SHARP

# ロボホン認定試験チュートリアル
# 4.アプリ開発（応用編）

Version 1.0.1

Last update 2016/09/28

SHARP CORPORATION

SHARP

## 更新履歴

| Version | Description | Date |
|---|---|---|
| 1.0.0 | 初版作成 | 2016/09/08 |
| 1.0.1 | 重要項目を追加 | 2016/09/28 |

SHARP

# 目次

# 1. 音声を使ったアプリケーションの作成

## 1.1 音声を使ったアプリの作成

ロボホンは音声 UI を備えており、音声を使ってユーザーとコミュニケーションを図ることができます。
開発者が作成するアプリにおいても、ユーザーからの入力や、ロボホンからのリアクションとして音声を使用するのが最適です。
ユーザーとロボホンの会話を簡単に作成するために、ロボホンアプリ開発では HVML と呼ばれる、シャープ独自のマークアップ言語にシナリオを書き込み、HVML ファイルを Java ファイルから読み出して実行します。
この章ではロボホンに特定の言葉を喋らせるアプリを作成し、ロボホン実機で実行するまでの流れを解説します。
HVML ファイルの詳しい仕様や、詳細情報は第 5 章にて解説します。

## 1.2 新しいプロジェクトの作成

それでは、新しいプロジェクトを作りながら、会話用シナリオの作成方法を確認してみましょう。
認定試験チュートリアル資料 3 章の「ロボホン用テンプレートファイルを利用して、アプリ開発を始める」を参考に新しいプロジェクトを作成してください。

**新しいプロジェクトで設定する値**

| 設定項目 | 設定する値 |
|---|---|
| Application Name | MyFirstScinario |

※上記以外の項目は任意の値をご指定ください。

## 1.3 レイアウトファイルの確認

Android Studio の左側のツリービューより、[res > layout >activity_main.xml] を表示して、レイアウトファイルを確認します。API レベルの設定を 21 に変更することで、背面ディスプレイの画面レイアウトを GUI で編集できるようになります。

**レイアウトファイルの確認**

| 番号 | 設定項目 |
|---|---|
| 1 | 左側のツリービューを[Android]モードに変更する |
| 2 | ツリービューより[res > layout >activity_main.xml]を選択して表示する |
| 3 | シュミレーター選択のプルダウンメニューより、[Galaxy Nexus]を選択する |
| 4 | ドロイド君マークのアイコンがある API レベルの設定を[21]に変更する |

編集するファイル: [res > layout > activity_main.xml]

※詳しくは認定試験チュートリアル資料３章「ロボホン用テンプレートファイルを利用して、アプリ開発を始める」をご確認ください。

## 1.4　ホーム用シナリオの編集

ロボホン用アプリの起動方法は背中の画面に表示されたアイコンをアップする他に、ロボホンが待機状態の時に直接話しかける事でも起動することが出来ます。
ロボホンに話しかけてアプリを起動するためには、ホーム用シナリオと呼ばれる、アプリ起動専用の特別なシナリオファイル(home.hvml)を用意する必要があります。
ホーム用シナリオの作成は必須です。また、アプリ内に１つしか作成することができません。

**ホーム用シナリオファイル(home.hvml)の命名規則**

[Android]ツリービュー [assets > home]内に

「パッケージ名(.を_に変更)＋_(アンダーバー)＋home.hvml」

ロボホン用テンプレートからアプリを作成すると [assets > home > パッケージ名+_home.hvml] に自動的に雛形ファイルが作成されます。

それでは、home.hvml を編集します。
HVML ファイルの<head />内に<situation />タグが２つ記載されている箇所を編集します。
<situation />タグ内の「てんぷれーと」「てんぷれーとあぷりしよう」と記載されている部分にアプリを起動するための言葉を入力します。今回はそれぞれ「おしゃべり」「ひまだな」と変更します。

変更前のコード

```
<situation priority="78" topic_id="start" trigger="user-word">
    ${Local:WORD_APPLICATION} eq てんぷれーと
</situation>
<situation priority="78" topic_id="start" trigger="user-word">
    ${Local:WORD_APPLICATION_FREEWORD} eq てんぷれーとあぷりしよう
</situation>
```

