HOVIS Genie/Appで学ぶ

Android Robot Programming

基本から楽しく学ぶ
ロボットプログラミングの原理



Dongbu Robot
アーカイブ

(マニュアルとアプリケーション
をダウンロードしてください。)

www.dongburobot.com

Dongbu Robot

Contents

01 アンドロイドロボットHovis Genieの紹介

1.1 Hovis 製作の背景

1.2 Hovis Genieの特徴

(1) Hovis Lite
(2) Hovis Genie

1.3 Hovis Genieのハードウェア構成

(1) ロボット本体
(2) 顔
(3) ホビスジニの仕様

1.4 Hovis Genieのソフトウェアの特徴

(1) MPSU (DRC-005T), OPSU (DRC-004TO), MID Side Board – ファームウェア
(2) MID – Android

1.5 Hovis Genieのアプリケーションを実行する

(1) メイン画面
(2) 自律行動
(3) 遠隔制御
(4) 朝の起床情報/童謡童話などの情報コンテンツのロボットコンテンツ化
(5) ソフトウェア提供の目的と活用

Android Robot Program-ming

02 アンドロイドとロボットの開発環境を準備する

2.1 JDK インストール

2.2 アンドロイドSDKインストール

2.3 Eclipseのインストールと環境設定

03 アンドロイドロボット(ジニ)プログラミング

3.1 HelloGenie プロジェクトの作成

3.2 HelloGenie プロジェクトの実行

3.3 HelloGenie プロジェクトの基本的な構成

(1) AndroidManifest.xml

(2) MainActivity.javaとactivity_main.xml

04 Hovis Genie ロボットのプログラミングを開始する

4.1 GenieApiDemoプロジェクトの作成

4.2 GenieApiDemoApplicationクラス

(1) ロボットサービスの接続

(2) ロボットサービスの接続の解除

4.3 ロボットシステム用BaseActivityの作成

05 HOVIS Genie API (Application Programing Interface)

5.1 駆動部関数

5.2 ナビゲーション関数

5.3 センサの関数

5.4 TTS(Text-to-Speech) 関数

5.5 サウンド(効果音)関数

5.6 マルチメディア(オーディオとビデオ)関数

5.7 モーションとサーボモーターに関連する関数

5.8 頭のLED制御に関する関数

06 Hovis Genie 基本的な例

6.1 メイン画面の構成

6.2 駆動部の制御

6.3 モーション制御

6.4 頭のLED制御

6.5 TTS制御

6.6 PSDセンサ制御

6.7 タッチセンサ制御

01 アンドロイドロボットHOVIS Genieの紹介

1.1 HOVIS 製作の背景

既存の開発されているサービスロボットは案内所などで見るようにサイズが大きく、PC用のタッチスクリーンが搭載されていてユーザーが立って作業をする試験的に作ったロボットがほぼ全てでした。 End User、つまり最終消費者が直接購入するのではなく機関や政府の事業として行われる場合がほとんどでした。

このような形のロボットは、サービスの内容が非常に限定され、供給者、つまり開発会社から直接プログラミングすることができない構造を持っています。これらの制限を克服し、ロボットサービスを拡張するためにHovisというアンドロイドロボットを製作しました。

HOVISはHome Serviceの略で、ホームサービスに実際に使用できるロボットを目的して製作しました。実際の家の中で使用するためには、家の中で発生する可能性のあるすべてのタイプのサービスを含むソフトウェアが必要です。

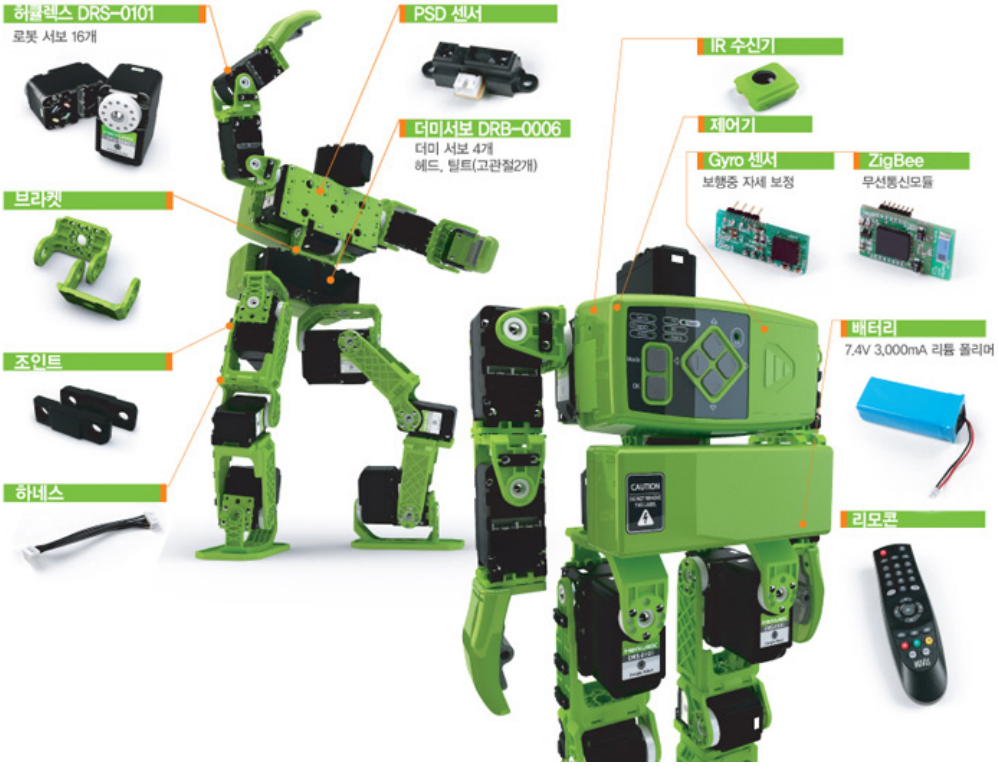
そのようなサービスを作成するために、現在のスマートフォンのソフトウェア供給方式を取り入れ、ロボットにもそのまま適用したのがHovis Genieです。現在のAndroidプラットフォームを利用して多数のエンジニアによって必要なサービスを供給できる構造です。

(株)Dongbu Robotはアンドロイドの端末機が搭載されたアンドロイドロボットを供給し、そのロボットからアンドロイドロボットサービスを作成するためのソフトウェア、つまりAPIとプログラミング方法を提供しています。

(1) Hovis Lite

Hovis Genieの説明の前にHovis Liteの簡単な説明が必要です。

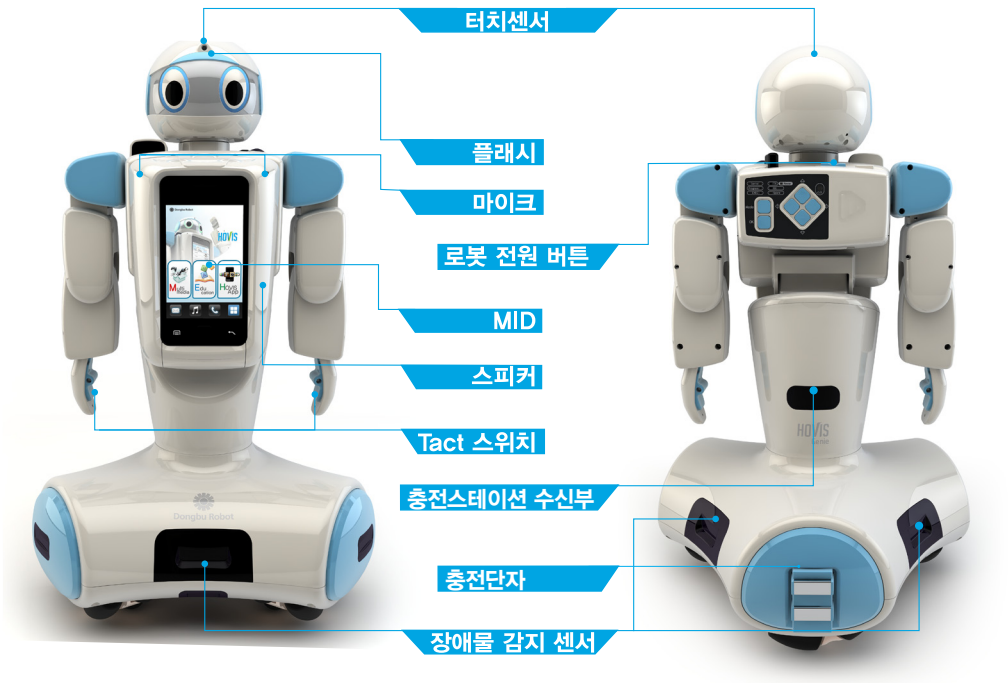
1.2 HOVIS Genieの特徴



ホビスライトは、マイクロプロセッサであるATMega128を使用した8bitの制御が搭載されたヒューマノイドロボットです。サーボモーター、様々なセンサ、様々な変身型ロボットを通したロボット自体の教育の目的のために製作されたロボットです。
上記の8bitの制御機はDRCと呼ばれ、ホビスライトの頭脳の役割をします。DRCとモーター、センサを制御するためのソフトウェアツールであるDR-Visual Logicを提供します。C言語のように変数の宣言と関数の実行、ロジック構成が可能なグラフィックス開発言語です。また、16軸が使用されるホビスヒューマノイドの複雑さを考慮して、簡単にモーションを作成できるようにDR-SIMというモーション作成ツールを提供します。
上記ホビスライトの足をオムニホイールに変えて、カバーを被せた後アンドロイド端末機を装着したものがホビスジニです。したがって、ホビスジニはアンドロイド端末機を介してDRCと各種センサを制御します。

(2) Hovis Genie

ホビスジニは基本的にホームサービスを目的として誕生したロボットです。アンドロイド端末機を採用して、各種サービスアプリケーションをアプリの形で実行すると目的に合ったサービスが実行されます。



ホビスジニには、頭の部分にあるタッチセンサ,フラッシュ、目のLEDを通して感情を表現でき、腕にはTactスイッチを装備して手のタッチによるシナリオも可能です。ジニの利便性と安心感は下部のオムニホイール車輪にあります。四方360度のどの方向にも右折せずに直接行くことができ、自動充電に適しており、教育にも活用が可能です。オムニホイール周辺には、距離センサ、床センサなどがありロボットが障害物を避ける時に有効に使用されます。

ホビスジニは組立型と完成型の二つの種類で提供します。組立型は、Androidを通じたロボット教育に活用するための目的とホビスライトのように受講生が最初からハードウェアを自分で組み立てた後、ソフトウェアを操作してみることができます。完成形はアパートなどのホームサービスに直接入るための目的で一般ユーザーはプログラミングなしでロボットの中にあるアプリを活用して、使用することができます。また、完成形もアンドロイド教育に活用できます。組み立て型の組み立てマニュアルは(株)Donghbu　Robotホームページ資料室とwww.hovis.co.kr/ guideからダウンロードして使用可能です。

1.3 HOVIS Genie のハードウェア構成

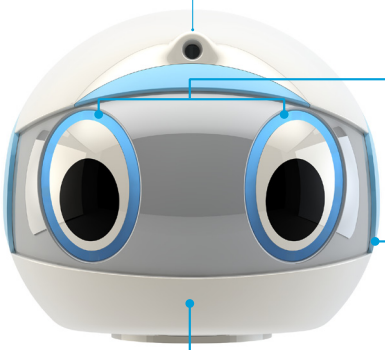
ホビスは基本的にロボット本体とMIDで大きく分かれます。MIDは装着が可能で、装着していない時には、一般的なアンドロイド端末機として活用できます。



(1) ロボット本体

ハードウェアには、ロボットの基本的な構成であるモーター、ブラケット、距離センサ、IR通信機などがあり、外部から機構を包んでいる形です。自動充電が可能な充電ステーションとアダプタ、リモコンなどのアクセサリが基本的に提供されます。

ジニの入力部分はMIDのタッチと音声認識、頭のタッチセンサ、手のTactスイッチ、車輪の距離センサと床センサなどを通じ、ロボットに外部値が入力されます。出力部分はMIDの画面、音声、音楽、頭部の目LED、モーターを基にしたロボットの動作などがロボットの出力表現を担当しています。この二つの入出力をもとにロボットサービスが行われ、プログラミングも可能です。



ロボットの電源ボタン

頭を触るとジニが感じ反応します。

自律モードで移動している時の頭をタッチすると、
ジニが一定時間の間動作を止めます。

目(LED)

充電中に充電状態に応じて、目の端の色が変わります。

- 充電完了時：青
- 充電中：赤
（充電されるほど赤い線が長くなる）

耳(LED)

- 平常時（自律/停止モード）は両方の耳が白に点灯しています。
（点滅していない）
- 音声認識モードの場合は認識可能な状態の時に点滅します。

口(LED)

ジニが話をしたり感情が変化した時に
1～3つの白色のLEDが点滅します。

(3) ホビスジニの仕様

ホビスジニは大人の膝の高さくらいのサイズです。自動充電やMIDを通したカメラ遠隔制御が可能です。詳細仕様は下記の通りです。

	項目	詳細仕様
ロボット	モデル名	HOVIS Genie
	サイズ	255(W) X 236.5(D) X 408.5(H)
	重さ	3.4kg
	モーター	11個
	センサ	障害物感知センサ,断崖センサ、IRセンサ、 タッチセンサ,Tactileセンサ,位置感知センサ
	移動方法	Omni-directional Drive / HerkuleX 0102 3ea
	ネットワーク	IR Receiver, ZigBee[option]
	外部スピーカー	External 2 way stereo Speaker(1W 2Ch)
	バッテリー容量	7.4V / 3,000mAh / リチウムポリマー
MID	ディスプレイ	3.5”TFT(480X320)
	画面の入力方式	停電式 Full Touch
	メインプロセッサ	MID : S5PC110, 1GHz
	カメラ	Front Rear Camera
	オーディオ	1ch MIC(MID 内蔵)
	内蔵メモリ	256MB
	外部メモリ	Micro SD 4G
	ネットワーク	Wi-Fi
その他	リモコン	IR 方式 / ZigBee Remocon[option]

1.4 HOVIS Genieのソフトウェアの特徴

ホビスジニは開発面で二つのソフトウェアの特徴があります。ロボットの制御機やセンサ制御ボードに入るファームウェアとMIDアプリケーションを開発するための応用ソフトウェアです。ファームウェアはC言語を使用するAVRのプログラミングであり、MID応用ソフトウェアはJavaを活用するアンドロイドのプログラミングです。(株)Dongbu Robotではアンドロイドのエンジニアによるロボットサービス用のAPIを提供します。

(1) MPSU (DRC-005T), OPSU (DRC-004TO), MID Side Board - ファームウェア

上記のボードは、ロボットの背中と内部に装着されてあり、ロボットのモーターとセンサを制御します。注命令はMIDから受けて、MPSUとOPSUには関連するファームウェアプログラムが埋め込まれています。MPSUはDRCと呼ばれ、ホビスライトのDRCと同じです。 MIDからの命令を受けてモーターや頭のLED、MID Side Boardなどを制御し、頭のタッチセンサ、肩の距離センサ機能も実行します。
OPSUはホビスジニの腹の奥にあるボードで、床センサ、距離センサ、充電ステーションのIRセンサなどの値を提供します。
MID Side Boardは、ホビスジニの胸とMIDの下に装着されたボードで、ジニのスピーカー、MID充電、MIDとMPSUの通信レベル変換などの機能を実行します。
MPSUとOPSUにはATmega128マイクロプロセッサが入っています。AVRのプログラミングによってファームウェアを作成します。ホビスライトではMPSU、つまりDRC AVR firmwareのプログラミング方法と教材を提供しています。しかし、ホビスジニMIDでサービスの実装を目的とするため、メーカーで組込んだファームウェアをそのまま使用します。したがって、他のエンジニアやユーザーがファームウェアを変更することは禁止しています。

(2) MID - Android

MIDはMultimedia Internet Deviceの略です。アンドロイドのOS以前にも、多くのスマートミニ端末機がありました。(株)Dongbu RobotのMIDはアンドロイドのOSを搭載し、アンドロイドのマーケットで通用するアプリケーションのダウンロード新規サービスを受けることができる端末機です。既存の端末機と異なる点は、ホビスジニはロボット本体を操作できるロボットサービスのプログラムが内部に含まれていることです。この内容は、Chapter.2から説明します。
MIDは、システムソフトウェア(ファームウェア)とアプリケーションに分かれます。アンドロイドのOSはLinuxベースのシステムで端末機自体のファームウェアを構成します。(株)Dongbu Robotは、システムソフトウェア、プログラミングの教材については、後で編纂する予定です。
ホビスジニのソフトウェアの最終的な目的は、MIDのファームウェアではなく、MIDのアンドロイドのアプリケーション開発です。JAVA言語で開発されたアンドロイドのアプリケーションは、すでにアンドロイドマーケットなどで数千万個が販売されています。同じようにホビスジニではロボットのアンドロイドのアプリケーションを開発してMIDにダウンロードすると、端末サービスではなく、ロボットサービスを実行することができます。

1.5 HOVIS Genieのアプリケーションを実行する

ホビスジニのMIDをオンにするとアンドロイドの端末機内部のプログラムではなくホビス関連プログラムがすぐにメイン画面に表示されます。基本アンドロイドのアプリは後ろに送り、ロボット関連のメインメニューに再構成したホビスジニ用ランチャー(スタート)プログラムです。

(1) メイン画面

ロボットをオンにすると画面にロボットの心臓の図と音が出てから、すぐにメイン画面に移動します。メイン画面は、上部のメインメニュー3つ、下のショートカットボタン、ホーム、ロボット変換ソフトボタンで構成されています。



メインメニューには、カメラや写真、音声に関連するマルチメディアメニューとして、朝の起床情報や遠隔制御などロボットに関連するホビスアプリメニュー、子供たちが見る童話や童謡などのコンテンツを集めた教育メニューで構成されています。マルチメディアは、端末機自体の基本的なアプリケーションの中でロボットに必要なマルチメディアを集めたものであり、ホビスアプリは、ロボットに関連する、つまり、ロボットの動作や、LED、タッチ、音声などのロボット要素を活用して開発されたロボット機能を活用するアプリケーションを集めたものです。そして、ホビスジニは基本的にホームロボットのコンセプトです。ホームで子供の教育に活用できるコンテンツである童話や童謡などはフラッシュや簡単なアプリで大量生産して配布しています。2番目のホビスアプリと第三の教育コンテンツのアプリは、(株)Dongbu RobotのMIDのアプリのストアから再ダウンロードが可能で、他のアプリケーションとコンテンツも接続してダウンロードすることができます。

そして、既存のアプリケーションも更新ファイルがあれば再ダウンロードが可能です。まだ外部公開用アプリストアではありませんが、Dongbu Robot用に簡単なアプリストアがあります。

(2) 自律行動

人々はロボットというと、TVや映画に出たアトムやターミネーターのようなロボットを想像します。しかし、そのようなロボットは実用性がなく、現実的に実行もできません。したがって、最も実用的に使われるロボットを選ぶとすれば、それは産業用ロボットと言えます。産業用ロボットは、目もなく、足もありません。ただ腕だけが動いて、正確な位置に正確な行動をすることでモノを作っています。一種の自動化機械です。

ホビスジニはそんな自動化産業ロボットではなくTV映画のアトムのようなヒューマノイドロボットです。当然実用性は小さいが、ホームサービスを介してその実用性を高めるために作ったロボットです。アトムは人とほぼ同じように行動して動きます。それは漫画だから可能なことです。現実的に作ったロボットの中で人と最も類似したロボットはまだないと見るべきです。ロボットが人に似ているということは人の行動を真似するということであるが、その中で最も優れた機能は、ロボットが自律的に行動することです。

自律ロボットは、モーター、タッチセンサ、床センサなどを利用して、センサ反応に応じて動くものです。しかし、その機能には限界があります。



1 장애물을 만났을 때 행동

주행 중 장애물을 만나면 제자리에서 몸을 돌려 다른 곳으로 이동합니다.

2 머리를 터치했을 때 행동

1 쓰다듬으면 기분이 좋다는 반응을 보입니다.

2 머리를 때리면 기분이 안 좋다는 반응을 보입니다.

3 배고플 때 행동

배터리가 20% 남아있으면 자동충전을 실시합니다.

4 시간대별 행동

1 아침에는 오늘 날씨를 물어보며 좋은 하루를 보내라고 말합니다.

2 점심에는 즐거운 점심시간이라고 말하며 점심은 먹었는지 물어봅니다.

3 저녁에는 오늘 날씨는 어땠는지 오늘 무엇을 하며 지냈는지 물어봅니다.
또한 저녁 9시가 되면 자야할 시간이라고 말하고 10시가 되면 잠잘 준비하자고 말하며 양치질이나 세수를 하는 동작을 합니다.

