略~
        <uses-library android:name="jp.co.sharp.android.rb.action.framework" android:required="true" />
        ~中略~
    </application>
</manifest>
```

①

②

- ③ <manifest> タグ内に <uses-permission> を定義します。
- ④ <application> タグ内に<uses-library> を定義します。

12.1 アクションの実行

アプリはアクションを Broadcast Intent により実行することができます。ただし、USB ケーブル接続中はアクションを実行できません。アクションをした後、実行したアクションの ID と名前を Broadcast Intent にて受け取ることが可能です。アクションを実行するサンプルコードを以下に記載します。

【SampleAction: MainActivity.java】

```
public class MainActivity extends Activity {
    /** アクション実行結果通知用 Action 定義 */
    public static final String ACTION_RESULT_ACTION = "jp.co.sharp.sample.action.action.RESULT_ACTION";
    ~中略~

    protected void onCreate(Bundle savedInstanceState) {
        ~中略~
        //指定ボタンの実装.
        Button assignButton = (Button) findViewById(R.id.action_assign_button);
        assignButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                sendBroadcast(getIntentForAction(numPicker.getValue()));
                dispId("");
                dispName("");
            }
        }); ~中略~
    }
    ~中略~
    /**
     * アクション開始用の Intent を設定する.
     */
    private Intent getIntentForAction(int id) {
        Intent intent = new Intent(ActionUtil.ACTION_REQUEST_ACTION);
        intent.putExtra(ActionUtil.EXTRA_REPLYTO_ACTION, ACTION_RESULT_ACTION);
        intent.putExtra(ActionUtil.EXTRA_REPLYTO_PKG, getPackageName());
        intent.putExtra(ActionUtil.EXTRA_REQUEST_ID, id);
        return intent;
    }
}
```

- ① アクション実行結果取得用の Action を定義します。
- ② アクション実行用 Intent を Broadcast します。
- ③ アクション実行用 Intent のインスタンスを生成します。
- ④ ①で定義したアクション結果取得用の Action を指定します。

- ⑤ アクション結果取得用 Intent の送信先に自 Package 名を指定します。
- ⑥ 実行するアクションの ID を指定します。

実行したアクションの ID を取得する時のサンプルコードを以下に記載します。

【SampleAction: MainActivity.java】

```
public class MainActivity extends Activity {
    /** アクション実行結果通知用 Action 定義 */
    public static final String ACTION_RESULT_ACTION = "jp.co.sharp.sample.action.action.RESULT_ACTION";
    ~中略~

    protected void onCreate(Bundle savedInstanceState) {
        ~中略~
        //アクション結果取得用レシーバー登録.
        mActionResultReceiver = new ActionResultReceiver();
        IntentFilter filterAction = new IntentFilter(ACTION_RESULT_ACTION);
        registerReceiver(mActionResultReceiver, filterAction);
    }
    ~中略~
    /**
     * アクション実行結果を受け取るための Broadcast レシーバー クラス.<br>
     * <p/>
     */
    private class ActionResultReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            int result = intent.getIntExtra(ActionUtil.EXTRA_RESULT_CODE, ActionUtil.RESULT_CANCELED);
            if (result == ActionUtil.RESULT_OK) {
                // 正常に完了した場合.
                int id = intent.getIntExtra(ActionUtil.EXTRA_RESULT_ID, -1);
                String name = intent.getStringExtra(ActionUtil.EXTRA_RESULT_NAME);
            } else {
                // 中断/キャンセルで終了した場合.
            }
        }
    }
}
```

- ① アクション実行結果取得用の Action を定義します。
- ② アクション結果受信用の IntentFilter を設定します。
- ③ 実行したアクションの ID を取得します。
- ④ 実行したアクションの名前を取得します。

13. 歌

本章では、アプリから歌を実行する方法について記載します。本機能はロボホンのビルド番号 02.05.00 以降で利用できます。利用時は「設定 - 端末情報 - ビルド番号」よりビルド番号を確認してください。インターフェース仕様の詳細は、[参考資料 \[2\]](#)を参照してください。

本 API が提供する機能は以下の通りです。

- ・ ランダムに選択された歌を実行する
- ・ 前回と同じ歌を実行する
- ・ 指定の歌を実行する
- ・ 歌の一覧を取得する

歌を利用するためには AndroidManifest.xml に以下のような記述を追加する必要があります。サンプルは SDK 内に同梱されています。

【SampleSong: AndroidManifest.xml】

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="jp.co.sharp.sample.song" >

    <application
        ~中略~
        <uses-library android:name="jp.co.sharp.android.rb.song.framework" android:required="true" />
        ~中略~
    </application>
</manifest>
```