変更後のコード

```
<situation priority="78" topic_id="start" trigger="user-word">
    ${Local:WORD_APPLICATION} eq おしゃべり
</situation>
<situation priority="78" topic_id="start" trigger="user-word">
    ${Local:WORD_APPLICATION_FREEWORD} eq ひまだな
</situation>
```

編集するファイル:　[assets> home > パッケージ名+_home.hvml]

続いて、<body />タグ内の<action index="2" />内に記載されている、<speech />タグの内容を変更します。先ほど設定した「ひまだな」と話しかけられ、実際にアプリを起動する時に喋らせる項目です。

変更前のコード

```
<speech>テンプレートを起動するね</speech>
```

変更後のコード

```
<speech>じゃあ一緒にお話しよう</speech>
```

編集するファイル: [assets> home > パッケージ名+_home.hvml]

最後に、hvml の version を増やします。hvml ファイルは、アプリインストール時にロボホン本体に登録されます。その際、すでに本体に登録されている hvml よりも、version 番号が大きくない場合は、hvml が登録されません。
本チュートリアルでは、この説明以降、version を増やす説明は省略します、hvml ファイルを変更した場合、必ず version を増やして保存するよう気をつけてください。

HVML バージョンの変更例

```
<?xml version="1.0" ?>
<hvml version="2.0">
    <head>
        <producer>jp.co.sharp.robohon.example.myfirstscinario</producer>
        <!-- TODO  このシナリオの説明文を入力してください(プログラムに影響はありません) -->
        <description>テンプレートのホーム起動シナリオ</description>
        <scene value="home" />
        <version value="2.0" />
        <situation priority="78" topic_id="start" trigger="user-word">${Local:WORD_APPLICATION} eq
            てんぷれーと
        </situation>
```

編集するファイル: [assets> home > パッケージ名+_home.hvml]

なお、以下の version は、xml 構文の仕様を定義するバージョン、hvml 自体の使用を定義するバージョンですので、変更しないように注意してください。

| 変更禁止のバージョン |
| --- |

```
<?xml version="1.0" ?>
<hvml version="2.0">
    <head>
        <producer>  jp.co.sharp.robohon.example.myfirstscinario</producer>
        <!-- TODO  このシナリオの説明文を入力してください(プログラムに影響はありません) -->
        <description>テンプレートのホーム起動シナリオ</description>
        <scene value="home" />
        <version value="2.0" />
        <situation priority="78" topic_id="start" trigger="user-word">${Local:WORD_APPLICATION} eq
            てんぷれーと
        </situation>
```

この状態でアプリをロボホンに転送して、アプリが起動した後、頭のボタンを押して一度停止させ、待機画面に戻った状態で、ロボホンに「ひまだな」と話しかけると、アプリを起動することが出来ます。

## 1.5　アプリが起動したら、自動的にシナリオファイルを実行する

ロボホン用のテンプレートで作成される雛形アプリは、ロボホンの背中の画面に発話ボタンが表示され、発話ボタンをクリックするとシナリオファイルが実行される仕組みとなっています。
今回は、雛形アプリを編集して、アプリが起動したら自動的にシナリオファイルが実行されるように変更します。

Android Studio 左側のツリービューより[java > プロジェクト名 > MainActivity]を開きます。
MainActivity で定義されている onCreate メソッド内の setOnClickListener イベントのコードをコピーします。
コピーしたコードを、onResume メソッド内の一番最後に貼り付けます。

**コピーするコード**

```
if (mVoiceUIManager != null) {
            VoiceUIVariableListHelper helper = new
VoiceUIVariableListHelper().addAccost(ScenarioDefinitions.ACC_HELLO);
            VoiceUIManagerUtil.updateAppInfo(mVoiceUIManager, helper.getVariableList(), true);
}
```

**コピーしたコードを貼り付けた例**

```
public void onResume() {
    super.onResume();
    Log.v(TAG, "onResume()");
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 一部省略 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    //Scene 有効化.
    VoiceUIManagerUtil.enableScene(mVoiceUIManager, ScenarioDefinitions.SCENE_COMMON);
    VoiceUIManagerUtil.enableScene(mVoiceUIManager, ScenarioDefinitions.SCENE01);

    if (mVoiceUIManager != null) {
        VoiceUIVariableListHelper helper = new VoiceUIVariableListHelper().addAccost(ScenarioDefinitions.ACC_HELLO);
        VoiceUIManagerUtil.updateAppInfo(mVoiceUIManager, helper.getVariableList(), true);
    }
}
```