5 자율행동

심심해하기, 사랑 표현하기, 애교 부리기, 시간 묻기, 혼자 놀기, 기지개 펴기, 머리긁기, 체조하기, 지휘하기 등 여러 가지 행동과 말을 합니다.

6 손 택트 스위치를 클릭했을 때 행동

1 **왼쪽 손 클릭 시 –** '안녕'하고 인사를 하거나 뽀뽀하며 네가 좋다는 긍정적인 감정표현을 합니다.

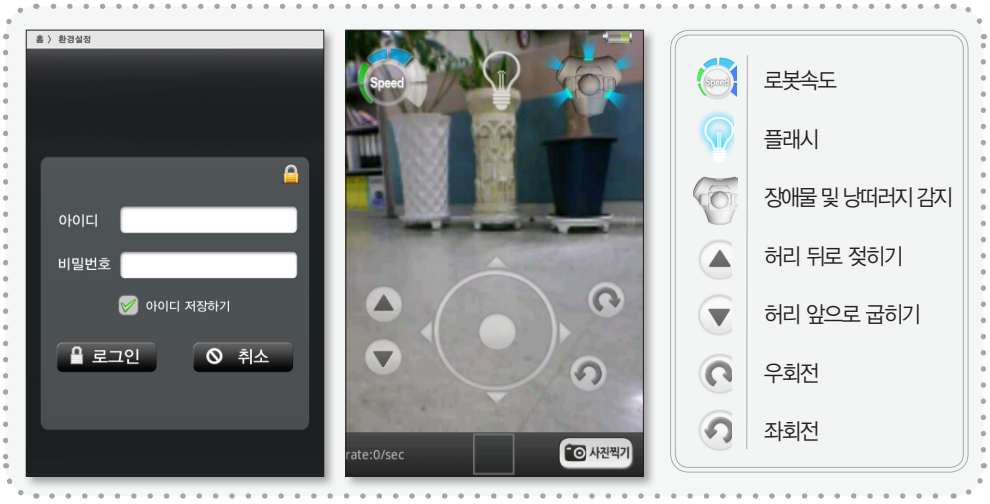
2 **오른쪽 손 클릭 시 –** 손목시계를 보는 시늉을 하며 현재 시간과 요일을 이야기합니다.

3 **양 손 함께 클릭 시 –** 양 손바닥 사이에 가볍고 탄력있는 공을 끼워 주면 공을 머리 위로 올린 후 공을 던져 줍니다 .

障害物に会った時には下の前方距離センサが作動して、頭を触った時には頭側のタッチセンサが反応します。バッテリーがなくなると自動的に充電器に移動します。また、時間帯別の行動パターンが異なる構成にしたり、ランダムにダンスをすることもできます。
自律行動はロボットのプログラマによってどれだけ詳細に製作するのかによってロボットの自律行動コンセプトが決まります。

(3) 遠隔制御

現代のロボットを制御する方法の中で最も多く活用されているのはロボットの遠隔制御です。家の中で使う掃除機にカメラを装着して家の中を遠隔から監視するように制御をしたり、火星にロボットを送って遠隔制御で火星探査ロボットを調整します。2004年にNASAが火星に送ったオファーチューナーティーは7年以上火星で探査任務を果たしました。探査方法は、オファーチューナーティーにあるカメラ映像を地球で受信し、地球から再び命令を下す方法です。電波伝搬速度は光の速度と同じなので、火星まで命令が送信される時間は4分27秒です。火星探査ロボットでも、掃除機ロボットでもカメラを用いた遠隔制御方法はすべて同じです。



同じようにホビスジニもMIDに搭載された前面部のカメラでの遠隔制御が可能です。消費者が持っているスマートフォンでホビスジニの制御アプリをインストールして、IDでログインすると、中継サーバを経てホビスジニMIDと接続されます。MID画面は遠隔制御していると表示されます。スマートフォンで移動UIを使ってホビスを移動させることができ、MIDに保存されたロボットのモーションを実行させることもできます。また、制御のアプリで文字を送る機能がありますが、文字を書いてMIDに送信すると、MIDはTTSから音声が出力されます。外部から見れば、ロボットが話すように見えます。

遠隔制御アプリは他のアプリとは異なる2つのアプリが必要です。MIDにインストールされているRe-cieveアプリとスマートフォンにインストールされている制御アプリです。現在、ほぼすべてのアプリは、MIDにインストールしてロボットの内部システムと連動するプログラミングをする必要があります。しかし、スマートフォンにインストールされている遠隔制御アプリは、一般的なスマートフォンアプリ開発と同じように開発できます。したがって、一般のエンジニアは最も早く製作することができ、簡単に自分のスマートフォンで操作できます。

メーカーから提供、制御するアプリではなく、一般のエンジニアたちが開発している制御アプリは、より様々な形で開発されます。モーション制御と一緒に音楽や音声を同時出力したり、スマートフォンのジャイロセンサを利用してロボットを移動させるなど、様々なサービスが可能な制御アプリを作成できます。

(4) 朝の起床情報/童謡童話などの情報コンテンツのロボットコンテンツ化

ロボットは動く端末機です。ユーザーが活用しようとするアプリを大量に流入できる生態系を作ると、ユーザーはロボットサービスを選んぶことができ、もっと簡単なサービスが登場することになります。

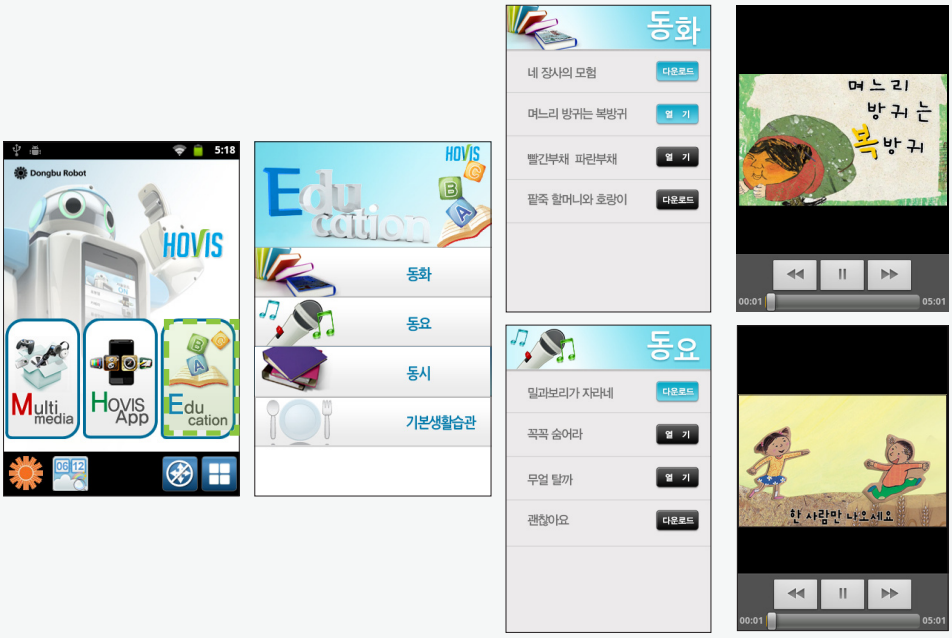
大量サービス生産のためには、現在のスマートフォンで使われている有用なアプリをロボットと組み合わせて大量に流入する方式が有効です。朝の起床情報がその例です。スマートフォンアプリ中のアラーム、カレンダー、 天気を合わせて、ロボットの動きを組み合わせたサービスです。個々のアプリをみるとスマートフォンと違うところはありませんが、例えば、朝にロボットがアラームを鳴らしながら部屋の中を声高に歩き回ります。その後、今日の天気の話をして、ユーザが記録しておいた日程を順番に言います。一種の家の秘書の役割を担います。

最も多くのコンテンツがある教育コンテンツ中の童話や童謡など、子供たちに活用できるアプリはロボットに細かい動作が加われば面白い立体アプリに生まれ変わります。子どもたちはロボット自体も面白いため童話や童謡に集中できます。

情報端末機のコンテンツとロボットのコンテンツは、動作の有無に応じて分類するとその差は小さくなります。しかし、結合された情報は、ユーザーにより多くの効果を与えます。これらのアイデアがメーカーや開発会社ではなく、一般のエンジニアたちが参加して作成した場合は、より多様なコンテンツが作成されると予想されます。



[朝の起床情報コンテンツ]



[童話、童謡のコンテンツ]

(5) ソフトウェア提供の目的と活用

ホビスジニの本体+アンドロイドGenieAPI+モジュールごとに制御例と教材を通してホビスジニアプリを開発できる環境を提供します。様々なアンドロイドのアプリの再生産と普及のためには、一般のエンジニアの教育が必要です。教育を受けたエンジニアがホビスジニ用アプリを多数作成できるように本教材を活用できるように、環境を提供する予定です。

ホビスジニロボットのサービスソフトウェアは、大きく、自律行動、遠隔制御、情報加工に分けられます。この3つの基本例は、(株)Dongbu Robotが提供します。本ソフトウェアをサンプルにして、様々なソフトウェアが再加工されて作られることを期待しながら、アンドロイドジニの紹介を終了します。

02 アンドロイドとロボットの開発環境を準備する

この章では、アンドロイドとロボット開発環境の準備について説明します。

アンドロイドとロボット開発環境を準備するには、次のように大きく4つの手順で行われます。

- JDKのインストール
- アンドロイドSDKのインストール
- エクリプスのインストールと環境設定
- MID 開発ボードの接続とデバッグ

2.1 JDKのインストール

JDK(Java Developmenet Kit)は、Java開発ツールです。アンドロイドは基本的にJavaに基づいて作成しました。したがって、Java開発ツールのインストールが必要です。JDKのインストールはアンドロイドの開発環境の準備に影響を与えるため、先に実行する必要があります。

01

Javaのサイトに移動してJava Downloadアイコンをクリック

http://www.oracle.com/technetwork/java/javase/downloads/index.html



02

Java SE Development Kit 項目からライセンスの同意をした後、自分のOSに合ったJDKをダウンロード
(例 : Windows 64bitはWindows x64, Windows 32bitはWindows x86)

Java SE Development Kit 7u5

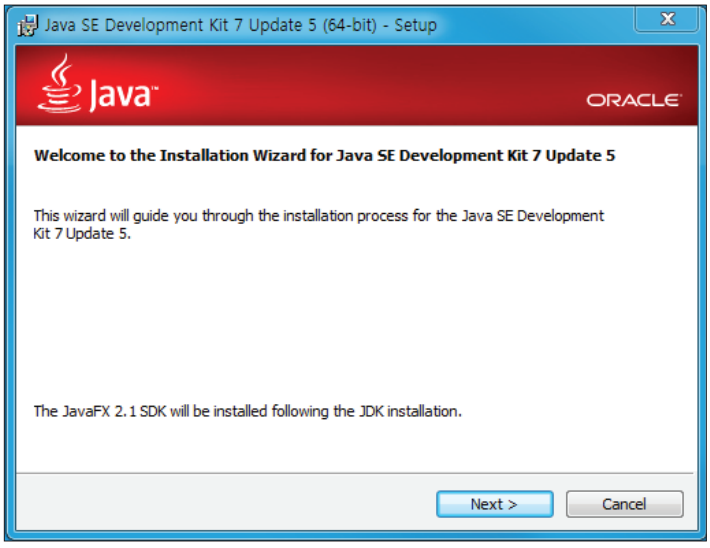
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

○ Accept License Agreement ◉ Decline License Agreement

Product / File Description	File Size	Download
Linux x86	64.1 MB	jdk-7u5-linux-i586.rpm
Linux x86	79.1 MB	jdk-7u5-linux-i586.tar.gz
Linux x64	64.93 MB	jdk-7u5-linux-x64.rpm
Linux x64	77.67 MB	jdk-7u5-linux-x64.tar.gz
Macosx-x64	97.3 MB	jdk-7u5-macosx-x64.dmg
Solaris x86	137.41 MB	jdk-7u5-solaris-i586.tar.Z
Solaris x86	82.01 MB	jdk-7u5-solaris-i586.tar.gz
Solaris SPARC	140.43 MB	jdk-7u5-solaris-sparc.tar.Z
Solaris SPARC	86.72 MB	jdk-7u5-solaris-sparc.tar.gz
Solaris SPARC 64-bit	16.45 MB	jdk-7u5-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	12.55 MB	jdk-7u5-solaris-sparcv9.tar.gz
Solaris x64	14.92 MB	jdk-7u5-solaris-x64.tar.Z
Solaris x64	9.54 MB	jdk-7u5-solaris-x64.tar.gz
Windows x86	87.95 MB	jdk-7u5-windows-i586.exe
Windows x64	92.36 MB	jdk-7u5-windows-x64.exe

03

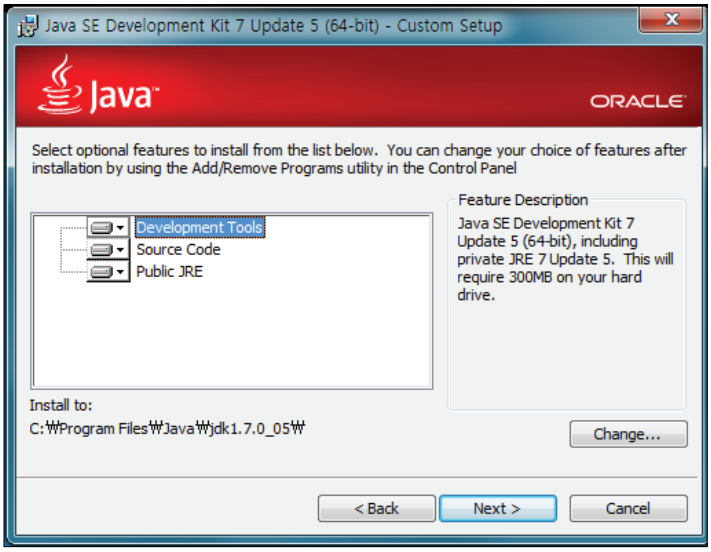
ダウンロードしたインストールファイルを実行して、JDKをインストール



04

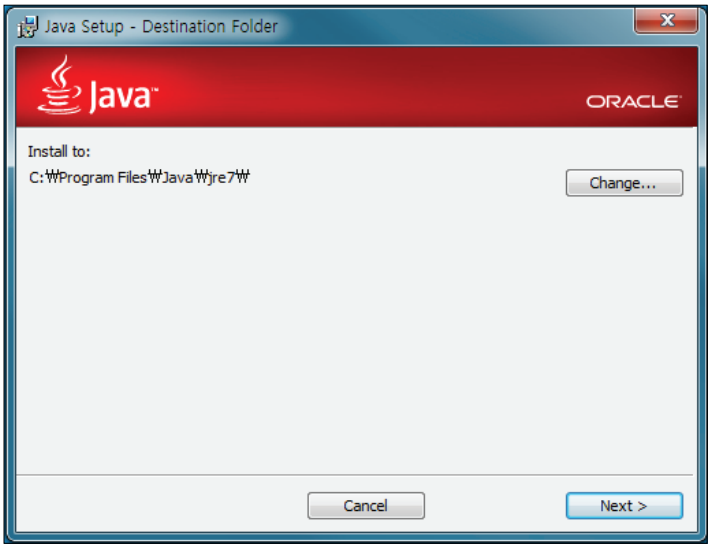
JDK インストール経路を指定

（基本的に指定された経路を使用して日本語が含まれている経路を指定しない）



05

JRE(Java Runtime Environment)インストール



06

インストール完了
（以降JavaFX SDKはインストールしなくてもよい）



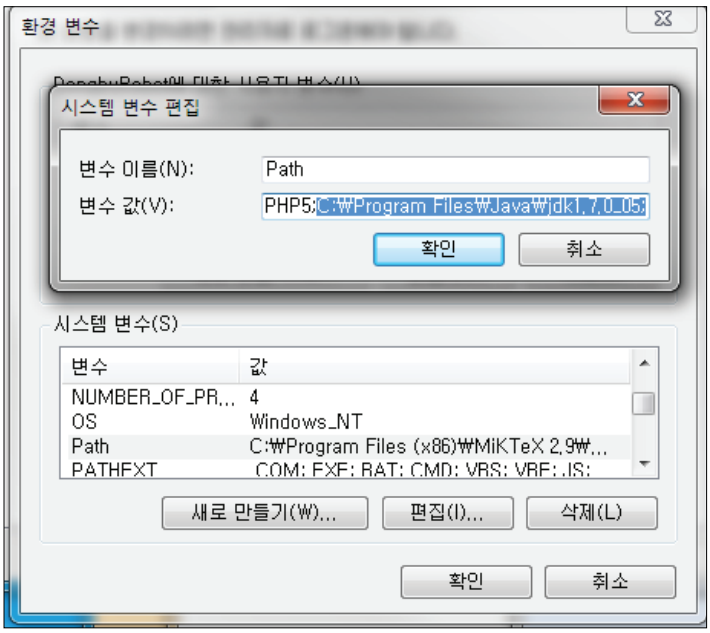
07

環境変数の設定コントロールパネル>システム>システムの詳細設定
>環境変数(N)をクリック



08

4番で指定した経路を追加する。ただし、ここではC:¥Program Files¥Java¥jdk1.7.0_05¥



2.2 アンドロイドSDKのインストール

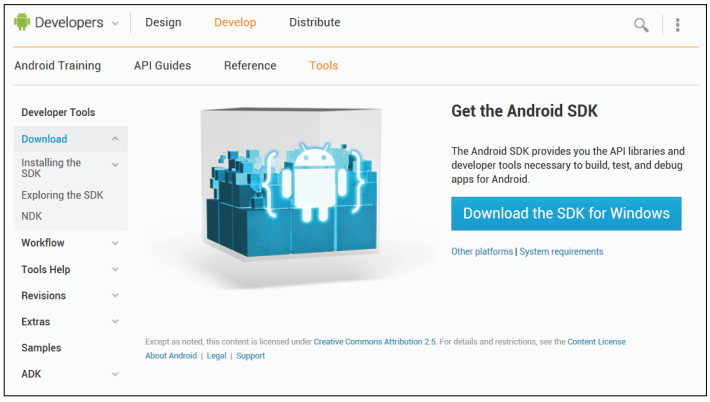
アンドロイドSDK（Software Development Kit）は、開発者がアンドロイドに基づいてアプリケーションを作成するために必要な様々なライブラリの集合です。アンドロイドSDKはアンドロイドの公式開発者サイトであるhttp://developer.android.com/からダウンロードできます。

アンドロイドの公式開発者サイトであるhttp://developer.android.com/では、アンドロイドの最新情報、初心者のためのチュートリアルとAPI（Application Programing Interface）の使い方などの多くの情報を提供しています。アンドロイドの発展は非常に速いので、本には最新のアンドロイドの情報がない場合がほとんどです。さらに、このサイトは、市販のアンドロイドの基本の本より多くの例を提供しています。
良いアンドロイド開発者として成長するためには、このサイトをよく利用することが必要不可欠です。この本ではアンドロイドを使ったロボットのプログラミングを重点的に扱っています。したがって、アンドロイドの開発に必要な補助的な情報を得るために、このサイトをしばしば訪問することをお勧めします。

これからアンドロイドSDKをインストールします。アンドロイドの開発者サイトを参照して下記のようにSDKをインストールします。.