①

① <application> タグ内に<uses-library> を定義します。

13.1 歌の実行

アプリは歌を Broadcast Intent により実行することができます。歌った後、実行した歌の ID と名前を Broadcast Intent にて受け取ることが可能です。

歌を実行するサンプルコードを以下に記載します。

【SampleSong: MainActivity.java】

```
public class MainActivity extends Activity {
    /** アクション実行結果通知用 Action 定義 */
    public static final String ACTION_RESULT_SONG = "jp.co.sharp.sample.song.action.RESULT_SONG";
    ~中略~

    protected void onCreate(Bundle savedInstanceState) {
        ~中略~
        //指定ボタンの実装.
        Button assignButton = (Button) findViewById(R.id.song_assign_button);
        assignButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                sendBroadcast(getIntentForSong(SongUtil.EXTRA_TYPE_ASSIGN));
                dispId("");
                dispName("");
            }
        }); ~中略~
    }
    ~中略~
    /**
     * 歌開始用の Intent を設定する.
     */
    private Intent getIntentForSong(String type) {
        Intent intent = new Intent(SongUtil.ACTION_REQUEST_SONG);
        intent.putExtra(SongUtil.EXTRA_REPLYTO_ACTION, ACTION_RESULT_SONG);
        intent.putExtra(SongUtil.EXTRA_REPLYTO_PKG, getPackageName());
        intent.putExtra(SongUtil.EXTRA_TYPE, type);
        if(type.equals(SongUtil.EXTRA_TYPE_ASSIGN)) {
            intent.putExtra(SongUtil.EXTRA_REQUEST_ID, numPicker.getValue());
        }
        return intent;
    }
}
```

① 歌実行結果取得用の Action を定義します。

- ② 歌実行用 Intent を Broadcast します。
- ③ 歌実行用 Intent のインスタンスを生成します。
- ④ ①で定義した歌結果取得用の Action を指定します。
- ⑤ 歌結果取得用 Intent の送信先に自 Package 名を指定します。
- ⑥ 歌実行タイプを指定します。
- ⑦ 実行する歌の ID を指定します。

実行した歌の ID と名前を取得する時のサンプルコードを以下に記載します。

【SampleSong: MainActivity.java】

```
public class MainActivity extends Activity {
    /** 歌実行結果通知用 Action 定義 */
    public static final String ACTION_RESULT_SONG = "jp.co.sharp.sample.song.action.RESULT_SONG";
    ~中略~

    protected void onCreate(Bundle savedInstanceState) {
        ~中略~
        //歌実行結果取得用レシーバー登録.
        mSongResultReceiver = new SongResultReceiver();
        IntentFilter filterSong = new IntentFilter(ACTION_RESULT_SONG);
        registerReceiver(mSongResultReceiver, filterSong);
    }
    ~中略~
    /**
     * 歌実行結果を受け取るための Broadcast レシーバー クラス.<br>
     * <p/>
     */
    private class SongResultReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            int result = intent.getIntExtra(SongUtil.EXTRA_RESULT_CODE, SongUtil.RESULT_CANCELED);
            if (result == SongUtil.RESULT_OK) {
                // 正常に完了した場合.
                int id = intent.getIntExtra(SongUtil.EXTRA_RESULT_ID, -1);
                String name = intent.getStringExtra(SongUtil.EXTRA_RESULT_NAME);
            } else {
                // 中断/キャンセルで終了した場合.
            }
        }
    }
}
```

①

②

③

④

- ① 歌実行結果取得用の Action を定義します。
- ② 歌実行結果受信用の IntentFilter を設定します。
- ③ 実行した歌の ID を取得します。
- ④ 実行した歌の名前を取得します。