編集するファイル: ［java ＞ パッケージ名 ＞ MainActivity］

onResume 内にコードを貼り付けたら、onCreate メソッド内の、setOnClickListener イベントを削除します。

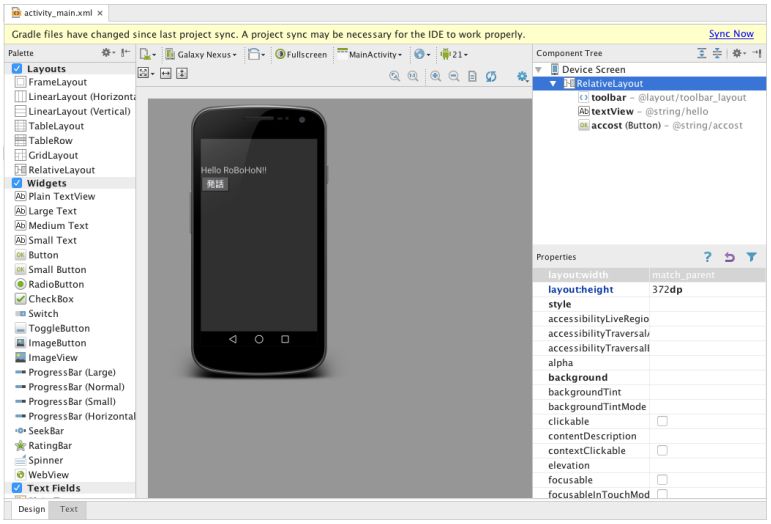
**削除するコード**

```
//発話ボタンの実装.
Button Button = (Button) findViewById(R.id.accost);
Button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (mVoiceUIManager != null) {
            VoiceUIVariableListHelper helper = new VoiceUIVariableListHelper().addAccost(ScenarioDefinitions.ACC_HELLO);
            VoiceUIManagerUtil.updateAppInfo(mVoiceUIManager, helper.getVariableList(), true);
        }
    }
});
```

編集するファイル: ［java ＞ パッケージ名 ＞ MainActivity］

setOnClickListener イベントを削除した後、イベントと紐付いているボタンパーツを削除します。

Android Studio 左側のツリービューより[res > layout > activity_main.xml]を表示します。
Activity_main.xml の内容が表示されている左下に[Design]と[Text]の切り替えタブが表示されているので、[Design]に切り替えます。[Design]に切り替えると、ロボホンの背中の画面に表示させるパーツが GUI 形式で表示されます。




(activity_main.xmlの編集画面)

画面の中央に表示されているバーチャル端末の画面内に表示されている、発話ボタン選択し、削除します。
(発話ボタンを選択した状態で、backspace キー、delete キー等をタイプすると削除できます。)

削除が完了すると、xml コードから、発話ボタンのタグが削除されます。
左下の[Design]と[Text]の切り替えタブを[Text]に切り替えて、発話タグが削除されている事を確認してみてください。

発話ボタンを削除した activity_main.xml の例

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="372dp"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="jp.co.sharp.robohon.example.myfirstscinario.MainActivity">

    <include
        android:id="@+id/toolbar"
        layout="@layout/toolbar_layout" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_below="@+id/toolbar"
        android:text="@string/hello"
        android:textSize="27sp" />

</RelativeLayout>
```

編集するファイル: [res > layout > activity_main.xml]

この状態でアプリを再生してみると、起動してすぐにロボホンが喋り出し、ロボホン用テンプレートの雛形ファイル hello.hvml で定義されているシナリオが実行されます。

## 1.6　発話用シナリオファイルを作成する

続いて、ロボホンに喋らせるための「台本」となるシナリオファイル(hvml ファイル)を作成します。
Android Studio のロボホン用テンプレートには予め、hello.hvml という hvml ファイルが雛形として生成されているので、このファイルを編集して、シナリオを作成していきます。
シナリオファイルには、ロボホンが聞き取る人間の言葉と、ロボホンに喋らせる言葉の両方を書き込むことが出来ますが、この章ではロボホンに指定したセリフを喋らせるサンプルアプリを作成してみます。
(人間の言葉を聞き取る方法は第５章「HVML ファイル」にて詳しく解説致します。)

Android Studio 左側のツリービューより[assets > other > パッケージ名+_hello.hvml]を表示します。
hello.hvml の head 内に記載されている、<situation />タグを削除します。

削除するコード