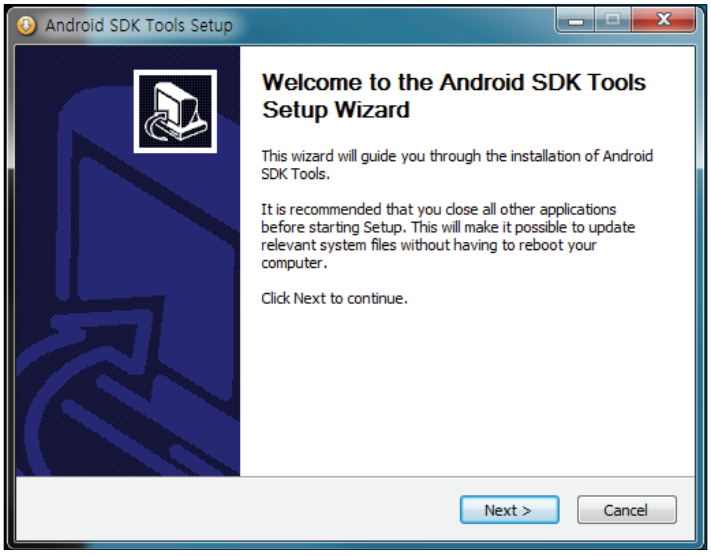
01

アンドロイドの開発者サイトに移動後、SDKをダウンロード
(http://developer.android.com/sdk/index.html)



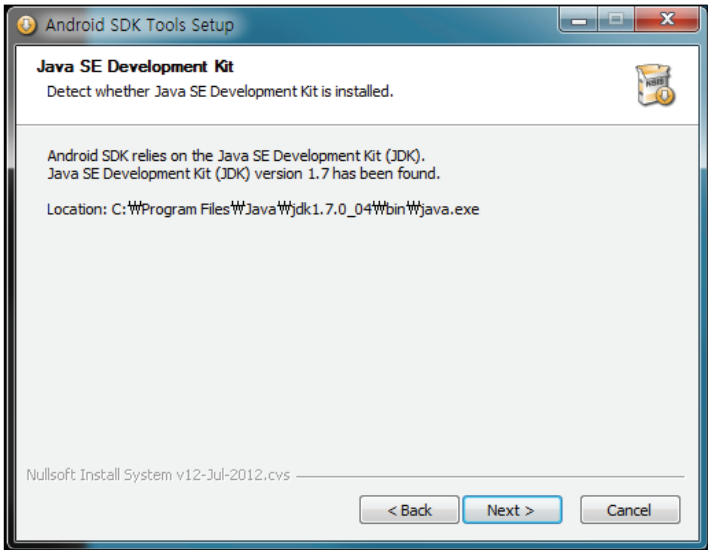
02

ダウンロードしたアンドロイドSDKのインストールファイルを実行し、Nextをクリック



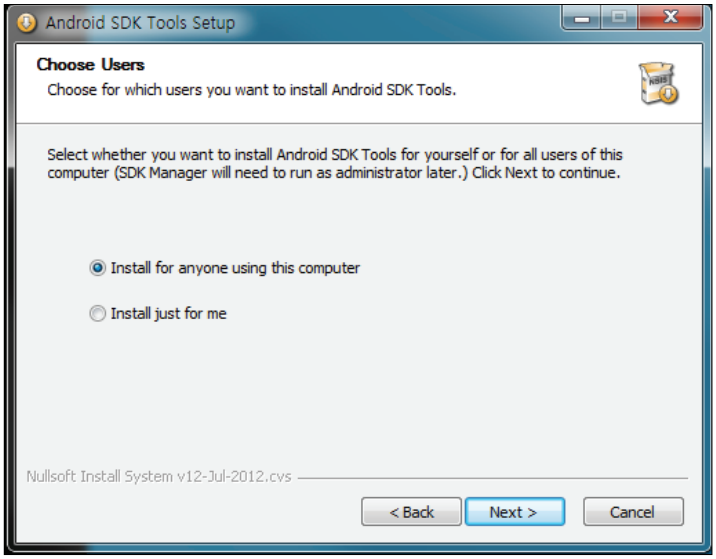
03

JDKのインストールかどうかを自動的にチェックします。インストールができたら、Nextをクリック
（JDKをインストールしていない場合は、JDKをまずインストール）



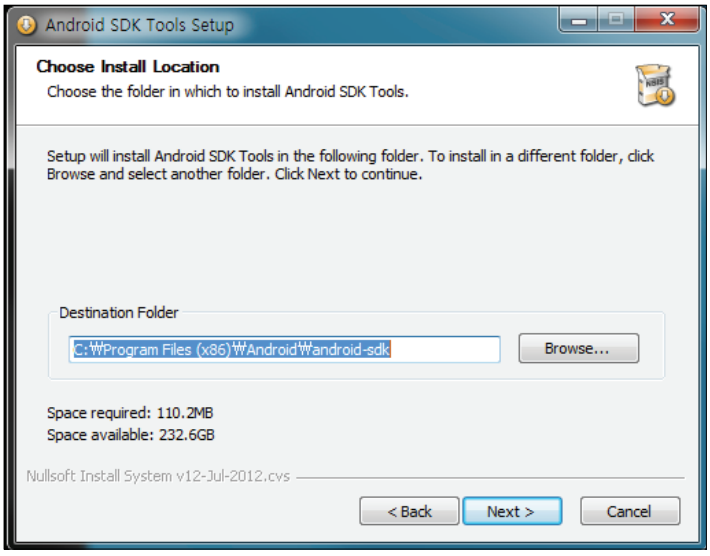
04

ユーザーの使用オプションを選択し、Nextをクリック



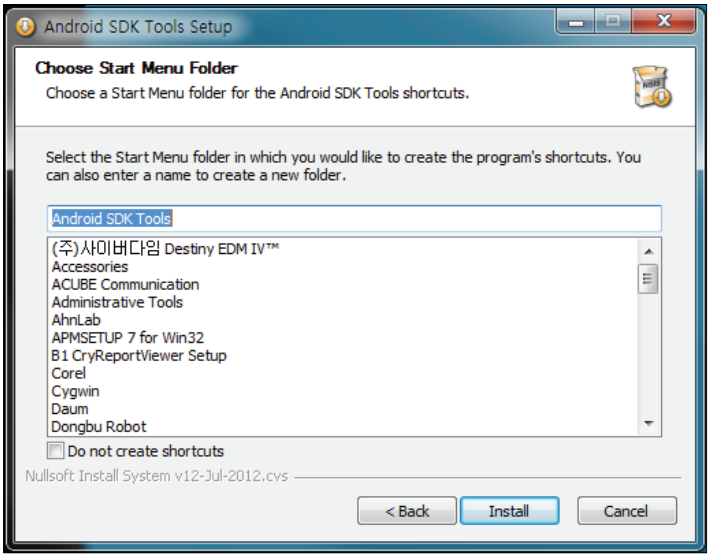
05

アンドロイドSDKをインストールする経路を指定し、Nextをクリック
（日本語を含む経路を指定しない。基本の経路の使用を推奨）



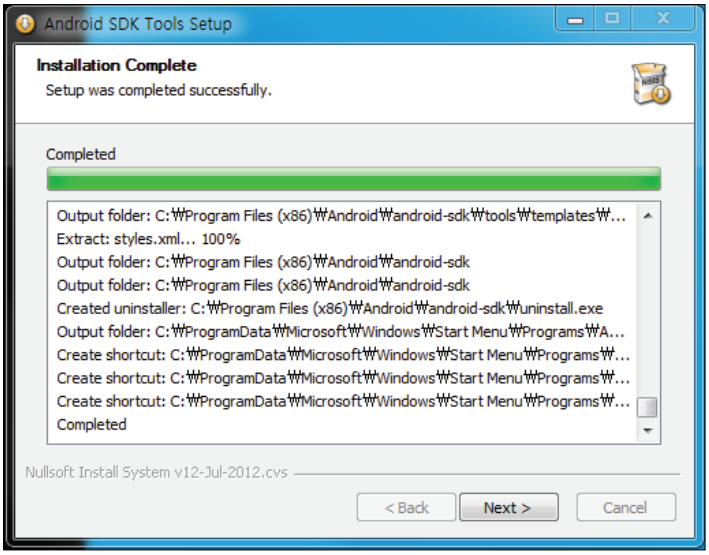
06

スタートメニューに登録される名前を選択した後、Nextをクリック



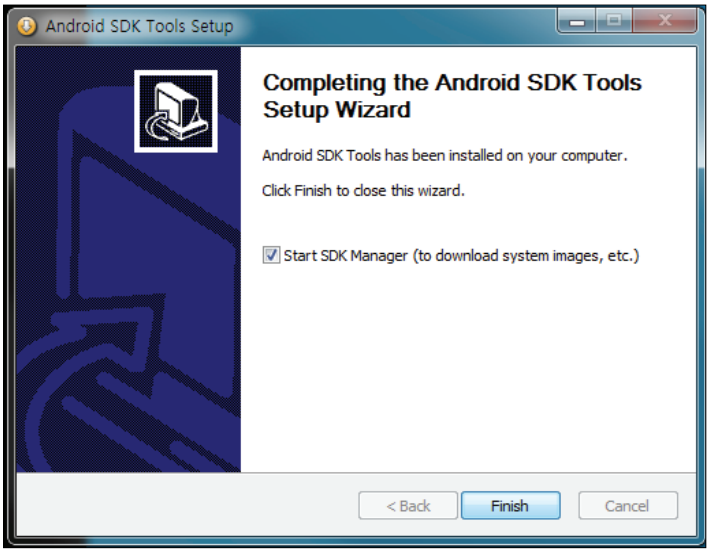
07

インストールした後、Nextをクリック



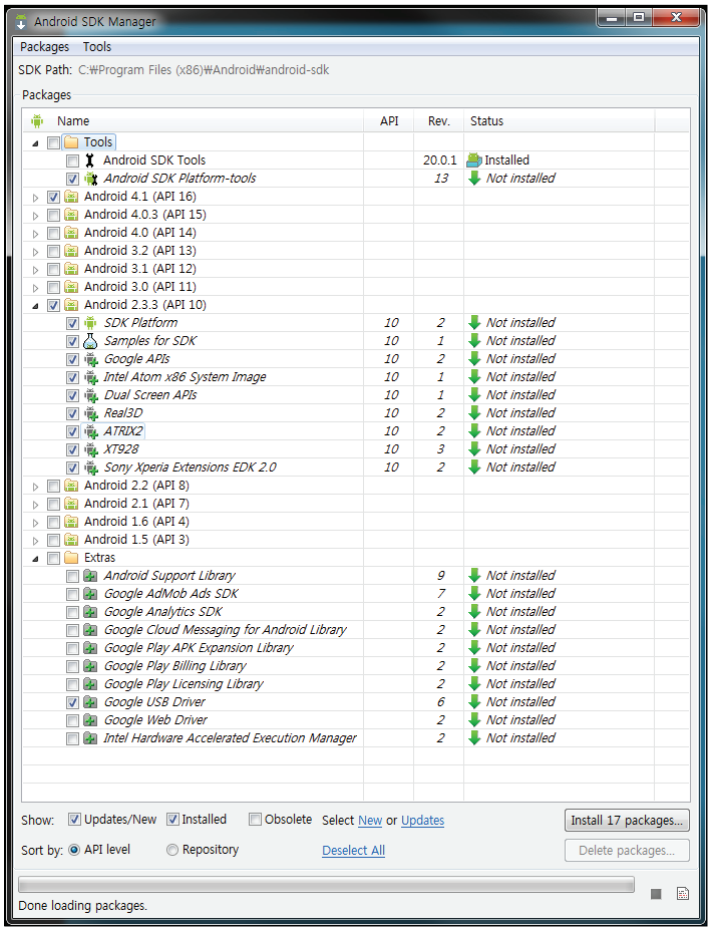
08

すべてのインストールが完了した後、Finishをクリックして SDK Managerを実行



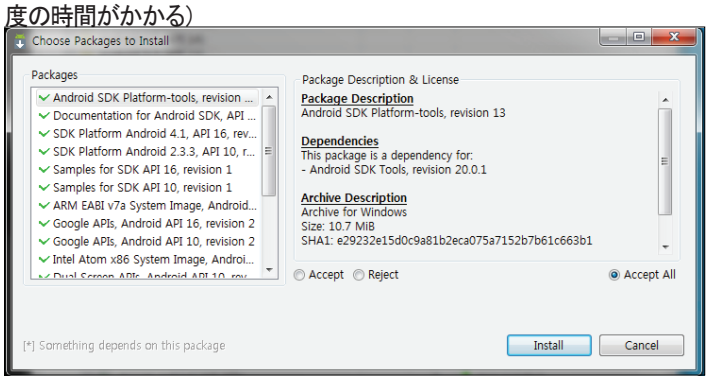
09

SDKマネージャーを実行すると、基本的な推奨事項がチェックされています。HovisGenieでのアプリ開発はAndroid2.3.3(API10)を利用するので、さらにAndroid2.3.3をチェックした後、Installボタンをクリック



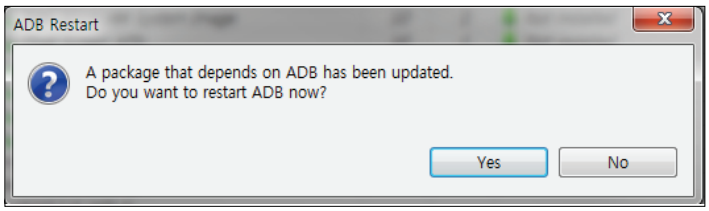
10

ライセンスに同意した後、Installをクリックしてインストール開始(このインストールプロセスは、1～3時間程度の時間がかかる)



11

インストールが完了したら、[Yes]をクリックしてADBを再起動して、SDKマネージャを閉じます。



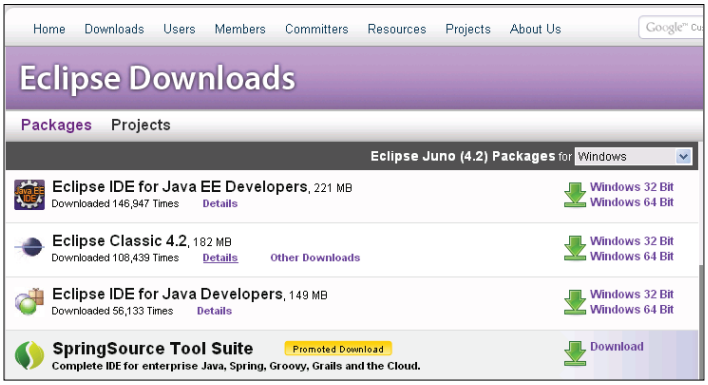
ADB (Android Debug Bridge)アンドロイドのアプリを開発する時にエミュレータや実際のアンドロイドデバイスの操作、デバッグに関連するインタフェースを提供します。

2.3 Eclipseのインストールと環境設定

アンドロイドの開発のために、様々なツールを使用しています。その中で最も多く使われているツールは、Eclipseです。Eclipseはオープンソースベースの統合開発環境としてソースコードの作成、デバッグ、および実行を便利に使えるように、様々な機能を提供する汎用の開発ツールです。
実はほとんどのアンドロイド開発者がEclipseを使用するというほど広く使われています。したがって、私たちもEclipseを使用してアンドロイドの開発環境を構築します。
Eclipseのインストールは、他のソフトウェアのインストールとは少し異なります。すべてのJavaベースのオープンソースプロジェクトのソフトウェアのインストールと同様に、ダウンロードした圧縮ファイルを解凍することで、インストールを完了できます。
Eclipseのインストールが完了したら、Eclipseをアンドロイドの開発環境で構築するためにADTPluginをインストールします。ADTは、Android Development Toolsの略で、Eclipse上でアンドロイドアプリの制作、UI(User Interface)の構成とアプリのデバッグなどを簡単に早くできるようにするツールです。
つまり、Eclipse上でのADT Pluginのインストールは Eclipseをアンドロイドアプリ開発に最適化された開発ツールに構築する作業です。
これからアンドロイドロボットアプリ開発のための環境を段階的に準備します。

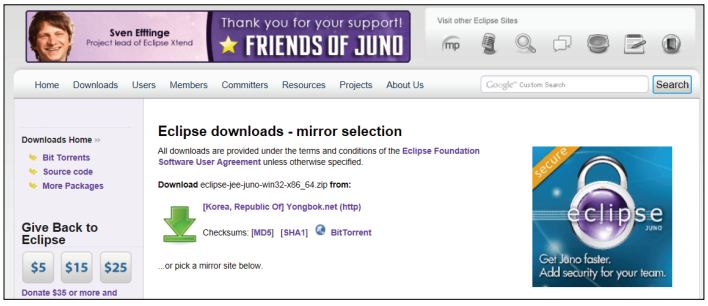
01

Eclipseのサイトに移動し、OSに合ったEclipse IDE for Java EE Developersを選択
(http://www.eclipse.org/downloads/)
*Eclipse3.6.2(Helios)またはそれ以降のバージョンをダウンロードする



02

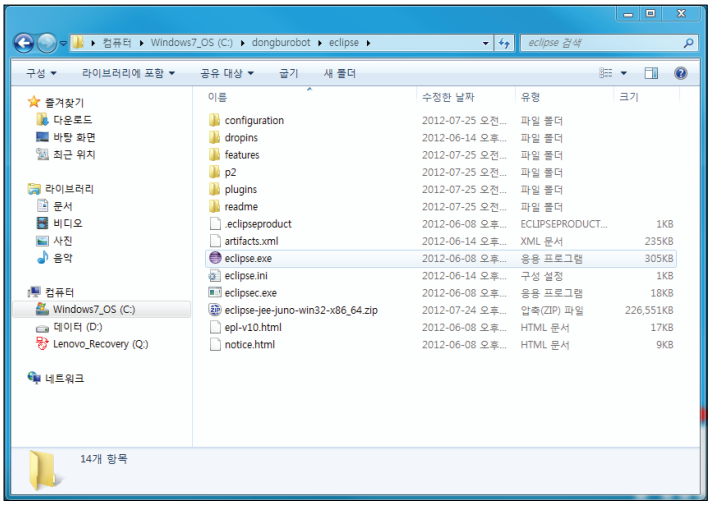
次のミラーサイトをクリックしてEclipseをダウンロード



03

ダウンロードしたEclipseの圧縮ファイルの解凍（ここではC:¥ dongburobot¥ eclipse¥）

注意：圧縮を解凍する経路に日本語を含まない。



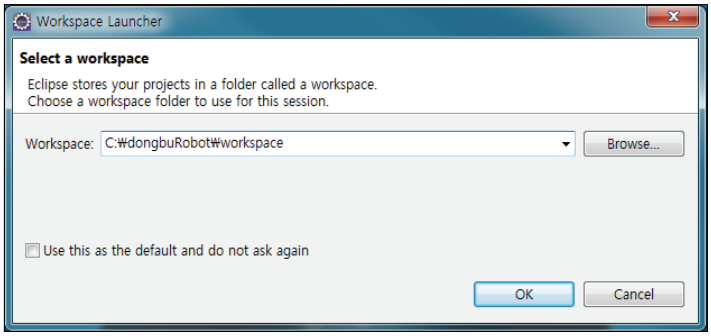
04

デスクトップにショートカットのアイコンを作成して、Eclipseを実行します。



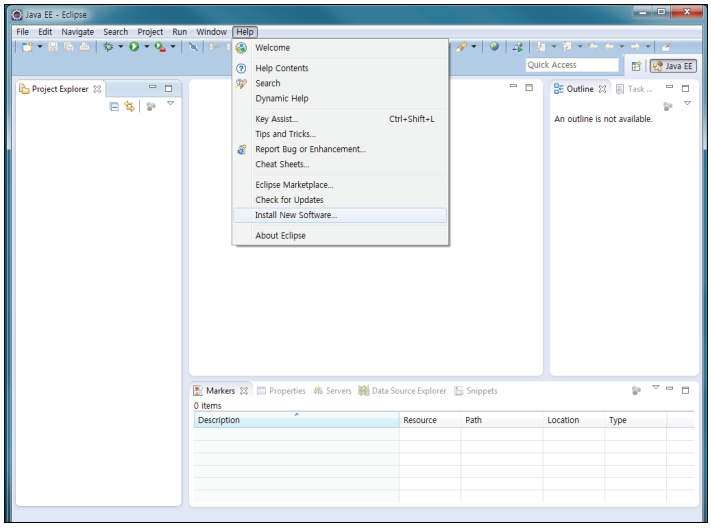
05

Eclipseを実行すると、Workspaceを指定してOKをクリックします。(Workspaceは、ソースコードを編集して保存するワークスペースを意味します。ここではC:¥ dongburobot¥ workspace¥)