```
<situation priority="75" topic_id="reply" trigger="user-word">${Lvcsr:Basic} include
    [おはよう,こんにちは,こんばんは]
</situation>
```

編集するファイル:　[assets > other > パッケージ名+_hello.hvml]

続いて、<body />内のコードも全て削除します。

<body />内を空にした状態の hello.hvml 例

```
<?xml version="1.0" ?>
<hvml version="2.0">
    <head>
        <producer>  jp.co.sharp.robohon.example.myfirstscinario</producer>
        <!-- TODO  このシナリオの説明文を入力してください(プログラムに影響はありません) -->
        <description>テンプレートシナリオ</description>
        <scene value="jp.co.sharp.robohon.example.myfirstscinario.scene01" />
        <version value="1.0" />
        <accost priority="75" topic_id="say"
            word="jp.co.sharp.robohon.example.myfirstscinario.hello.say" />
    </head>
    <body>
    </body>
</hvml>
```

編集するファイル:　[assets > other > パッケージ名+_hello.hvml]

<body />内を空にしたら、今回の対話内容を<topic />タグを使用して定義します。

以下のコードを<body />タグ内に記述していきます。

追加するコード

```
<topic id="say" listen="false">
    <action index="1">
        <speech>大好きだよ。ずっと一緒にいようね。</speech>
        <behavior id="assign" type="normal" />
    </action>
</topic>
```

編集するファイル:　[assets > other > パッケージ名+_hello.hvml]

この状態で、アプリを実行してみると、ロボホンが<speech />タグ内に記載された言葉を喋ります。

| タグの解説 | **Topic タグ** |
|---|---|
| Topic タグは会話や音声聞き取りなど、HVML 内で実行したい項目をブロック化するためのタグです。 | |
| **<topic id="ID 名" listen="true または false"></topic>** | |
| 属性名 | 値 |
| **id** | ID 属性には Topic タグの ID 名を入力します。 |
| **listen** | Listen 要素は、Topic タグ内で音声認識を行うかのフラグです。True に設定すると Topic 内で音声認識を行うことが出来ます。 |
| **その他** | このタグには更に属性が存在します。詳しくは SDK 資料[HVML 2.0 Specification]をご確認ください。 |

| タグの解説 | **Speech タグ** |
|---|---|
| Speech タグはタグで囲まれた中の言葉をロボホンが喋るタグです。<topic />タグ内に配置された、<action />直下に配置します。 | |
| **<speech>大好きだよ。ずっと一緒にいようね。</speech>** | |
| 属性名 | 値 |
| **なし** | |

# 2. プロジェクターを使ったアプリケーションの作成

## 2.1 プロジェクターを使ったアプリの作成

ロボホンの頭部にはプロジェクターが搭載されています。
このプロジェクターを使って、写真や動画、地図など様々なコンテンツをテーブル等に映し出すことが出来ます。
プロジェクターは開発者が作成するロボホンアプリにおいても、とても簡単に組み込むことができ、プロジェクターを活用することで、ロボホンの表現できる幅を広げる事ができます。
この章ではプロジェクターを使ったアプリを実際に作成しながら、組み込み方を確認していきます。

## 2.2 新しいプロジェクトの作成

それでは、新しいプロジェクトを作りながら、アプリへのプロジェクターの組み込み方法を確認してみましょう。
本資料３章の「ロボホン用テンプレートファイルを利用して、アプリ開発を始める」を参考に新しいプロジェクトを作成してください。

**新しいプロジェクトで設定する値**

| 設定項目 | 設定する値 |
|---|---|
| Application Name | MyFirstProjector |

※上記以外の項目は任意の値をご指定ください。

## 2.3 レイアウトファイルの設定

以前の項目の通り、レイアウトファイルを設定します。