06

EclipseでHelp> Install New Software...を選択します。

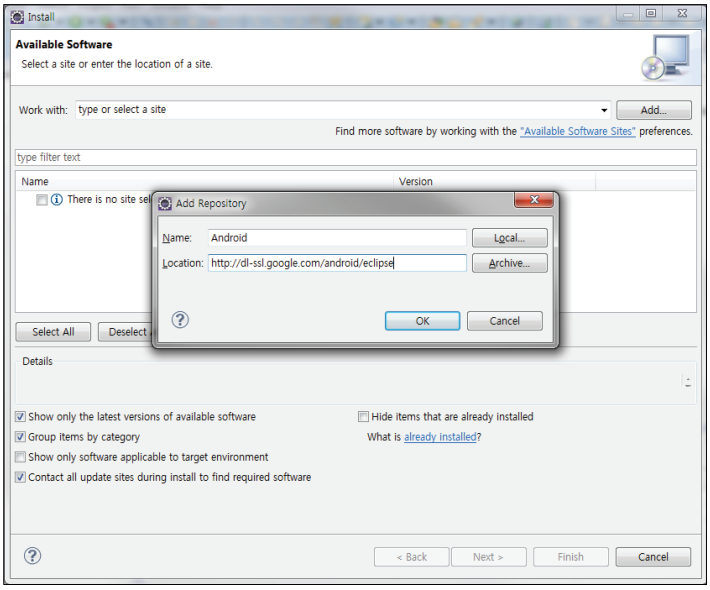


07

Addをクリックすると表示されるAddRepositoryウィンドウに次のようにNameとLocationを入力します。

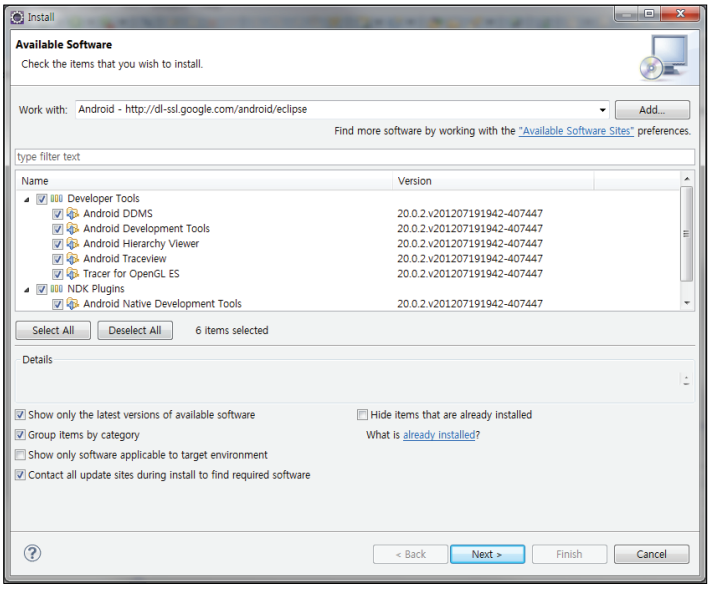
Name : Android

Location : http://dl-ssl.google.com/android/eclipse/



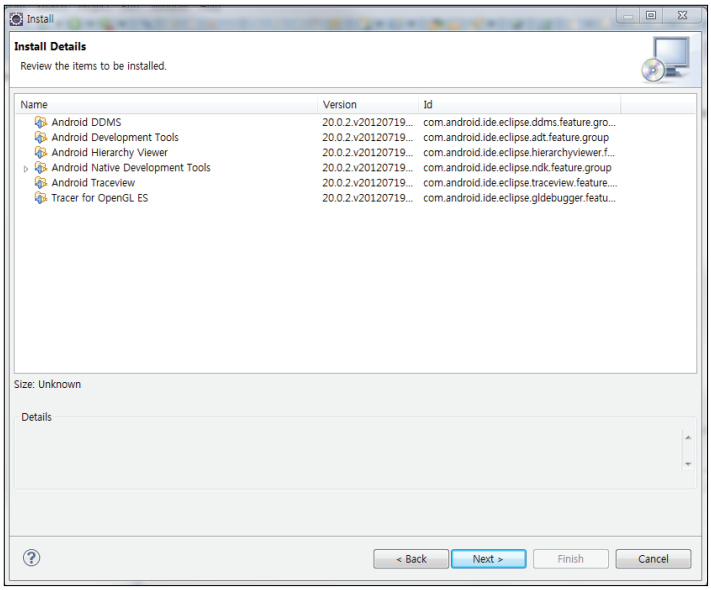
08

7で追加したRepositoryを選択すると、しばらくして次のような画面が表示されます。Select Allを選択して、[Next]をクリックします。



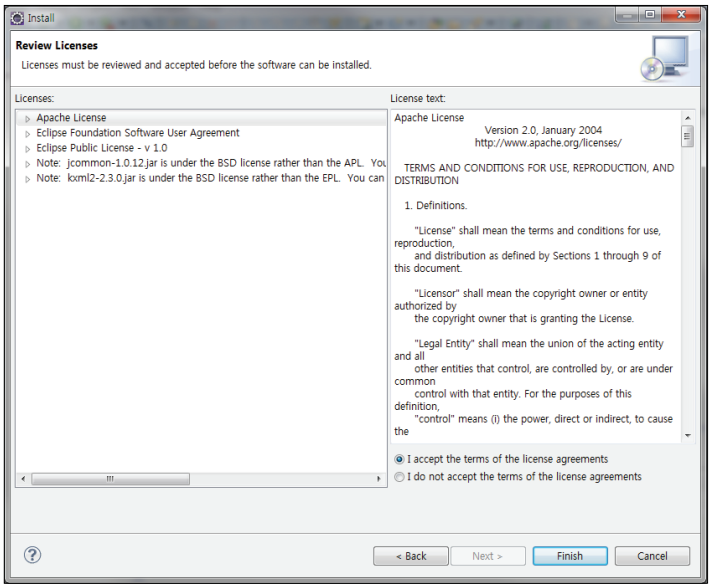
09

インストールするEclipseのPluginのリストを確認して「Next]をクリックします。



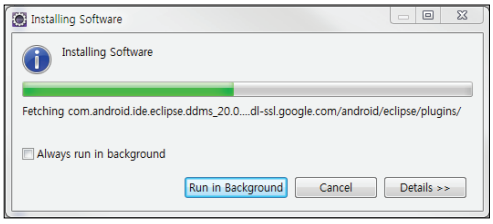
10

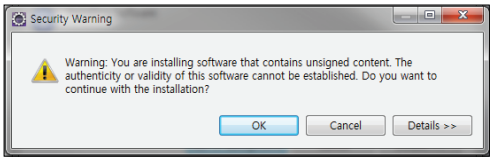
ライセンスに同意した後、[Finish]をクリックしてインストールを開始します。



11

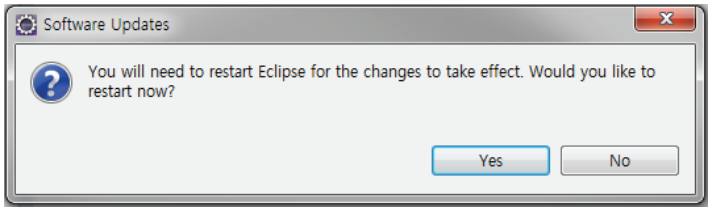
インストール中に次のようなセキュリティの警告が表示されることがあります。[OK]をクリックして進行します。





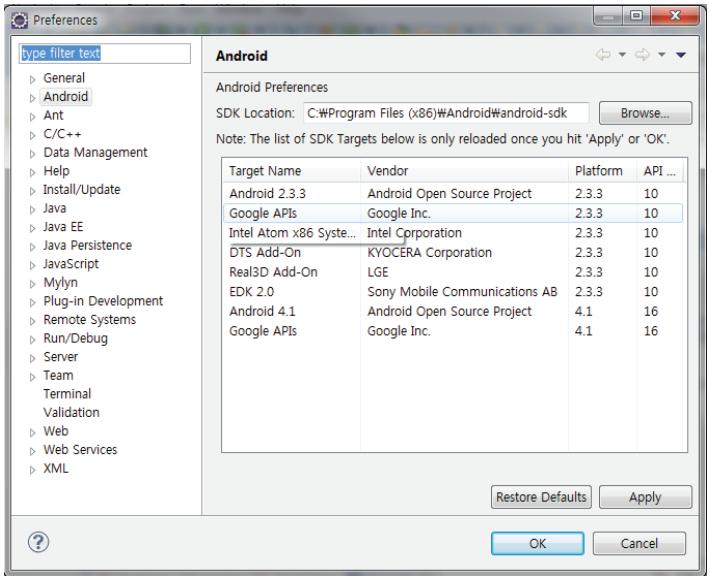
12

プラグインのインストールが完了したら、Eclipseを再起動します。



13

EclipseでWindow>Preferencesを実行して、AndroidのタブのSDKLocationを指定します。SDKLocationには、2.2章でインストールしたAndroidSDKの経路を指定します。Applyボタンをクリックして適用した後、[OK]をクリックして設定を完了します。



03 アンドロイドロボット(ジニ)プログラミング

Chapter2ではアンドロイドのプログラミングのための開発環境を準備しました。この章では、本格的にアンドロイド基盤のプログラミングを開始します。

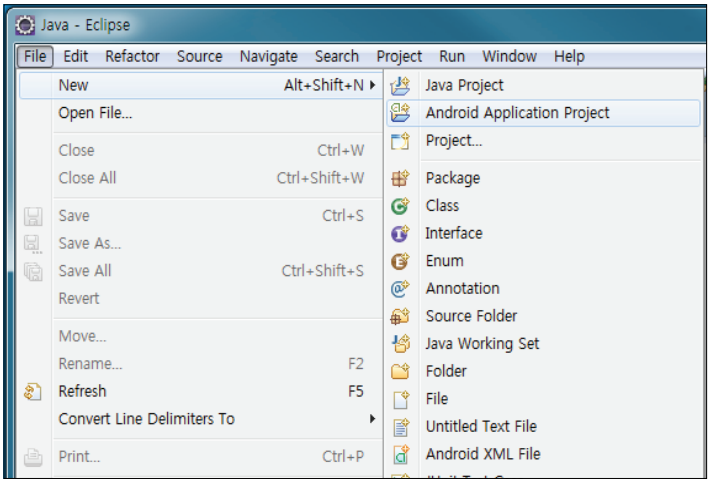
この章の目的は、アンドロイドのプラットフォーム上で動作する最初のアンドロイドAppの製作です。目標を達成するためにまず、アンドロイドの理解が必要です。最初のAppを作成するプロセスを介してアンドロイドの基本的な構造と特徴を説明します。しかし、アンドロイドは膨大です。この章では、すべての内容を扱うには限界があるのでアンドロイドの基本の本またはアンドロイド開発者サイト(http://developer.android.com/index.html)を参照してください。

アンドロイドのすべての内容を説明しませんが、この本の例を介してアンドロイドの全般的な理解ができます。それでは、私たちの目標であるアンドロイドベースのロボットプログラミングのために、最初のAppのHelloGenieを開始します。

3.1 Hello Genieプロジェクトの作成

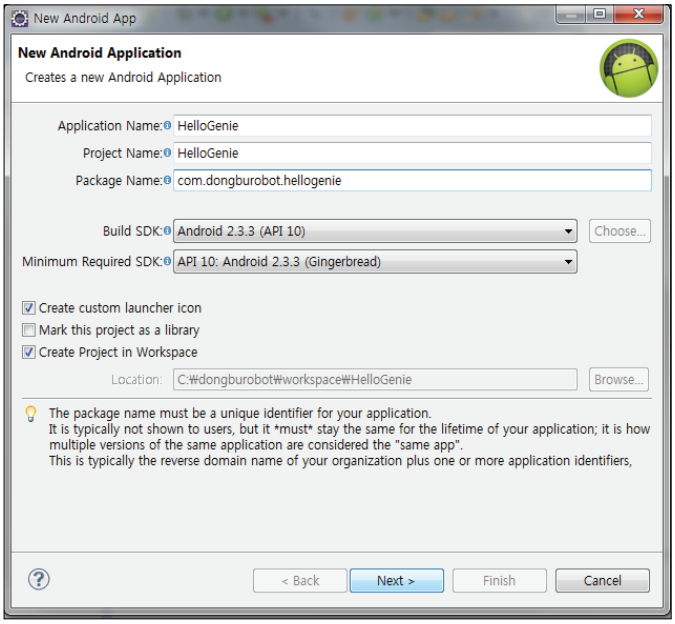
01

Eclipseを実行し、File> New> Android Application Projectを選択します。



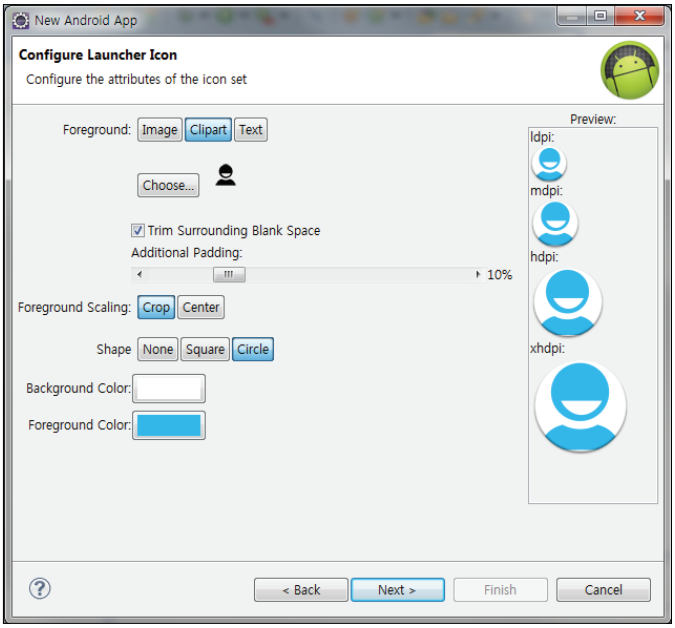
02

App、プロジェクト、パッケージの名前を指定します。この時、BuildSDKの項目は必ずAndroid2.3.3（API10）に指定します。その理由は、Hovis Genieに付着するMIDのアンドロイドバージョンが2.3.3であるためです。すべての名前を指定した後、[Next]をクリックして次に移動します。



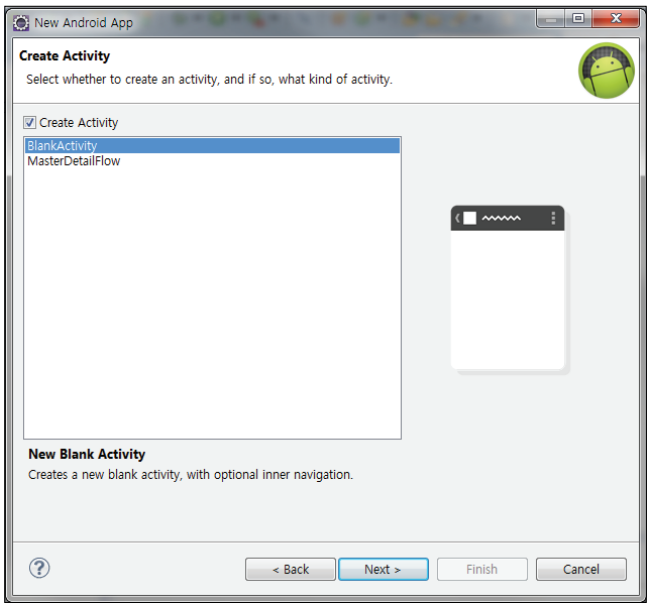
03

ランチャーアイコンを設定します。ランチャーアイコンとは、スマートフォンのAppを示すための四角いアイコンです。[Next]をクリックして次に移動します。



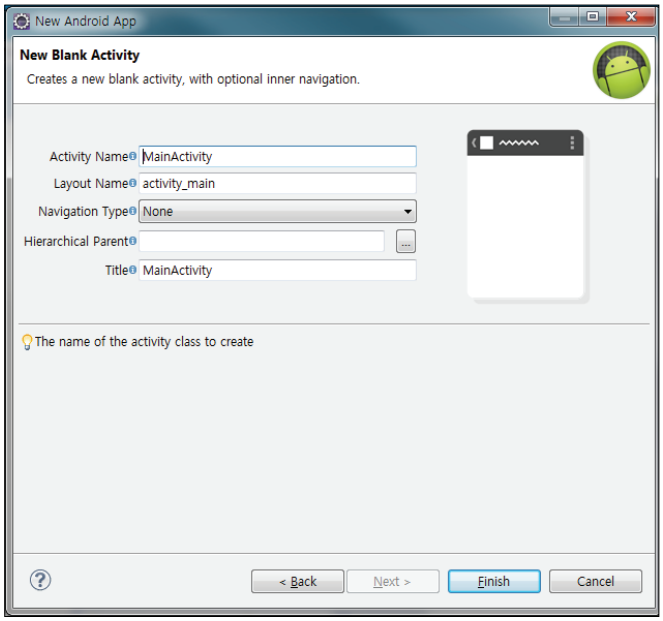
04

Activityを生成します。Activityとはアンドロイドフォンの1つの画面を意味します。たとえば、アンドロイドフォンのメッセージを見ている途中に電話が来て画面が切り替わる場合、一つのActivityが他のActivityに転換されたものです。このActivityの詳細については、後で説明します。BlankActivityを選択して[Next]をクリックして次へ移動します。(MasterDetailFlowは、Tabletで使用できるように新たに追加された機能です。この本では説明しません。)



05

生成されたBlankActivityの名前を指定します。ここでは、基本的に指定された名前を使用します。[Finish]をクリックしてプロジェクトの作成を完了します。



3.2 Hello Genie プロジェクトの実行

HelloGenieプロジェクトを実行します。Eclipseで作成したHelloGenieプロジェクトを実行する方法は2つあります。実際のアンドロイドデバイスと仮想のエミュレータデバイスでプロジェクトの実行ができます。ここでは、Hovis　Genieを制御するアンドロイドデバイスであるMIDを利用します。

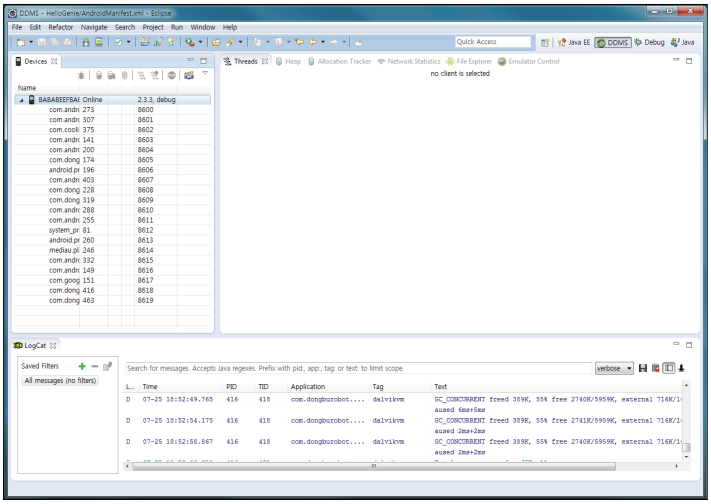
01

USBジャックを利用してMIDとPCを接続



02

EclipseのDDMSを使用して、デバイスの接続を確認し、デバイスを選択



03

Eclipseのツールバーから▶ボタンを押してプロジェクトを実行してMIDで実行された画面を確認します。

[TIP] もしMIDが検索されなかったら？

[設定> アプリケーション>開発> USBデバッグ]でUSBデバッグを無効にして再度チェックする

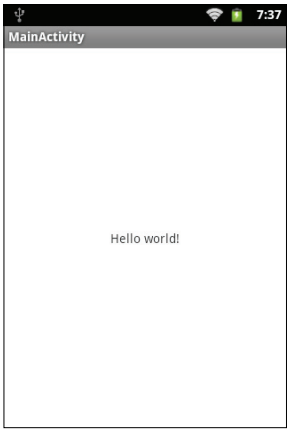
[TIP] もし起動しない場合は？

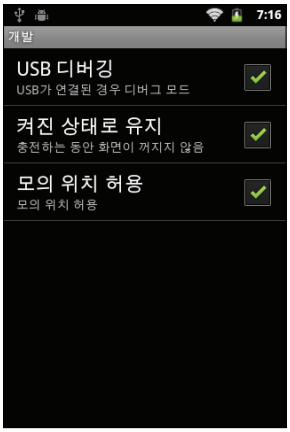
ボタンをクリックしてxmlソースコードが選択されている場合は実行されません。この場合、*. xml.outというファイルが生成されるため、このファイルを消して/srcの任意のjavaファイルを開き、実行ボタンを押して実行してください。

[TIP] 次のようなエラーメッセージが出たら？

Parser exception for /HelloGenie/ AndroidManifest.xml: 文書のルート要素の後にあるマークアップが正しい形式である必要があります。

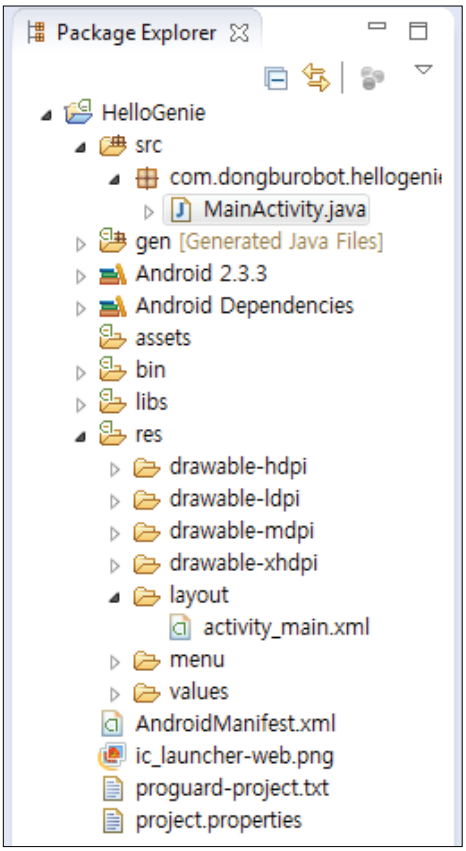
Eclipseに追加されたADT Pluginのバグです。次のソースコードの説明を参照して、ソースを変更してください。





3.3 Hello Genie プロジェクトの基本構成

最初のAppのHelloGenieプロジェクトを作成して、実際のアンドロイドデバイスであるMIDで実行しました。次は作成されたHelloGenieのプロジェクトファイルを確認します。 HelloGenieプロジェクトの基本構成は、Eclipseの左にあるPackage Explorerウィンドウで確認できます。構成は次のとおりです。



/src

Appのソースを含んでいるディレクトリです。現在は、プロジェクトを作成する時に指定したMainActivityを基本的に含みます。

/gen

このディレクトリにあるファイルは自動的に作成されます。リソースと環境に関する情報があるファイルです。

/bin

プロジェクトを作ると作成されるファイルがあるディレクトリです。

/libs

プロジェクトに含まれる外部ライブラリがあるディレクトリです。今後のロボットサービスを提供するライブラリを含んでいる時にこのディレクトリを利用します。

/res

プロジェクトのリソースを含みます。

/res/drawable

プロジェクトに使用されるイメージファイルを含みます。

/res/layout

このディレクトリは、画面のUI(User Interface)を説明するファイルを含みます。一つのActivityは一つの画面を示します。一つのActivityは、このディレクトリに含まれている一つのxmlファイルを使用します。

/res/menu

このディレクトリは、アンドロイドフォンでメニューボタンを押した時に出る画面について説明するxmlファイルを含みます。

/ res / values

その他のリソース(文字列、またはスタイル)を指定するxmlファイルを含みます。

AndroidManifest.xml

このファイルは、プロジェクトのコンポーネントの構成と基本的なプロパティについて定義します。
一つのプロジェクトには、一つのAndroidMnaifest.xmlファイルがあります。

HelloGenieプロジェクトの基本的な構成は、多くのディレクトリうを含みます。この中の/src、/ res、そしてAndroidManifest.xmlファイルは、App駆動理解に必要な部分です。

(1) AndroidManifest.xml

まず、AndroidManifest.xmlファイルです。一つのAppは、一つのAndroidMainfest.xmlを含みます。AndroidManifest.xmlファイルはAppのコンポーネントの構成を定義して、Appの権限とプロパティを指定する役割をします。つまり、Appの重要な設定を含んでいるファイルです。以下は、AndroidManifest.xmlファイルです。

```
1: <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2:     package="com.dongburobot.hellogenie"
3:     android:versionCode="1"
4:     android:versionName="1.0" >;
5:
6:     <uses-sdk
7:         android:minSdkVersion="10"
8:         android:targetSdkVersion="10" />;
9:
10:    <application
11:        android:icon="@drawable/ic_launcher"
12:        android:label="@string/app_name"
13:        android:theme="@style/AppTheme" >;
14:        <activity
15:            android:name=".MainActivity"
16:            android:label="@string/title_activity_main" >;
17:            <intent-filter>
18:                <action android:name="android.intent.action.MAIN" />
19:                <category android:name="android.intent.category.LAUNCHER" />
20:            </intent-filter>
21:        </activity>
22:    </application>
23:
24: </manifest>
```

Listing1. AndroidManifest.xml

AndroidManifest.xmlの最上位の構造<manifest>タグです。このタグは、Appの基本の情報が含まれています。2行目のPackageプロパティは、プロジェクトを作成する時に指定したパッケージ名です。3～4行目のandroid:versionCode、android:versionNameは、Appのバージョン情報を指定します。このプロパティは、今後Appをアップグレードして再配布する時に変更する部分です。

<manifest>のサブタグは<uses-sdk>と<application>があります。<uses-sdk>タグは、Appの動作環境を指定する役割をします。7行目のandroid:minSdkVersionプロパティはこのAppが動作できる最小限のアンドロイドバージョンを指定する部分です。8行目のandroid:targetSdkVersionプロパティはAppが使用しているAPILevelを意味します。API Level10はアンドロイド2.3.3ジンジャーブレッドのバージョンです。現在、ロボットに搭載されるアンドロイドデバイスであるMIDのアンドロイドのバージョンが2.3.3であるため、プロパティを10に指定します。

AndroidManifest.xmlの最上位構造<manifest>タグです。このタグは、Appの基本情報が含まれています。2行目のPackageプロパティは、プロジェクトを作成するときに指定したパッケージ名があります。3～4行目にandroid:versionCode、android:versionNameは、Appのバージョン情報を指定します。このプロパティは、今後Appをアップグレードして再配布する時に変更する部分です。

<application>タグは、Appのコンポーネントの構成を指定します。アンドロイドは4つのAppの重要なコンポーネントがあります。これは、Activity、Service、Broadcast receiver、そしてContent Providerです(コンポーネントの詳細説明はWebサイトを参照)。現在HelloGenieプロジェクトは、単に一つのActivityで構成されているので、一つの<activity>タグを含みます。

11行目に書かれた<application>タグのandroid:iconプロパティは、Androidデバイスでのアイコンを設定します。プロパティ値“@ drawable/ ic_launcher”は、Androidデバイスの解像度によって/ res/ drawable-hdpi/、...、/ res/ drawable-mdpiのディレクトリのic_laucherという名前の画像ファイルを示します(@はもう存在するリソースを参照する時に使用するキーワード)。12行目のandroid:labelプロパティは、Appの名前を指定し、“@ string/ app_name”は/ res / values/ strings.xmlに指定されたapp_nameの文字列を参照します。13行目は、Appのstyleを指定し、“@ style/ AppTheme”は/ res / values/ styles.xmlに指定されたAppThemeを参照します。

14～21行目では、プロジェクトに含まれているActivityを設定します。15行目のandroid:nameプロパティは、このActivityが実装されたソースコードの場所を指定します。このActivityが実装されたソースコードの位置はcom.dongburobot.hellogenie.MainActivityです。しかし<manifest>タグで指定されたpackageの名前は省略が可能で、プロパティ値をMainActivityに指定しても構いません。 16行目は、このActivityが外部に表示される名前を指定します。

17～20行目には、<intent-filter>があります。インテントフィルタは、このActivityに渡されるインテントをフィルタリングします。ここでインテントとは重要なコンポーネントを必要に応じて動作させるために必要な情報を持つオブジェクトであり、多数または特定のコアコンポーネントを活性化させます。
具体的にはランチャー(ホームスクリーン画面)でHelloGenie　Appを実行するときにandroid.inent.action.MAINのActionとandroid.intent.category.LAUNCHERのCategoryプロパティを持つインテントをHelloGenieに渡します。渡されたインテントはインテントフィルタによって2つのフィルタのプロパティを持っている　. MainActivityを実行します。

AndroidManifest.xmlファイルはAppの構成を説明する重要なファイルであり、一つのAppは、一つのAndroidManifest.xmlを必ず持っています。AndroidManifest.xmlを通してHelloGenie Appは<uses-sdk>タグを介してAndroid2.3.3で実行され、<application>タグを介して一つのActivityを含んでいます。現在唯一の<activity>であるMainActivityはインテントフィルタにより、ランチャー(ホームスクリーン画面)でユーザーがアイコンをクリックした時に発生するインテントを受信して実行されます。
次に、MainActivityを説明します。

(2) MainActivity.javaとactivity_main.xml

一つのActivityとは、一つの画面を意味します。ユーザーは、この画面(Activity)を通して、スマートフォンを操作します。簡単に考えると、一つのAppは多数の画面で構成されているので、複数のActivityの組み合わせです。

一つのAppでユーザーがそのAppを実行する時に、最初に表示される一つのActivityがあります。これは、AndroidManifest.xmlに定義されていますが、HelloGenieプロジェクトでは、これがMainActivityです。

MainActivityにもユーザーとの相互作用のためにはユーザインタフェースの定義部と処理部が存在します。ここでユーザーインターフェイスの定義部はactivity_main.xmlに対応し、処理部はMainActivity.javaに対応します。まず、MainActivity.javaファイルです。

```
1: package com.dongburobot.hellogenie;
2:
3: import android.os.Bundle;
4: import android.app.Activity;
5: import android.view.Menu;
6:
```

```
 7: public class MainActivity extends Activity {
 8:
 9:     @Override
10:     public void onCreate(Bundle savedInstanceState) {
11:         super.onCreate(savedInstanceState);
12:         setContentView(R.layout.activity_main);
13:     }
14:
15:     @Override
16:     public boolean onCreateOptionsMenu(Menu menu) {
17:         getMenuInflater().inflate(R.menu.activity_main, menu);
18:         return true;
19:     }
20: }
```

Listing2. MainActivity.java

すべてのAndroidのActivityはandroid.app.Acitivityクラスを受けて作成されます。MainActivityクラスもActivitiyクラスを受けて作成されました。Activityクラスを受けると、Activityのライフサイクルに応じて、様々なメソッドをオーバーライドできます。ちなみに、Activityのライフサイクルは非常に重要な内容で、必ず理解してください。詳細については、http://developer.android.com/guide/components/activities.htmlを参照して、詳細は後で説明します。

このうち10行目のonCreate()メソッドは、MainActivityが最初に作成される時に実行されるメソッドです。onCreate()の12行目には、次のような行があります。

```
setContentView(R.layout.activity_main);
```

この行のsetContentView()はactivity_main.xmlに定義されているユーザインタフェースを、現在のMainActivityに設定します。setContentView()に引数しているR.Layout.activity_mainは、res / layout/ activity_main.xmlを示しています。次はactivitiy_main.xmlです。

```
1: <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     xmlns:tools="http://schemas.android.com/tools"
3:     android:layout_width="match_parent"
4:     android:layout_height="match_parent" >
5:
6:     <TextView
7:         android:layout_width="wrap_content"
8:         android:layout_height="wrap_content"
9:         android:layout_centerHorizontal="true"
10:         android:layout_centerVertical="true"
11:         android:text="@string/hello_world" />
12: </RelativeLayout>
```

Listing3. activity_main.xml



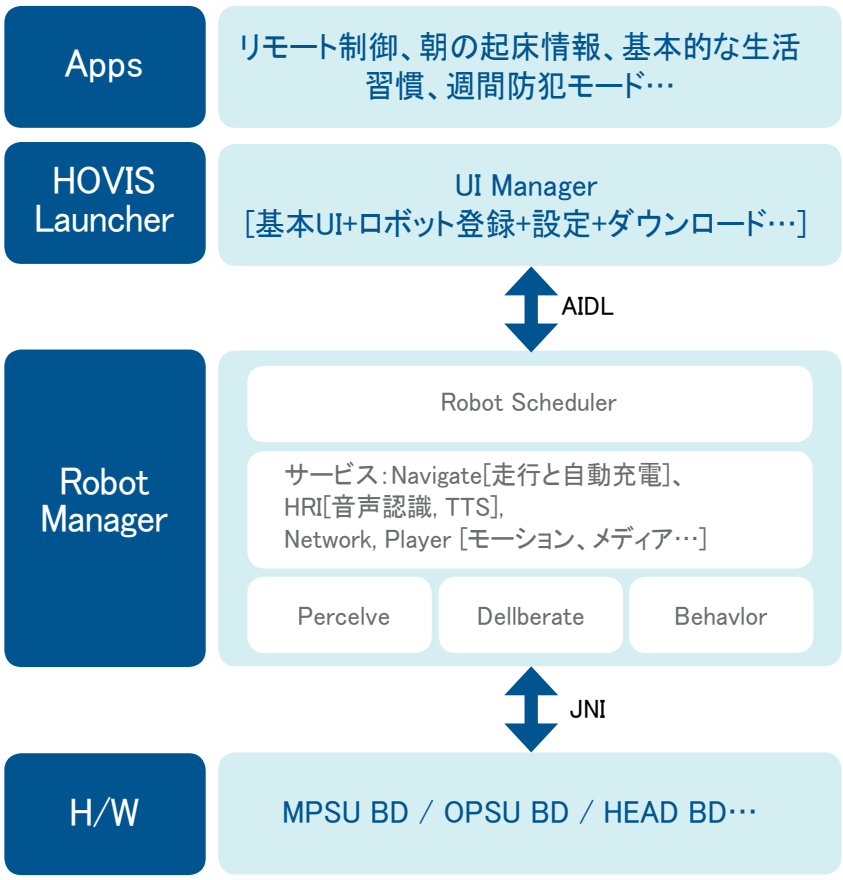
activity_main.xmlはRelativeLayoutとサブ要素であるTextViewで構成されています。 RelativeLayoutは、一つ以上のビューをビュー間の相対的な位置で配置するレイアウトです。TextViewは画面にテキストを出力するビューです。Activitiy_main.xmlで定義されたユーザインタフェースが実際の画面で、以下の図のように表現されます。

ただし、ここでは詳細な説明を省略します。詳細については、Activityのライフサイクルと一緒に後で例で説明します。XMLレイアウトファイルの詳細については、http://developer.android.com/guide/topics/ui/declaring-layout.htmlを参照してください。

04 HOVIS Genie ロボットのプログラミングを開始する

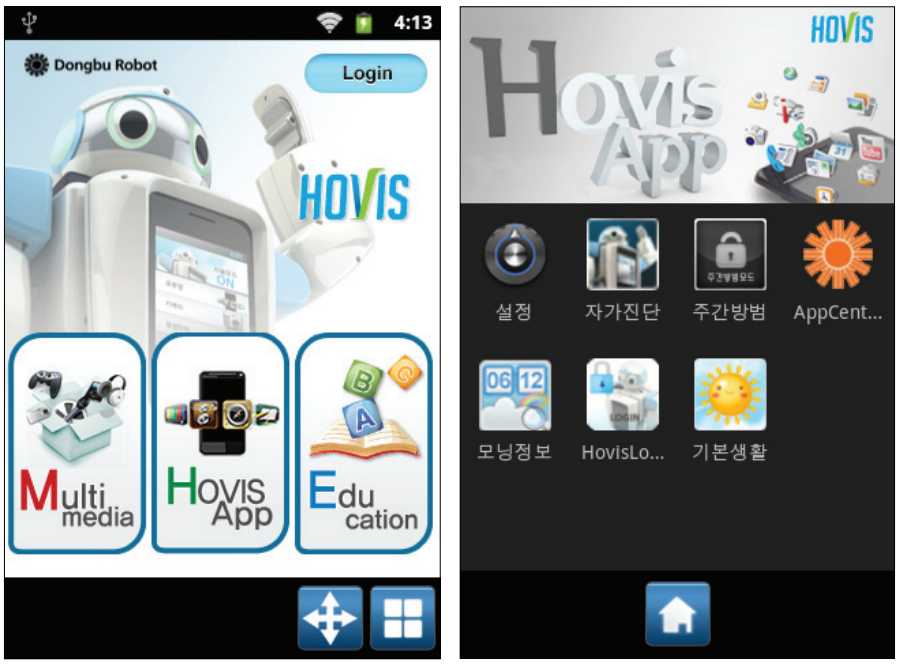
Chapter2ではアンドロイドのプログラミング環境を構成し、Chapter3では、最初のアンドロイドAppを実行して、全体的なアンドロイドプロジェクトの構成を説明しました。この章では、本格的にHovis Genieロボットのプログラミングを開始します。

まずHovis Genieロボットのプログラミングを開始する前に、Hovis Genieで動作するS/ W構造の理解が必要です。これにより、ロボットの動作原理を理解し、サービスを利用することで、ロボットAppの製作が可能です。[図4-1]は、Androidシステム上のHovis Genieの全体的なS/ Wの構造を示します。



[図 4-1] Hovis Genie S/W

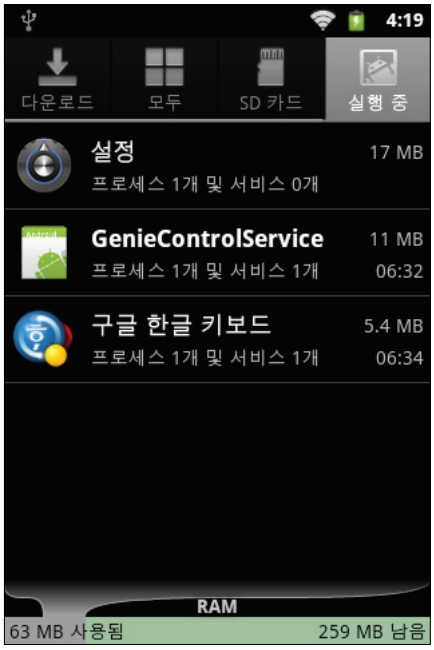
アンドロイドのシステム上でのHovisGenieのS/W構成は、アプリケーションであるApps、HOVIS Launcher、そしてRobot　Managerに分けます。このうち上位のアプリケーションレベルであるAppsとHovis Launcherは、それぞれのロボットAppとホームスクリーン画面で[図4-2]のように簡単に確認ができます。



[図 4-2] Hovis LauncherとロボットApp

一方、Robot　Managerは、システム内で実行されるので外部に現れませんが重要な役割を担当しており、Robot Schedulerとサービスがあります。まず、Robot Schedulerはロボットのイベント処理を担当し、ロボットの状態によって自律モードと充電モードなどの主要モードを決定して実行する部分です。次に、サービスはロボットのセンサとモーターなどのハードウェア全般を抽象化して、これを簡単に使用できるようにロボットサービスを提供する部分です。

特に、サービスは私たちが製作するロボットAppがロボットを簡単に制御できるようにAPI(Application Programing Interface)を提供します。Robot Managerの上位にあるHovis Launcher(ホームスクリーン画面)とロボットAppはRobot　Managerのサービスを使用して、ロボットに関連する作業を実行します。



[図 4-3] Robot Managerレベルのサービスは GenieControlService

このサービスはアンドロイドで[図4-3]のようにGenieControlServiceという名前で実行されます。 GenieControlServiceは[図4-1]のRobot Managerレベルを実装しており、Robot Schedulerとサービスが含まれています。重要なモジュールを含んでいるGenieControlServiceはロボットの制御のために必要なので、ロボット運用時には、常にバックグラウンドで動作する特徴があります。そして、予期せぬシステム障害に問題が発生しても自動的に修復される特徴があります。

要約すると、ロボットAppでロボット、H / Wを使用するためには、必ずGenieControlServiceが必要です。この章では、作成するAppでロボットサービスであるGenieControlServiceにアクセスする方法を説明します。この章の例はChapter5のサービスを利用したロボットプログラミングの例として使用されます。次の章で使用される例なので、正確な理解が必要です。それでは開始します。

4.1 GenieApiDemo プロジェクトの作成

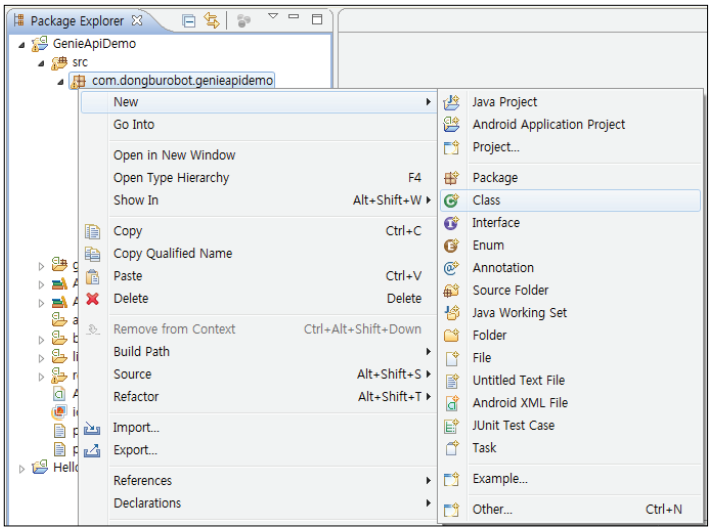
この章の例のための新しいプロジェクトを作成します。 Eclipseで、File> New> Android Application Projectウィンドウを実行して、次のように入力して、プロジェクトを作成します。(詳細はChapter3を参照してください。)

- Application Name : GenieApiDemo
- Project Name : GenieApiDemo
- Package Name : com.dongburobot.genieapidemo
- Build SDK : Android 2.3.3 (API 10)
- Minimum Required SDK : API 10: Android 2.3.3 (Gingerbread)

GenieApiDemoプロジェクトが作成されると、GenieApiDemoApplication.javaとBaseActivity.javaを追加します。そしてGenieControlServiceを使用するためにHovisService.jarファイルをlibsフォルダに追加します。