**レイアウトファイルの設定**

| 番号 | 設定項目 |
|---|---|
| 1 | 左側のツリービューを[Android]モードに変更する |
| 2 | ツリービューより[res > layout >activity_main.xml]を選択して表示する |
| 3 | シュミレーター選択のプルダウンメニューより、[Galaxy Nexus]を選択する |
| 4 | ドロイド君マークのアイコンがある API レベルの設定を[21]に変更する |

※詳しくは認定試験チュートリアル資料３章「ロボホン用テンプレートファイルを利用して、アプリ開発を始める」をご確認ください。

## 2.4 アプリの雛形をカスタマイズする

プロジェクターの呼び出し等をコーディングする前に、アプリが起動したら、自動的にシナリオファイルが呼び出されるように変更しましょう。
前項目の「音声を使ったアプリケーションの作成」を参考にしながら、テンプレートから出力された雛形ファイルを編集します。

| テンプレートの雛形ファイルを修正項目 | |
|---|---|
| 番号 | 項目 |
| 1 | ホーム用のシナリオファイルを編集し、音声で起動する際の言葉を編集する<br>(例:ぷろじぇくたーあぷり、ぷろじぇくたーみたいな) |
| 2 | MainActivity 内の onCreate メソッドに記載されたボタンクリックイベントの内部コードを、onResume の最後にペーストして、onCreate メソッドのボタンクリックイベントを削除する |
| 3 | Activity_main.xml を開き、[Design]編集モードに切り替えて発話ボタンを削除する |
| 4 | hello.hvml のシナリオファイルを編集し、<topic id="say" />だけを残して、それ以外の Topic タグを削除する |

上記の編集を行い、一度アプリの転送を行って、アプリの起動と同時に、hello.hvml ファイルが実行されるか確認してみてください。

## 2.5 プロジェクターを呼び出すコードの追加

今回のアプリでは、会話用の HVML ファイルを自動実行して、<topic />タグの内容(発話)を実行した後に、プロジェクターを呼び出す処理を実行してみたいと思います。

ロボホン用のテンプレートから生成された MainActivity のコードには、予めプロジェクターを呼び出すための処理がコメントアウトされて記述されているので、コメントアウトを解除し、HVML ファイルにプロジェクターを呼び出すコードを記述するだけで作成できます。

**MainActivity の編集**

Android Studio の左側のツリービューを[Android]モードに変更し、[java > パッケージ名 > MainActivity]を表示します。

既にプロジェクターを使用するためのコードが、MainActivity 内に記載されていますので、[//**TODO** プロジェクター・・・]とコメントが記載されている箇所の直後の行のコメントアウトを解除します。

| コメントアウトを解除する項目 | |
|---|---|
| 行数 | 項目 |
| 78 | //***TODO*** *プロジェクターイベントの検知登録(プロジェクター利用時のみ).*<br>setProjectorEventReceiver(); |
| 135 | //***TODO*** *プロジェクターイベントの検知破棄(プロジェクター利用時のみ).*<br>**this**.unregisterReceiver(**mProjectorEventReceiver**); |
| 159 | //***TODO*** *プロジェクターマネージャの開始(プロジェクター利用時のみ).*<br>startService(getIntentForProjector()); |

MainActivity の作業はこれで完了です。

続いて、hello.hvml のシナリオファイルを編集します。

**hello.hvml の編集**

Android Studio の左側のツリービューを[Android]モードに変更し、[assets > other > パッケージ名+_hello.hvml]を表示します。

予め雛形ファイルに存在する<topic id=”say” />タグを以下のとおりに編集します。

編集したコード

```
<topic id="say" listen="false">
    <action index="1">
        <speech>はーい！プロジェクターだね。</speech>
        <behavior id="assign" type="normal" />
    </action>
    <next href="#projector" type="default"/>
</topic>
```

編集するファイル: [assets > other > パッケージ名+_hello.hvml]

つづいて、<topic id=”say” />の下に、<next />タグで指定した<topic id=”projector” />タグを追加します。

追加するコード