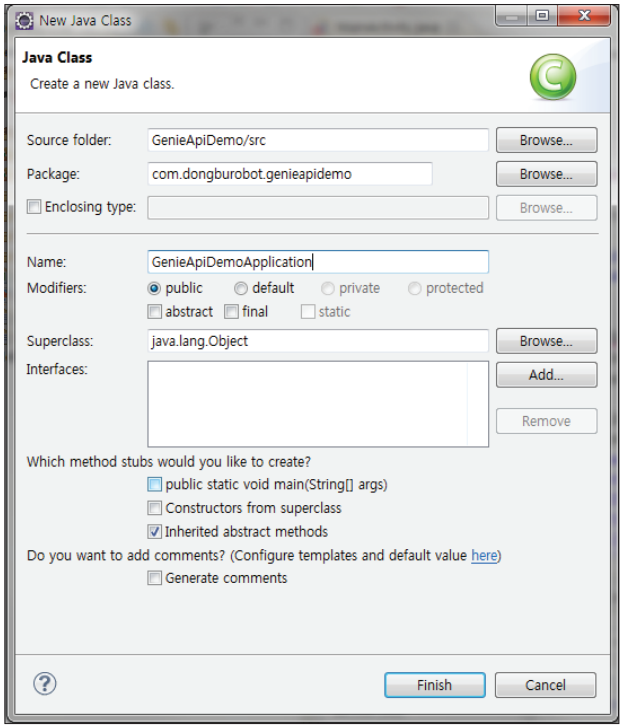
01

src/com.dongburobot.genieapiを右クリックした後 New > Classをクリック



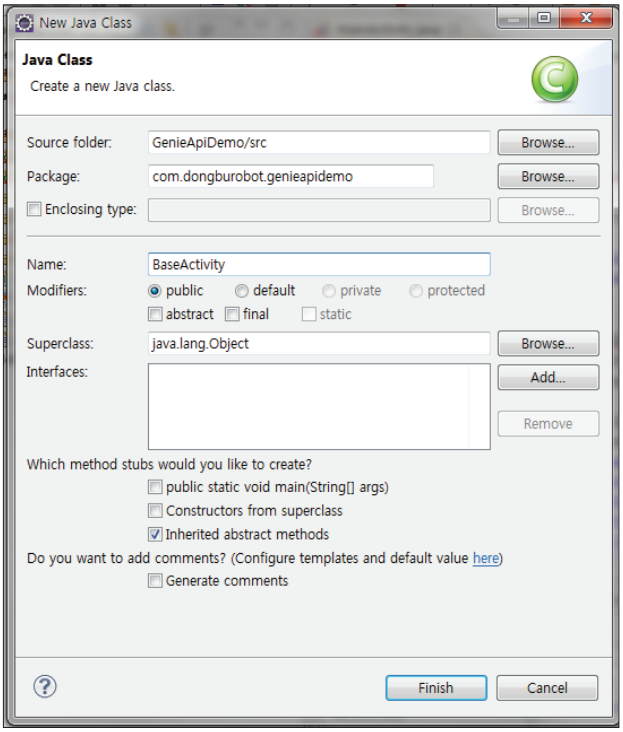
02

GenieApiDemoApplication.javaを追加します。



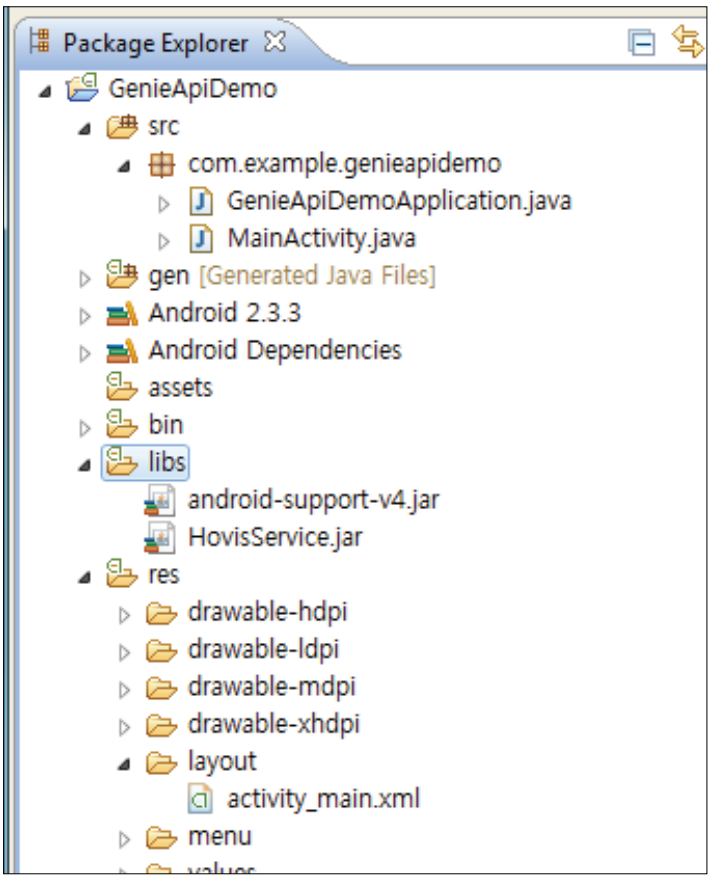
03

同じ方法でBaseActivity.javaを追加します。



04

HovisService.jarファイルをドラッグしてlibsフォルダに追加します。HovisService.jarファイルはDongbu Robotホームページからダウンロードします。
(http://www.dongburobot.com/jsp/cms/view.jsp?code=100122)



ここでlibsフォルダのHovisService.jarはGenieControlServiceに含まれているロボットに関連するAPIの宣言部を一つのファイルにまとめて配布されたファイルです。このファイルがプロジェクトに正しく含まれていない場合、プロジェクトが正常に作成されませんのでご注意ください。それでは本格的にGenieControlServiceとバインディングを介してロボットAppの開発を開始します。

※注意 - EclipseにインストールされたADT(Android Development Tools)のバージョンによって、HovisService.jarファイルを追加する方法が違う場合があります。最新バージョンのEclipseとADT Pluginを使用してください。

4.2 GenieApiDemoApplicationのクラス

GenieApiDemoApplicationクラスは、Applicationクラスを受けて作成します。Applicationクラスはandroid.app.Applicationに定義されているクラスでアンドロイドAppを抽象的に表現するクラスです。Applicationクラスは、Appを実行する間、全体の状態の維持が必要な場合に使用されます。つまり、App全体で使用できるグローバル変数を宣言するときに便利です。GenieApiDemoプロジェクトでGenieApiDemoApplicationクラスの役割は、Appが実行されている間にGenieControlServiceで提供するサービス(HovisGenieService)にバインドしてサービスを利用できるようにグローバル変数を宣言することです。

GenieApiDemoApplication クラスのソースコードは次のとおりです。

```
1: package com.dongburobot.genieapidemo;
 2:
 3: import android.app.ActivityManager;
 4: import android.app.ActivityManager.RunningServiceInfo;
 5: import android.app.Application;
 6: import android.content.ComponentName;
 7: import android.content.Intent;
 8: import android.content.ServiceConnection;
 9: import android.os.IBinder;
10: import android.util.Log;
11:
12: import com.dongburobot.HovisGenieServiceInterfaces.IHovisGenieService;
13:
14: public class GenieApiDemoApplication extends Application {
15:
16:     // HovisGenieService サービスバインディング
17:     public IHovisGenieService mBinder = null;
18:
19:     private boolean mIsBinded;
20:     private Intent mIntent;
```

```
21:
22:    public ServiceConnection mConnection = new ServiceConnection() {
23:
24:        public void onServiceConnected(ComponentName name, IBinder service) {
25:            mBinder = IHovisGenieService.Stub.asInterface(service);
26:        }
27:
28:        public void onServiceDisconnected(ComponentName name) {
29:            mBinder = null;
30:        }
31:    };
32:
33:    protected boolean isServiceRunning() {
34:        ActivityManager manager = (ActivityManager)getSystemService(ACTIVITY_SERVICE);
35:
36:        for (RunningServiceInfo serviceInfo: manager.getRunningServices(Integer.MAX_VALUE)) {
37:            if (serviceInfo.service.getClassName().equals("com.dongburobot.geniecontrol.HovisGenieService")) {
38:                return true;
39:            }
40:        }
40:          }
41:          return false;
42:
43:    }
44:
45:    protected boolean connectService() {
46:          if ( isServiceRunning() ) {
47:              mIntent = new Intent(IHovisGenieService.class.getName());
48:              mIsBinded = bindService(mIntent, mConnection, BIND_AUTO_CREATE);
49:              return mIsBinded;
50:          }
51:
52:          return false;
53:    }
54:
55:    protected void disconnectService() {
56:          if (mIsBinded) {
```

```
57:             if (mConnection != null) {
58:                 unbindService(mConnection);
59:                 mConnection = null;
60:             }
61:         }
62:
63:         mIntent = null;
64:     }
65:
66:     // Application 開始と終了時にサービスバインド
67:     @Override
68:     public void onCreate() {
69:         super.onCreate();
70:
71:         if ( !connectService() )
72:             Log.i("service", "not connected");
73:     }
74:
75:     @Override
76:     public void onTerminate() {
77:         super.onTerminate();
78:
79:         disconnectService();
80:     }
81: }
```

[Listing. 4-1] GenieApiDemoApplication.java

今からGenieApiDemoApplicationクラスの作成を開始します。
GenieApiDemoApplication.javaファイルを開き、次のようにGenieApiDemoApplicationクラスがApplicationクラスを受けるように変更します。

public class GenieApiDemoApplication extends Application { … }

‘extends Application’ を追加すると、GenieApiDemoApplicationに赤の下線が追加されます。この時、下線の上でCtrl + Shift + Oを押すと、含める必要があるパッケージとOverrideする必要があるメソッドが自動的に追加されます。

TIP. Eclipseで追加されていないパッケージの追加は、"Ctrl + Shift + O"です。
さらに、ソースの作成時にコードを自動補完する機能は、"Ctrl + Space"です。

再定義するメソッドは、onCreate()とonTerminate()です。ここでonCreate()は、Appを初めて開始する時に呼び出されるメソッドです。GenieApiDemo Appを初めて開始する時GenieContorlServiceで提供するHovisGenieServiceに接続するメソッドであるconnectService()を実行するので、これを追加します。逆に、Appを終了する時には、onTerminate()が呼び出されます。Appを終了する時には、サービスの接続を終了するので、disconnectService()メソッドを追加します。

```
…
import android.app.Application;
…
public class GenieApiDemoApplication extends Application {
…
@Override
    public void onCreate() {
        super.onCreate();
        a
        if ( !connectService() )
            Log.i("service", "not connected");
    }

    @Override
    public void onTerminate() {
        super.onTerminate();

        disconnectService();
    }
}
```

connectService()とdisconnectService()は、サービスを接続して終了する役割をします。このメソッドを定義する前に、サービスインターフェイスオブジェクトを宣言します。

```
import com.dongburobot.HovisGenieServiceInterfaces.IHovisGenieService;


public class GenieApiDemoApplication extends Application {
public IHovisGenieService mBinder = null; // サービスインターフェースオブジェクト…
    …
    …
}
```

mBinderはIHovisGenieServiceのオブジェクトです。今後すべてのロボットサービスの利用はmBinderを介して行われるため、重要なオブジェクトです。ここでIHovisGenieServiceクラスはGenieControlService内部のサービスであるHovisGenieServiceにアクセスするためのインタフェースです。IHovisGenieServiceはHovisService.jarに定義されています。HovisService.jarファイルに定義されたIHovisGenieServiceクラスのパッケージのネームスペースはcom.dongburobot.HovisGenieServiceInterfaces.IHovisGenieServiceです。したがって、IHovisGenieServiceを使用するために、これをimportします。

> 詳細なIHovisGenieServiceはAIDL(Android Interface Definition Language)で書かれたIPC(Inter Process Communication)を担当しています。ここでは、サービスにバインドして、ロボットサービスを使用することに重点を置くので詳細を省略しています。IPCに関するAIDLに興味がある方はAndroid Developerホームページを参照してください。
> – http://developer.android.com/guide/components/aidl.html

続いて[Listing. 4-1]を参照して、connectService()、disconnectService()、isServiceRunning()を追加します。そして19～20行の変数とオブジェクトも追加します。

最後にGenieApiDemoApplicationクラスはアンドロイドベースのApplicationクラスを受けて作成したので、AndroidManifest.xmlにこれを明示します。次のように11行目にandroid:name=“. GenieApiDemoApplication”を追加します。

```
1: <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2:     package="com.dongburobot.genieapidemo"
3:     android:versionCode="1"
4:     android:versionName="1.0" >
5:
6:     <uses-sdk
7:         android:minSdkVersion="10"
8:         android:targetSdkVersion="10" />
9:
10:    <application
11:        android:name=".GenieApiDemoApplication"
12:        android:icon="@drawable/ic_launcher"
13:        android:label="@string/appa_name"
14:        android:theme="@style/AppTheme" >
15:        <activity
16:            android:name=".MainActivity"
17:            android:label="@string/title_activity_main" >
18:            <intent-filter>
19:                <action android:name="android.intent.action.MAIN" />
20:                <category android:name="android.intent.category.LAUNCHER" />
21:            </intent-filter>
22:        </activity>
23:    </application>
24:
25: </manifest>
```

[Listing 4-2] AndroidManifest.xml

(1) ロボットサービスの接続

私たちのAppとロボットサービスは、Appが初めて実行される時に接続します。　Appを開始するとGenieApiDemoApplicationのonCreate()を呼び出します。そしてonCreate()の内部でconnectService()を呼び出してサービスにバインドします。connectService()によって接続されたロボットサービスは、Appが終了するまで、その接続が維持されます。

connectService()は、その内部でisServiceRunning()を介してサービスがバックグラウンドで動作することを確認して、bindService()を介してロボットサービスにバインドします。そしてバインドの結果に応じて、trueまたはfalseを返還します。

```
@Override
public void onCreate() {
    super.onCreate();

    if ( !connectService() )
        Log.i("service", "not connected");
}

protected boolean connectService() {
        if ( isServiceRunning() ) {
            mIntent = new Intent(IHovisGenieService.class.getName());
            mIsBinded = bindService(mIntent, mConnection, BIND_AUTO_CREATE);
            return mIsBinded;
        }

        return false;
    }
}
```

まずconnectService()が実行されると33行目のisServiceRunning()を呼び出して、現在のMIDでロボットサービス(GenieServiceControl)が実行されているかを確認します。
isServiceRunning()は、現在のロボットサービスがバックグラウンドで動作する時にtrueを返還し、それ以外の場合はfalseを返還します。
もし現在のバックグラウンドでロボットサービスが動作している場合、bindService()を呼び出して、実際にロボットサービスにバインドします。binsService()はandroid.contentに定義されており、元は次のとおりです。

```
public abstract boolean bindService(Intent service, ServiceConnection conn, int flags)
```

パラメータ

service	サービスの接続先のターゲットを定義します。インテントは、サービスに定義されたインテントフィルタと一致するコンポーネントネームなどを指定します。
conn	サービスが開始または停止された時の情報を受けるオブジェクトを指定します。
flags	サービスバインディングオプション, BIND_AUTO_CREATE, BIND_DEBUG_UNBIND, BIND_NOT_FOREGROUND, BIND_ABOVE_CLIENT, BIND_ALLOW_OOM_MANAGEMENT, or BIND_WAIVE_PRIORITY.

返還の値

サービスバインディングに成功するとtrueを返還し、それ以外の場合はfalseを返還します。

```
mIsBinded = bindService(mIntent, mConnection, BIND_AUTO_CREATE);
```

bindService()の最初の引数は、インテントです。インテントは一つのAppコンポーネントが他のAppコンポーネントを実行するために送るオブジェクトです。ここでは、AppがGenieControlServiceにbindService()を介してインテントを送信することによって、GenieControlServiceで提供するロボットサービスを使用するというメッセージを送信します。このインテントはconnectService()の中に次のように宣言します。

```
mIntent = new Intent(IHovisGenieService.class.getName());
```

GenieContorlServiceのAndroidManifest.xml のうちに

```
<intent-filter>
<action android:name="com.dongburobot.HovisGenieServiceInterfaces.IHovisGenieService"/>
</intent-filter>
```

Intentのコンストラクタの引数であるIHovisGenieService.class.getName()は、ロボットサービスのインタフェースのクラス名com.dongburobot.HovisGenieServiceInterfaces.IHovisGenieServiceを返します。そしてmIntentは、このクラス名を持つIntentのオブジェクトが割り当てられます。mIntentがロボットインタフェースのクラス名を持つ理由は、ロボットサービスに定義されたインテントフィルタが次のようになっているからです。

bindService()の二番目の引数は、サービスが開始または終了する時に、それに関する情報を受信するServiceConnectionクラスのオブジェクトです。ServiceConnectionのオブジェクトであるmConnectionは22～31行目で宣言しました。

```
public ServiceConnection mConnection = new ServiceConnection() {
        public void onServiceConnected(ComponentName name, IBinder service) {
            mBinder = IHovisGenieService.Stub.asInterface(service);
        }

        public void onServiceDisconnected(ComponentName name) {
            mBinder = null;
        }
    };
```

onServiceConnected()は、サービスがバインドされた時にコールバック(システムで自動的にメソッドを呼び出す)されmBinderに接続されたサービスのインターフェイスを割り当てます。一方で、onServiceDisconnected()は、サービスが中断された時にコールバックされ、サービスを無効にしないため、mBinderはnullに設定します。つまり、mConnectionはロボットサービスの接続状態に応じてmBinderを管理する役割をします。

bindService()の最後から三番目の引数は、サービスバインディングオプションです。BIND_AUTO_CREATEはバインディングされている限りサービスを自動的に生成します。その他の多くのオプションがありますが、ほとんど使いません。

bindService()を呼び出すと、サービス接続の状態に応じてServiceConnectionクラスのオブジェクトmConnectionに定義された2つの再定義メソッドによってIHovisGenieServiceのオブジェクトmBinderにインターフェイスを割り当てます。そしてconnectService()がすべて実行された後Appがサービスを使用する準備が終わります。

(2) ロボットサービスの接続の解除

サービスの解除はGenieApiDemoApplicationクラスのonTerminate()内のdisconnectService()の呼び出しを介して行われます。　onTerminate()は、Appが終了する時に自動的に呼び出されるので、Appを終了する時にサービスも一緒に終了します。

```
@Override
public void onTerminate() {
    super.onTerminate();

    disconnectService();
}
```

```
protected void disconnectService() {
    if (mIsBinded) {
        if (mConnection != null) {
            unbindService(mConnection);
            mConnection = null;
        }
    }

    mIntent = null;
}
```

disconnectService()が呼び出されると、mIsBindedを介して現在サービスがバインドされた状態であることを確認します。サービスがバインドされた状態であると、mConnectionがnullの状態ではないことを確認してunbindService()を呼び出します。unbindServiceの元は次のとおりです。

```
public abstract void unbindService(ServiceConnection conn)
```

パラメータ

conn	サービス接続する時にbindService()に適用されるServiceConnectionのオブジェクト

```
unbindService(mConnection);
```

unbindService()は、現在接続されているサービスを無効にします。unbindService()の引数は、サービスを接続する時のbindService()の二番目の引数として使用していたServiceConnectionのオブジェクトです。

unbindService()を呼び出すと、サービスの接続が解除されるとともに、引数として渡されたmConnectionに再定義したonServiceDisconnected()を呼び出します。そしてmBinderにnullを設定して、サービス接続の解除を完了します。

```
public ServiceConnection mConnection = new ServiceConnection() {
        public void onServiceConnected(ComponentName name, IBinder service) {
            mBinder = IHovisGenieService.Stub.asInterface(service);
        }

        public void onServiceDisconnected(ComponentName name) {
            mBinder = null;
        }
    };
```

サービス接続の解除を正確にして、ロボットシステムの異常動作を防止する効果もあります。そのためサービスの接続をすると、必ず接続を解除する習慣が重要です。ファイル入出力時にファイルを開くと、閉じるというプログラミングの習慣が重要なように、ロボットサービスを接続する時にも接続して、必ず接続を解除する習慣を身につけるようにします。

4.3 ロボットシステム用BaseActivityの作成

アンドロイドデバイスの1つの画面は、一つのActivityで構成されます。Appを製作する時に一つの画面を作るというのは一つのActivityを作成することです。したがって、通常のスマートフォンAppの一つの画面はandroid.app.Activityに定義されたActivityクラスを受けて制作します。

ロボットAppも一つの画面はActivityクラスを受けて作成できます。しかし、他のスマートフォンAppとは異なり、次の2つのことをActivity製作をする前に考慮する必要があります。

考慮 事項

1。 **ロボットが1台なので、制御の同期が必要**

2。 **App画面から直接On / Offの制御が必要**

まず、ロボットが1台なので、制御の同期が必要です。制御の同期化が保証されないと、ロボットの正確な動作を保証できません。しかし、アンドロイドのActivityの管理は制御の同期を保証していない可能性があります。そのリスクは、一つの画面が消えても、その画面を表示するActivityが終了していないことにあります。

たとえば、ロボットの前進Activityがあり、このActivityがロボットに前進命令を下しているうちにロボットのバッテリーが不足して充電モードに入り、画面には充電Activityが出ていると考えてみましょう。この時にロボットの前進Activityが終了されず、スタックに積もって継続ロボットの前進命令を下した場合、充電Activityから実行すべき自動充電走行に影響を与えます。
もちろん、このような状況はほとんど発生しません。また、開発者がこのボトルネックの状況を事前に予想し、それに対処できるようにコードを記述することもできます。しかし、最も良い方法は、このような状況がまったく発生しないように、事前に遮断することです。

作成するロボットAppでは一つのActivityが画面から消えるとそのActvitityが完全に終了するようにします。Activityが画面から消える時に必ず呼び出されるメソッドはonPause()です。したがって、onPause()でActivityが完全に終了するようにします。

```
1: package com.dongburobot.genieapidemo;
2:
3: import android.app.Activity;
4: import android.os.Bundle;
5: import android.view.Window;
6: import android.view.WindowManager;
7:
8: public class BaseActivity extends Activity {
9:
10:     protected GenieApiDemoApplication myRobotApp;
11:
12:     @Override
13:     protected void onCreate(Bundle savedInstanceState) {
14:         super.onCreate(savedInstanceState);
15:
16:         // 画面の設定
17:         this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
18:                 WindowManager.LayoutParams.FLAG_FULLSCREEN);
19:         this.getWindow().addFlags(WindowManager.LayoutParams.FLAG_SHOW_WHEN_LOCKED
20:                 | WindowManager.LayoutParams.FLAG_DISMISS_KEYGUARD
21:                 | WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON
22:                 | WindowManager.LayoutParams.FLAG_TURN_SCREEN_ON);
23:         this.requestWindowFeature(Window.FEATURE_NO_TITLE);
24:
```

```
25:         myRobotApp = (GenieApiDemoApplication)this.getApplicationContext();
26:     }
27:
28:     @Override
29:     protected void onPause() {
30:         super.onPause();
31:
32:         this.finish();
33:     }
34: }
```

[Listing 4-2] BaseActivity.java

BaseActivityの13行目onCreate()はActivityが作成されるときに呼び出されます。onCreate()でActivityが開始した時に必要な部分を作成します。現在必要なのは、画面を設定するコードです。これは、考慮事項2に該当します。

```
// 画面の設定(考慮事項2)
        this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                WindowManager.LayoutParams.FLAG_FULLSCREEN);
        this.getWindow().addFlags(WindowManager.LayoutParams.FLAG_SHOW_WHEN_LOCKED
                | WindowManager.LayoutParams.FLAG_DISMISS_KEYGUARD
                | WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON
                | WindowManager.LayoutParams.FLAG_TURN_SCREEN_ON);
        this.requestWindowFeature(Window.FEATURE_NO_TITLE);
```

17～23行目のコードは、画面を設定するコードです。17～18行目のコードは、現在の画面をFULL SCREENで使用するために設定します。19～22行目のコードは、画面にロックかかっても画面を見せてくれて、アンドロイドのキー入力画面を表示しません。そして、スクリーンを常にオンにして、点灯している状態を維持するflagを使用して画面を設定します。23行目のコードは、画面のタイトルバーを取り除きます。他にも、画面の設定ができる多くのflagがあります。詳細については、アンドロイドの開発者サイトを参照します。

29行目onPause()はActivityが画面から消える時に呼び出されます。前に説明したように、ロボットAppは、ロボット制御の同期化のためにスタックにActivityを残しません。したがって、32行目でfinish()を利用してActivityが完全に終了するようにします。これは、考慮事項の1に該当します。

```
@Override
protected void onPause() {
    super.onPause();
    // 考慮事項 1 に該当する
    this.finish();
```

最後にGenieApplicationDemoApplicationのオブジェクトが10行目で宣言されています。 myRobotAppはgetApplicationContext()によって返された現在のプロセスのグローバルApplicationオブジェクトを割り当てます。onCreate()の内部25行目のコードはBaseActivityを受けて作成されたすべてのActivityからGenieApiDemoApplicationクラスが含まれているオブジェクトと変数をグローバルに使用可能です。これにより、ロボットサービスをすべてActivityで使用できるベースを設定します。

```
protected GenieApiDemoApplication myRobotApp;
…
myRobotApp = (GenieApiDemoApplication)this.getApplicationContext();
```

これによりロボットプログラミングを開始する基本的な設定を完了しました。この章では、ロボットを制御するために、ロボットサービスを利用する方法について説明しました。次の章では、ロボットサービスで提供される関数を説明します。

05 HOVIS Genie API (Application Programing Interface)

Hovis　Genieを制御するためのサービスは、MIDでバックグラウンドで動作しているGenieContorl-Serviceによって行われます。ユーザーはGenieControlServiceにバインディングをすることで、ロボットのサービスを利用できます。

前の章では、ロボットサービスを使用するために、サービスとしてバインドする方法について説明しました。この章では、本格的なロボットプログラミングする前にHovis Genie APIで提供される関数を説明します。Hovie Genie APIが提供する機能は次のとおりです。

駆動部関数

駆動部の関数は、Hovis　Genieのオムニホイール駆動部に関する関数の集合です。この関数は、駆動部の車輪の設定と動きを主に設定するのに使用されます。実際のロボット駆動はナビゲーション関数を介して行われます。

ナビゲーション関数

ナビゲーション関数は、Hovis　Genieを実際に動かす役割を担当しています。この関数は、オムニホイールの駆動方式を抽象化して、x、y、thetaの座標に移動ができる役割をします。

センサ関数

センサに関する関数は、Hovie　Genieに付着した各種センサ(距離センサ、下感知センサ、タッチセンサなど)を制御するためのインターフェイスを提供します。

TTS(Text-to-Speech)関数

TTS 関数は、文字列の入力をロボットの音声に変換する関数です。

サウンド(効果音)関数

サウンド関数は、ロボットの状況に応じて様々な効果音を発生する関数です。

マルチメディア(オーディオおよびビデオ)関数

ロボットでオーディオおよびビデオファイル(MP3、MP4、AVI、...)を再生、停止するための機能を提供します。

モーションとサーボモーターに関する関数

モーションとサーボモーターに関する関数は、ロボットのモーションの機能を提供します。

5.1 駆動部関数

dmel_drive_servo_on

```
boolean dmel_drive_servo_on(ComponentName cn)
                    throws android.os.RemoteException
```

ロボット駆動部(DRS-0102)サーボモーターの3つを"Servo ON"にする。
Parameters:
cn – Componentの名前
Returns:
成功するとtrue、失敗するとfalseを返す
Throws:
android.os.RemoteException

dmel_drive_set_movable

```
void dmel_drive_set_movable(ComponentName cn,
                    boolean flag)
                     throws android.os.RemoteException
```

ロボット駆動部の駆動が可能かどうかを設定します。
Parameters:
cn – Componentの名前
flag – 駆動可能設定はtrue、不可設定false
Throws:
android.os.RemoteException

dmel_drive_get_movable

```
boolean dmel_drive_get_movable(ComponentName cn)
                    throws android.os.RemoteException
```

ロボット駆動部が駆動するかどうかを返します。
Parameters:
cn – Componentの名前
Returns:
ロボット駆動部の駆動が可能な時true、不可の時falseを返す
Throws:
android.os.RemoteException

dmel_robot_set_horizontal_velocity

```
boolean dmel_robot_set_horizontal_velocity(ComponentName cn,
                                           double vel,
                                           double ang)
                                    throws android.os.RemoteException
```

ロボット駆動部の平行移動速度を設定する。
Parameters:
cn – Componentの名前
vel – 平行移動速度(単位:m/ s)
ang – 平行にで補正された角度(単位:radであり、値の+はCCW、 – はCW)
Returns:
Throws:
android.os.RemoteException

dmel_robot_set_rotation_velocity

```
boolean dmel_robot_set_rotation_velocity(ComponentName cn,
                                         double vel)
                                    throws android.os.RemoteException
```

ロボット駆動部の角速度を設定する。
Parameters:
cn – Componentの名前
vel – 角速度(rad/ s)
Returns:
成功した場合はtrue、失敗した場合falseを返す
Throws:
android.os.RemoteException

dmel_robot_set_velocity

```
boolean dmel_robot_set_velocity(ComponentName cn,
                                double lin,
                                double ang,
                                double horizontal_ang,
                                boolean flag)
                                throws android.os.RemoteException
```

ロボットの駆動部を駆動する。
Parameters:
cn – Componentの名前
lin – 線速度(m/ s)
ang – 角速度(rad/ s)+はCCW、 – はCW
horizontal_ang – 指定された値だけ(+はCCW、 – はCW)補正された角度に平行移動
flag – trueの場合は平行移動しhorizontal_ang値によって移動、falseの場合は回転移動。ang値によって移動
fThrows:
android.os.RemoteException

dmel_param_set_max_lin_velocity

```
boolean dmel_param_set_max_lin_velocity(ComponentName cn,
                                        double vel)
                                        throws android.os.RemoteException
```

ロボット駆動部の最大線速度を設定する。
Parameters:
cn – Componentの名前
vel – 最大線速度(m/ s)
Returns:
設定に成功した場合はtrue、失敗した場合false
Throws:
android.os.RemoteExceptio

dmel_param_get_max_lin_velocity

```
double dmel_param_get_max_lin_velocity(ComponentName cn)
                                        throws android.os.RemoteException
```

ロボット駆動部の最大角速度を返す。
Parameters:
cn – Componentの名前
Returns:
最大線速度(m/ s)
Throws:
android.os.RemoteException

dmel_param_set_max_ang_velocity

```
boolean dmel_param_set_max_ang_velocity(ComponentName cn,
                                        double vel)
                                          throws android.os.RemoteException
```

ロボット駆動部の最大角速度を設定する。
Parameters:
cn – Componentの名前
vel – 最大角速度(rad/ s)
Returns:
設定に成功した場合はtrue、失敗した場合false
Throws:
android.os.RemoteException

dmel_param_get_max_ang_velocity

```
double dmel_param_get_max_ang_velocity(ComponentName cn)
                                    throws android.os.RemoteException
```

ロボット駆動部の最大角速度を返す。
Parameters:
cn – Componentの名前
Returns:
最大角速度(rad/ s)
Throws:
android.os.RemoteException
android.os.RemoteException

5.2 ナビゲーション関数

dmel_robot_set_pose

```
boolean dmel_robot_set_pose(ComponentName cn,
                            double x,
                            double y,
                            double theta)
                           throws android.os.RemoteException
```

ロボットの現在位置をx、y、thetaで設定します。ロボットの現在位置を(0、0、0)に設定すると、現在の位置が基準点となる。
Parameters:
cn – Componentの名前
x – (meter)
y – (meter)
theta – (Radian–PI～+ PI)
Returns:
成功した場合はtrue、失敗した場合falseを返す
Throws:
android.os.RemoteException

dmel_robot_get_pose

```
double[] dmel_robot_get_pose(ComponentName cn)
                         throws android.os.RemoteException
```

開始点を基準にロボットの現在位置を返す。
Parameters:
cn – Componentの名前
Returns:
開始点基準double[0]～double[2]にそれぞれx、y、theta(Radian–PI～+ PI)値を返す
Throws:
android.os.RemoteException

dmel_navigate_move_pose

```
void dmel_navigate_move_pose(ComponentName cn,
                             double x,
                             double y,
                             double theta)
                             throws android.os.RemoteException
```

ロボットの開始点を基準に(x、y、theta)のポイントに移動する。
Parameters:
cn – Componentの名前
x – 開始点の基準x(meter)
y – 開始点の基準y(meter)
theta – 開始点からのロボットHeadingを基準にしたロボットの方向(Radian-PI～+ PI)
Throws:
android.os.RemoteException

dmel_navigate_prepare

```
void dmel_navigate_prepare(ComponentName cn)
                           throws android.os.RemoteException
```

ロボットのナビゲーションのための初期化を実行します。初期化時に、ロボットの現在位置を(0、0、0)に設定し、現在のHeadingを0度に設定する。
Parameters:
cn – Component 名前
Throws:
android.os.RemoteException

dmel_navigate_add_goal_pose

```
boolean dmel_navigate_add_goal_pose(ComponentName cn,
                                    double x,
                                    double y,
                                    double theta,
                                    boolean rflag,
                                    int time)
                                    throws android.os.RemoteException
```

ロボットが訪問するポイントを追加します。ロボットに設定された開始点(0、0、0)を基準に訪問するポイントを決定する。(dmel_navigate_prepare()を参照)ロボットは設定された訪問ポイントをdmel_navigate_start()を介して訪問を開始する。
Parameters:
cn – Componentの名前
x – 開始点の基準x(meter)
y – 開始点の基準y(meter)
theta – 開始点からのロボットのHeadingを基準にしたロボットの方向(Radian-PI～+ PI)
rflag – trueロボットが訪問ポイントに到着する時にtheta度方向を眺める
time – ロボットが訪問ポイントに到着する時の待機時間(sec)Returns:
Throws:
android.os.RemoteException

dmel_navigate_start

boolean dmel_navigate_start(ComponentName cn)
throws android.os.RemoteException
ロボットのナビゲーションを実行します。ロボットのナビゲーションは事前に設定されたgoal_pose点ずつ訪問する。
Parameters:
cn – Componentの名前
Returns:
成功した場合はtrue、失敗した場合falseを返す
Throws:
android.os.RemoteException

dmel_navigate_stop

boolean dmel_navigate_stop(ComponentName cn)
throws android.os.RemoteException
ロボットのナビゲーションの実行を中断する。
Parameters:
cn – Componentの名前
Returns:
成功した場合はtrue、失敗した場合falseを返す
Throws:
android.os.RemoteException

5.3 センサ関数

dmel_robot_get_obs

double[] dmel_robot_get_obs(ComponentName cn)
throws android.os.RemoteException
前方感知PSDセンサの値を返します。前方から時計回りに0、1、2、3、4
Parameters:
cn – Componentの名前
Returns:
前方感知センサ5つの値をdouble[0]～double[4]に返す
Throws:
android.os.RemoteException

dmel_robot_get_cliff

```
double[] dmel_robot_get_cliff(ComponentName cn)
                        throws android.os.RemoteException
```

床感知PSDセンサの値を返す。前方から時計回りに0、1、2
Parameters:
cn – Componentの名前
Returns:
床感知センサの3つの値をdouble[0]～double[2]に返す
Throws:
android.os.RemoteException

dmel_hri_get_head_touch_info

```
boolean dmel_hri_get_head_touch_info(ComponentName cn)
                              throws android.os.RemoteException
```

頭をタッチされているかどうかを返す。
Parameters:
cn – Componentの名前
Returns:
頭タッチしている時true、していなければfalseを返す
Throws:
android.os.RemoteException

dmel_hri_get_hand_touch_info

```
boolean[] dmel_hri_get_hand_touch_info(ComponentName cn)
                              throws android.os.RemoteException
```

両手をタッチされているかどうかを返す。
Parameters:
cn – Componentの名前
Returns:
タッチされている時true、していなければfalseを返す。boolean[2](boolean[0]:左手のタッチ、boolean[1]:右手のタッチ)
Throws:
android.os.RemoteException

5.4 TTS(Text-to-Speech) 関数

dmel_tts_speak

```
void dmel_tts_speak(ComponentName cn,
                    java.lang.String text)
                    throws android.os.RemoteException
```

TTS(Text-to-Speech)をPlayする。
Parameters:
cn – Componentの名前
text – 音声に変換する文字列String ex)"こんにちは。ホビスジニです。"
Throws:
android.os.RemoteException

dmel_tts_stop

```
void dmel_tts_stop(ComponentName cn)
                   throws android.os.RemoteException
```

TTS(Text-to-Speech)を停止する 。
Parameters:
cn – Component 名前
Throws:
android.os.RemoteException

5.5 サウンド(効果音)関数

dmel_sound_play_intro

```
void dmel_sound_play_intro(ComponentName cn)
                           throws android.os.RemoteException
```

効果音を再生する。(Intro効果音)
Parameters:
cn – Componentの名前
Throws:
android.os.RemoteException

dmel_sound_play_chimes

void dmel_sound_play_chimes(ComponentName cn)
throws android.os.RemoteException
効果音を再生する。(Chimesの効果音)
Parameters:
cn – Component 名前
Throws:
android.os.RemoteException

dmel_sound_play_dingdong

void dmel_sound_play_dingdong(ComponentName cn)
throws android.os.RemoteException
効果音を再生する。(ディンドンの効果音)
Parameters:
cn – Component 名前
Throws:
android.os.RemoteException

dmel_sound_play_horn

void dmel_sound_play_horn(ComponentName cn)
throws android.os.RemoteException
効果音を再生する。(Hornの効果音)
Parameters:
cn – Component 名前
Throws:
android.os.RemoteException

dmel_sound_play_fail

void dmel_sound_play_fail(ComponentName cn)
throws android.os.RemoteException
効果音を再生する。(失敗の効果音)
Parameters:
cn – Component 名前
Throws:
android.os.RemoteException

[音声認識に関する関数]

dmel_voice_recognition_start

void dmel_voice_recognition_start(ComponentName cn)
throws android.os.RemoteException
音声認識する時に音声入力を開始する。
Parameters:
cn – Component 名前
Throws:
android.os.RemoteException

dmel_voice_recognition_stop

void dmel_voice_recognition_stop(ComponentName cn)
throws android.os.RemoteException
音声認識する時に音声入力を終了する。
Parameters:
cn – Component 名前
Throws:
android.os.RemoteException

dmel_voice_recognition_set_mode

void dmel_voice_recognition_set_mode(ComponentName cn,
boolean is_auto)
throws android.os.RemoteException
音声認識を行うことができるモードに切り替える。
Parameters:
cn – Componentの名前
is_auto – trueの場合は、自動的に音声認識開始と終了を繰り返す
Throws:
android.os.RemoteException

dmel_voice_recognition_get_mode

boolean dmel_voice_recognition_get_mode(ComponentName cn)
throws android.os.RemoteException
音声認識可能かどうかを返す。
Parameters:
cn – Componentの名前
Returns:
音声認識が可能な場合はtrue、不可能な場合はfalse
Throws:
android.os.RemoteException

5.6 マルチメディア(オーディオとビデオ)関数

dmel_audio_set_volume

```
void dmel_audio_set_volume(ComponentName cn,
                           float volume)
                           throws android.os.RemoteException
```

ボリュームの大きさを設定する。最小ボリューム:0、最大ボリューム:1
Parameters:
cn - Componentの名前
volume - ボリュームの大きさを設定する(0～1)
Throws:
android.os.RemoteException

dmel_audio_get_volume

```
float dmel_audio_get_volume(ComponentName cn)
                            throws android.os.RemoteException
```

現在のボリュームの大きさを返す。最小ボリューム:0、最大ボリューム:1
Parameters:
cn - Componentの名前
Returns:
ボリュームの大きさを返す(0～1)
Throws:
android.os.RemoteException

dmel_audio_play_mp3

```
void dmel_audio_play_mp3(ComponentName cn,
                         java.lang.String path)
                         throws android.os.RemoteException
```

MP3ファイルを再生する。(基本の経路は/ sdcard/ dongbu/で指定されている)
実際の経路が/ sdcard/dongbu/audio/test.mp3の場合
mBinder.dmel_audio_play_mp3(getComponentName()、"audio/ test、mp3");
Parameters:
cn - Componentの名前
path - MP3ファイルの経路(ファイル名を含む、基本の経路を除く)
Throws:
android.os.RemoteException

dmel_audio_stop_mp3

void dmel_audio_stop_mp3(ComponentName cn)
throws android.os.RemoteException
MP3 再生を停止する。
Parameters:
cn – Component 名前
Throws:
android.os.RemoteException

dmel_video_play_mp4

void dmel_video_play_mp4(ComponentName cn,
java.lang.String path)
throws android.os.RemoteException
MP4ファイルを再生する。(基本の経路は/ sdcard/ dongbu/で指定されている)
実際の経路が/ sdcard/dongbu/video/test.mp3の場合
mBinder.dmel_video_play_mp4(getComponentName()、"video/ test、mp3");
Parameters:
cn – Componentの名前
path – MP4ファイルの経路(ファイル名を含む、基本の経路を除く)
Throws:
android.os.RemoteException

5.7 モーションとサーボモーターに関する関数

dmel_motion_play

void dmel_motion_play(ComponentName cn,
java.lang.String motion)
throws android.os.RemoteException
ロボットモーションファイルを再生する。(基本の経路は/ sdcard/ dongbu/ motion/で指定されている)
Parameters:
cn – Componentの名前
motion – モーションファイル名ex)"01n_shake_head.dmt"
Throws:
android.os.RemoteException

dmel_motion_play2

```
void dmel_motion_play2(ComponentName cn,
                       java.lang.String motion,
                       boolean repeat,
                       boolean sync)
                         throws android.os.RemoteException
```

ロボットモーションファイルを再生する。メディアSyncと繰り返し設定が可能です。
（経路は/ sdcard/ dongbu/ motion/に指定されている）
Parameters:
cn – Componentの名前
motion – モーションファイル名ex）"01n_shake_head.dmt"
repeat – 停止の命令が入るまで連続的に繰り返し実行される。
sync – メディアと終了を合わせる。（repeatよりも優先される）
Throws:
android.os.RemoteException

dmel_motion_stop

```
void dmel_motion_stop(ComponentName cn)
                       throws android.os.RemoteException
```

現在実行中のモーションを中断する。
Parameters:
cn – Component 名前
Throws:
android.os.RemoteException

dmel_motion_servo_on

```
boolean dmel_motion_servo_on(ComponentName cn,
                             int id)
                             throws android.os.RemoteException
```