```
<topic id="projector" listen="false">
    <action index="1">
        <control function="Projector_Control"
target="jp.co.sharp.robohon.android.myFirstProjector"></control>
    </action>
</topic>
```

編集するファイル: [assets > other > パッケージ名+_hello.hvml]

上記のコードの中で使用している<control />タグは、HVML から Java 言語で記述された Activity のメソッドを呼び出す為のタグです。

| タグの解説 | **Control タグ** |
|---|---|
| Control タグは HVML ファイルから、Java 言語で記述された Activity 内のメソッドなどを呼び出す際に使用します。 | |
| **<control function="変数名" target="パッケージ名"></control>** | |
| 属性名 | 値 |
| **function** | 呼び出しを行いたい Function の名前を指定します。 |
| **target** | 今回のアプリのパッケージ名を入力します。 |

**Activity_main.xml(レイアウトファイル)の編集**

最後に、プロジェクターが映し出す画像を設定します。

今回のアプリは背中の画面をそのままプロジェクターで投影しているので、背中の画面のレイアウトファイルを変更することで、プロジェクターに反映することが出来ます。

今回は任意の画像ファイルを表示してみましょう。初めに、表示したい PNG 形式の画像ファイルを準備してください。

Android Studio の左側のツリービューを[Android]モードに変更し、[res > layout > activity_main.xml]を表示し、表示方式を[Text]モードに指定します。

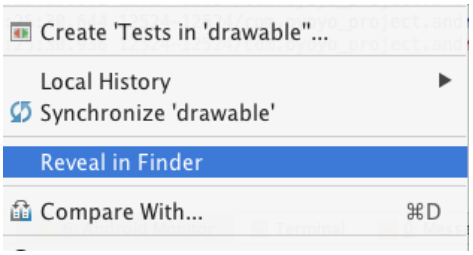
Activity_main.xml 内に ImageView のパーツを追加します。

追加するコード

```
<ImageView
    android:src="@drawable/example"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/imageView"/>
```

編集するファイル: [res > layout > activity_main.xml]

最後に、表示させたい画像をプロジェクトに追加します。
Android Studio の左側のツリービューを[Android]モードに変更し、[res > drawable] ディレクトリを表示します。
drawable を右クリックし、[Reveal in Finder] ( [Reveal in Explorer] ) をクリックします。




(drawable を右クリックした時)

ディレクトリが表示されたら、プロジェクターで投影したい png 画像ファイルを[drawable]ディレクトリ内にコピーし、ファイル名を[example.png]に変更します。

この状態で、アプリを実行すると、ロボホンがプロジェクターを写してくれます。

| プロジェクターを利用する上での注意点 |
| --- |
| プロジェクターはロボホンにオーナー登録されたユーザーしか使用することが出来ません。<br>プロジェクターを初めて使用する際は、ロボホンのシステムが自動的に実施するプロジェクター起動の練習が必要です。<br>USB ケーブルを接続した状態だと、プロジェクターを起動することが出来ません。 |

# 3. 電話帳を使ったアプリケーション

## 3.1 電話帳 API

電話帳 API はロボホンの電話帳アプリに登録された情報、オーナーの個人情報、ロボ情報(ロボホンの名前等)を取得することが出来ます。

今回は簡単に概要と基本的な使用方法をご説明致します。

電話帳 API についての詳細は SDK 資料[0401_SR01MW_Application_Programming_Guide]を御覧ください。

| 電話帳 API で出来ること | |
|---|---|
| できること | 連携／データ取得方法 |
| オーナー情報の取得 | API |
| ロボ情報の取得 | API |
| 電話帳に登録された情報の取得 | API |
| 電話帳情報の新規登録・削除の完了通知 | Broadcast |
| 電話帳・オーナー・ロボ情報の更新通知 | Broadcast |
| 電話帳新規登録・追加登録・検索・削除画面の表示 | アプリ連携起動 |
| あなたについて画面の表示 | アプリ連携起動 |
| ロボ情報画面の表示 | アプリ連携起動 |

**電話帳 API の使用方法**

電話帳 API を利用するには、[manifests > AndroidManifest.xml] 内に、電話帳へのアクセスを許可する user-permission タグを追加します。

追加するコード

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

編集するファイル: [manifests > AndroidManifest.xml]

続いて、MainActivity にて、AddressBookManager の、ライブラリファイルの読み込みを行います。

追加するコード

```
import jp.co.sharp.android.rb.addressbook.AddressBookCommonUtils;
import jp.co.sharp.android.rb.addressbook.AddressBookManager;
import jp.co.sharp.android.rb.addressbook.AddressBookVariable.OwnerProfileData;
```

編集するファイル: [java > パッケージ名 > MainActivity]

続いて AddressBookManager を呼び出して、使用したい項目を取得します。

コードのサンプル

```
AddressBookManager addressMng = AddressBookManager.getService(_context);

OwnerProfileData ownerData = addressMng.getOwnerProfileData();
String nickName = ownerData.getNickname();
```

編集するファイル: [java > パッケージ名 > MainActivity]