ID値に対応する上体モーターに"Servo ON"を設定する。
Parameters:
cn – Component 名前
id – id モーターのID
（0:右の肩モーター、1:右の上腕モーター、2:右の下腕モーター、3:左の肩モーター、
4:左の上腕モーター、5:左の下腕モーター、18:頭モーター、19:腰モーター）
Returns:
成功した場合はtrue、失敗した場合はfalseを返す
Throws:
android.os.RemoteException

dmel_motion_servo_off

```
boolean dmel_motion_servo_off(ComponentName cn,
                              int id)
                              throws android.os.RemoteException
```

ID値に対応する上体モーターに”Servo OFF”を設定する。
Parameters:
cn – Component名前
id – idモーターのID
(0:右の肩モーター、1:右の上腕モーター、2:右の下腕モーター、3:左の肩モーター、
4:左の上腕モーター、5:左の下腕モーター、18:頭モーター、19:腰モーター)
Returns:
成功した場合はtrue、失敗した場合はfalseを返す
Throws:
android.os.RemoteException

dmel_motion_set_servo

```
boolean dmel_motion_set_servo(ComponentName cn,
                              int on)
                              throws android.os.RemoteException
```

すべての上体モーターに”Servo ON”を設定する。
Parameters:
cn – Component 名前
on – Servo ON : 1, Servo OFF : 0
Returns:
成功した場合はtrue、失敗した場合はfalseを返す
Throws:
android.os.RemoteException

dmel_motion_get_position

```
int dmel_motion_get_position(ComponentName cn,
                             int id)
                             throws android.os.RemoteException
```

ID値に対応する上体モーターの現在位置の値を返す。
Parameters:
cn – Componentの名前
id – モーターのID(0:右の肩モーター、1:右の上腕モーター、2:右の下腕モーター、3:左の肩
モーター、4:左の上腕モーター、5:左の下腕モーター、18:頭モーター、19:腰モーター)
Returns:
このモーターの現在位置の値を返す
Throws:
android.os.RemoteException

dmel_motion_set_calib_value

```
boolean dmel_motion_set_calib_value(ComponentName cn,
                                    int id,
                                    int val)
                                    throws android.os.RemoteException
```

IDの値に対応する上体モーターに零点調整値を設定する。
Parameters:
cn – Componentの名前
id – モーターのID(0:右の肩モーター、1:右の上腕モーター、2:右の下腕モーター、3:左の肩モーター、4:左の上腕モーター、5:左の下腕モーター、18:頭モーター、19:腰モーター)
val – 零点調整値(-127～127)
Returns:
零点調整が成功するとtrue、失敗するとfalseを返す
Throws:
android.os.RemoteException

5.8 頭のLED制御に関する関数

dmel_hri_set_brow_beam

```
void dmel_hri_set_brow_beam(ComponentName cn,
                            int brightness)
                            throws android.os.RemoteException
```

額のLEDを設定する。
Parameters:
cn – Component 名前
brightness – 0 : オフ、1:弱めにオン、2:中間にオン、3:明るくオン
Throws:
android.os.RemoteException

dmel_hri_set_eye_led

```
void dmel_hri_set_eye_led(ComponentName cn,
                          int eyeCode,
                          int colorCode)
                          throws android.os.RemoteException
```

目のLEDを設定する。
Parameters:
cn – Componentの名前
eyeCode – 0:既存の状態を維持、1:くるくる、2:眉毛点滅、3:眉毛停止、4:完全にオン、5:完全に点滅、6:左右の効果、7:上下の効果、8:オフ
colorCode – 1 : RED LED, 2 : BLUE LED, 3 : PURPLE (RED + BLUE) LED
Throws:
android.os.RemoteException

dmel_hri_set_ear_led

```
void dmel_hri_set_ear_led(ComponentName cn,
                          int earCode)
                          throws android.os.RemoteException
```

耳のLEDを設定する。
Parameters:
cn – Componentの名前
earCode – 0:既存の状態を維持、1:オン、2:オフ、3:点滅
Throws:
android.os.RemoteException

dmel_hri_set_mouth_led

```
void dmel_hri_set_mouth_led(ComponentName cn,
                            int mouthCode)
                            throws android.os.RemoteException
```

口のLEDを設定する。
Parameters:
cn – Componentの名前
mouthCode – 0:既存の状態を維持、1:中央をオン、2:三つをオン、3:中で点滅、4:三つを点滅、5:話す
Throws:
android.os.RemoteException

06 Hovis Genie 基本的な例

Chapter6では、ロボットサービスによる本格的なロボットプログラミングを開始します。そして、Chapter4でGenieApiDemoプロジェクトを作成して、ロボットサービスを利用するための基盤を説明しました。この章では、様々なロボット制御に関する例を追加してGenieApiDemoプロジェクトを完成させます。

GenieApiDemoは全7つの画面を組み合わせています。つまり、GenieApiDemoは7つのActivityを組み合わせた形のAppです。今後、各節で追加して修正する必要があるファイルは、[表6-1]のとおりであり、GenieApiDemo Appに含まれている7つのActivityを示します。

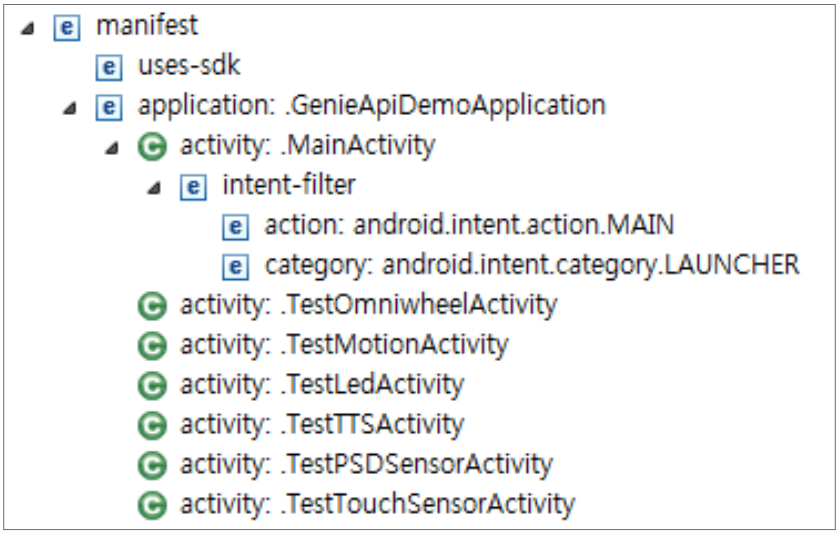
節	内容	UI レイアウト	ソースコード
6.1.	メイン画面	activity_main.xml	MainActivity.java
6.2.	駆動部の制御	activity_testomniwheel.xml	TestOmniwheelActivity.java
6.3.	モーション制御	activity_testmotion.xml	TestMotionActivity.java
6.4.	頭のLED制御	activity_testled.xml	TestLedActivity.java
6.5.	TTS制御	activity_testtts.xml	TestTTSActivity.java
6.6.	PSDセンサ制御	activity_testpsdsensor.xml	TestPSDSensorActivity.java
6.7.	タッチセンサ制御	activity_testtouchsensor.xml	TestTouchSensorActivity.java

[表 6-1] GenieApiDemoの画面構成

この章の各節での例を一つずつ作成することにより、Appを構成するActivityが追加されます。したがって、Appの構造が変更されます。これらの変更は、AndroidMainfest.xmlに追加宣言されます。AndroidManifest.xmlを開き、プロジェクトを作成する時にすでに追加したMainActivityを除いた残りの6つのActivityを[Listing6-1]の22～27行を参照して追加します。

```
1: <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2:     package="com.dongburobot.genieapidemo"
3:     android:versionCode="1"
4:     android:versionName="1.0" >
5:
6:     <uses-sdk
7:         android:minSdkVersion="10"
8:         android:targetSdkVersion="10" />
9:
10:    <application
11:        android:name=".GenieApiDemoApplication"
12:        android:icon="@drawable/ic_launcher"
13:        android:label="@string/app_name"
14:        android:theme="@style/AppTheme" >
15:        <activity
16:            android:name=".MainActivity">
17:            <intent-filter>
18:                <action android:name="android.intent.action.MAIN" />
19:                <category android:name="android.intent.category.LAUNCHER" />
20:            </intent-filter>
21:        </activity>
22:        <activity android:name=".TestOmniwheelActivity" />
23:        <activity android:name=".TestMotionActivity" />
24:        <activity android:name=".TestLedActivity" />
25:        <activity android:name=".TestTTSActivity" />
26:        <activity android:name=".TestPSDSensorActivity" />
27:        <activity android:name=".TestTouchSensorActivity" />
28:    </application>
29:
30: </manifest>
```

[Listing 4-2] BaseActivity.java



[図6-1] AndroidManifest.xmlの構造

AndroidManifest.xmlを介して見るGenieApiDemo Appの構造は、図6-1のとおりです。GenieApiDemo<application>(=App)はGenieApiDemoApplication.javaによって動作し、7つの<activity>があります。各<activity>はMainActivity.java、…、TestTouchSensorActivity.javaによって動作し、その中のMainActivityはLAUNCHERカテゴリとMAINアクションを持つ<intent-filter>があるので、ランチャーからAppを開始した時に最初に登場する画面を説明します。

本格的な例を作成する前にAndroidManifest.xmlにAppの構造を定義します。したがって、各節での例を作成する時にAndroidManifest.xmlの心配はありません。しかし、実際にAppを開発する際に、開発者がAndroidManifest.xmlにAppの構造を正しく指定していないことに起因する多くのエラーを発生させるので注意してください。

最後に、今回の章からは例を作成する方法を中心に、必要な部分を説明します。足りない部分は、Androidの開発者サイトまたは他のアンドロイドの基本の本を参照してください。

6.1 メイン画面の構成

アンドロイドは一つの画面が一つのActivityで構成されます。そして、このActivityには、通常のUIのレイアウトを構成するXMLとJavaファイルが含まれています。メイン画面でも同様に、XMLとJavaファイルがあります。XMLファイルは、activity_main.xmlであり、JavaファイルはMainActivity.javaです。

activity_main.xmlとMainActivity.javaは4章でGenieApiDemoプロジェクトを作成する時に、すでにプロジェクトに自動的に含まれています。したがって、プロジェクトに追加するファイルはありません。

EclipseのPackage ExplorerのGenieApiDemo> res> layoutフォルダは、プロジェクトで使用するすべてのUIのレイアウトが含まれています。activity_main.xmlファイルもここにあります。activity_main.xmlファイルを探し、ダブルクリックして開き、[Listing6-2]にソースを追加します。

```
1: <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     xmlns:tools="http://schemas.android.com/tools"
3:     android:layout_width="match_parent"
4:     android:layout_height="match_parent" >
5:
6:         <GridView
7:             android:id="@+id/gridview_apidemos"
8:             android:layout_width="match_parent"
9:             android:layout_height="match_parent"
10:            android:padding="10dp"
11:            android:verticalSpacing="10dp"
12:            android:horizontalSpacing="10dp"
13:            android:numColumns="auto_fit"
14:            android:columnWidth="60dp"
15:            android:stretchMode="columnWidth"
16:            android:gravity="center"
17:            android:background="#222222"
18:        />
19:
20: </RelativeLayout>
```

[Listing 6-2] activity_main.xml

activity_main.xmlは<RelativeLayout>を最上位のタグに、サブに<GridView>を含む構造です。
ここで、RelativeLayoutは、画面のウィジェットをどのような基準から相対的に配置するかを決めるレイアウトであり、GridViewには、画面に特定のウィジェットを格子状に配置します。

activity_main.xmlの構造によって構成されるUIを画面で示すと[図6-2]のとおりです。



[図 6-2] メイン画面

<GridView>は画面全体を占めています。8～9行のサイズのプロパティandroid:layout_widthとandroid:layout_heightに指定された値“match_parent”は、上位のウィジェットとサイズを同じにします。これは上位のプロパティである<ReleativeLayout>が画面全体を占めているためです。
画面全体を占める<GridView>は、格子状にサブウィジェットを含んでいます。<GridView>の残りのプロパティは、サブウィジェットをどのように配置するかを決める設定です。現在のサブウィジェットの配置は、android:numColumns=”auto_fit”に設定され、サブウィジェットのサイズが一行にいくつのウィジェットを配置するかに影響を与えます。

➔ ReleativeLayoutの主要なプロパティ

http://developer.android.com/guide/topics/ui/layout/relative.html

➔ GridViewの主要なプロパティ

http://developer.android.com/guide/topics/ui/layout/gridview.html

<GridView>のサブウィジェットはアイコンイメージとテキストで構成されています。これを定義する別のXMLファイルが必要です。 / res / layoutにdemoitems.xmlファイルを追加して、ソースコードを次のように変更します。

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3:     android:layout_width="fill_parent"
4:     android:layout_height="fill_parent"
5:     android:orientation="vertical"
6:     android:gravity="center">
7:
8:     <ImageView
9:         android:layout_width="65dp"
10:        android:layout_height="65dp"
11:        android:padding="6dp"
12:        android:gravity="center"
13:        android:id="@+id/demo_icon"/>
14:    <TextView
15:        android:layout_width="65dp"
16:        android:layout_height="30dp"
17:        android:gravity="center"
18:        android:ellipsize="end"
19:        android:singleLine="true"
20:        android:id="@+id/demo_name"/>
21:
22: </LinearLayout>
```

[Listing 6-3] demoitems.xml

demoitems.xmlは<ImageView>と<TextView>で構成されています。これは、[図6-2]のイメージアイコンとテキストをそれぞれ表現しています。これは、<GridView>に表現されるウィジェットの構造を表現したものであり、実際のデータソースを<GridView>に受け入れることではありません。実際のデータソースを受け入れることはMainActivity.javaで行われます。続いてMainActivity.javaを開き、[Listing6-4]を参照してソースコードを作成します。

1) BaseActivity クラスを受ける(18行目)

BaseActivityは4章で作成したクラスで、ロボットプログラミングの特性に応じて、Activityクラスを再定義したクラスです。 BaseActivityは、MIDの全体の画面を使用して、Activityがアンドロイドスタックに積まれることを防止する役割をします。また、サービスを利用するためのグローバル変数であるmBinderを使用できるようにします。

2) MainActivity クラスのメンバ変数の宣言(19～23行目)

```
public LinkedList<String> mDemoNameList;
public LinkedList<Integer> mDemoImgList;
private GridView mMainGrid;
private MainAdapter mMainAdapter;
```

mMainGridはactivity_main.xmlに宣言されたGridViewを割り当てるための変数です。mMainAdapterは(4)で作成するInnerクラスMainAdapterのオブジェクトとしてGridViewに割り当てるデータソースを設定するために使用されます。GridViewのサブウィジェットはアイコンイメージとテキストで構成されますが、この両方に該当する情報は、それぞれのmDemoImgListとmDemoNameListに割り当てられています。

3) onCreate()作成(26～50行目)

(2)で宣言されたメンバ変数を利用するソースコードが入ります。
28行目では、setContentView()を利用してMainActivityにactivity_main.xmlを割り当てます。
30～44行目では、GridViewにサブウィジェットを出力するデータをLinkedListに割り当てます。 46～48行目は、GridViewにAdapterを設定するコードです。 Adapterとは、多数のデータソースを供給する際の役割をします。最後に、49行目は、GridViewに割り当てられた各ウィジェットがクリックできるように設定するコードです。

注：アンドロイドでソース割り当て（R.java）

R.drawable.ファイル名 ➔ /res/drawable-*dpiのイメージリソース

R.layout.ファイル名 ➔ /res/layoutにあるレイアウトXMLファイルへのアクセス

R.id.ID名 ➔ 各レイアウトでファイルに宣言されたウィジェットをIDを使ってアクセス

4) MainActivityのInner クラスであるMainAdapter作成（97～126行目）

今回の例のコードで最も重要な部分です。MainAdapterはBaseAdapterを受けるAdapterクラスです。 MainAdapterはGridViewdにデータソースを供給する役割をします。MainAdapterはBaseAdapterを受けてからgetView()、getCount()、getItem()、getItemId()をオーバーライドする必要があります。

MainAdapterに実装されたgetView()、getCount()は、Adapterが適用されたウィジェットが出力される時に呼び出されます。getItem()とgetItemId()は必ず実装する必要がありますが、実際の役割はないので省略します。

getView()メソッドは、GridViewが画面にアイコンとテキストで構成されるサブウィジェットを描画するたびに一度ずつ呼び出されます。MainActivityが作成される時にonCreate()ですでにGridViewに出力する実際のデータイメージとテキストを割り当てます。 getView()は、このデータを利用しています。

getCount()は、GridViewに出力する実際のデータの個数を返します。ここの例では、6つのイメージとテキストがありますので6を返します。ここで返された6はgetView()を6回を呼び出せるようにする役割をして、最終的にGridViewには、6つのアイコンとテキストが表示されます。

Adapterは、今後のリストUIなど多数のデータソースを必要とするソースコードで常に使用されます。よく使われる部分なので、理解してください。

5) OnItemClickListener インターフェイスを多重に受ける（18行目）

```
public class MainActivity extends BaseActivity implements OnItemClickListener
```

GridViewでアイコンをタッチする時にそのタッチイベントを処理するためにOnItemClick-Listenerインターフェイスを多重に受けます。“implements OnItemClickListener”を入力して“MainActivity”にonItemClick()を追加します。

6) onItemClick() 作成(58～95行目)

onItemClick()は、GridViewでアイコンタッチイベントを処理するメソッドです。
onItemClick()の引数のうちintpositionは、GridViewでタッチされたアイコンの位置を示します。 positionの値は、左上が0であり、一つずつ増加します。ここの例では、Wheelのアイコンが0を返し、タッチセンサアイコンが5を返します。

onItemClick()は、返されたpositionの値に応じて、それぞれの例Activityに画面を切り替えます。他のActivityを開始するコードは次のとおりです。

```
Intent intent = new Intent(this, クラス名.class);
startActivity(intent);
```

startActivity()によって新しいActvitiyに切り替わるとMainActivityはonPause()関数を呼び出しますが、定義されていないので、親クラスであるBaseActivityのonPause()関数を呼び出します。 BaseAcitivityのonPause()関数を呼び出すと、finish()によってActivityが完全に終了します(4章参照)。

```
 1: package com.dongburobot.genieapidemo;
 2:
 3: import java.util.LinkedList;
 4:
 5: import android.content.Intent;
 6: import android.os.Bundle;
 7: import android.view.LayoutInflater;
 8: import android.view.Menu;
 9: import android.view.View;
10: import android.view.ViewGroup;
11: import android.widget.AdapterView;
12: import android.widget.AdapterView.OnItemClickListener;
13: import android.widget.BaseAdapter;
14: import android.widget.GridView;
15: import android.widget.ImageView;
16: import android.widget.TextView;
17:
18: public class MainActivity extends BaseActivity implements OnItemClickListener {
19:     public LinkedList<String> mDemoNameList;
```

```
20:    public LinkedList<Integer> mDemoImgList;
21:
22:    private GridView mMainGrid;
23:    private MainAdapter mMainAdapter;
24:
25:    @Override
26:    public void onCreate(Bundle savedInstanceState) {
27:        super.onCreate(savedInstanceState);
28:        setContentView(R.layout.activity_main);
29:
30:        mDemoNameList = new LinkedList<String>();
31:        mDemoImgList = new LinkedList<Integer>();
32:        mDemoNameList.add("Wheel");
33:        mDemoNameList.add("Motion");
34:        mDemoNameList.add("LED");
35:        mDemoNameList.add("TTS");
36:        mDemoNameList.add("距離センサ");
37:        mDemoNameList.add("タッチセンサ");
38:
39:        mDemoImgList.add(R.drawable.icon1);
40:        mDemoImgList.add(R.drawable.icon1);
41:        mDemoImgList.add(R.drawable.icon1);
42:        mDemoImgList.add(R.drawable.icon1);
43:        mDemoImgList.add(R.drawable.icon1);
44:        mDemoImgList.add(R.drawable.icon1);
45:
46:        mMainGrid = (GridView) findViewById(R.id.gridview_apidemos);
47:        mMainAdapter = new MainAdapter();
48:        mMainGrid.setAdapter(mMainAdapter);
49:        mMainGrid.setOnItemClickListener(this);
50:    }
51:
52:    @Override
53:    public boolean onCreateOptionsMenu(Menu menu) {
54:        getMenuInflater().inflate(R.menu.activity_main, menu);
55:        return true;
```

```
56:    }
57:
58:    public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
59:
60:        Intent intent;
61:
62:        switch (position) {
63:        case 0:
64:            // 車輪（オムニホイール）制御Activityに移動
65:            intent = new Intent(this, TestOmniwheelActivity.class);
66:            startActivity(intent);
67:            break;
68:        case 1:
69:            // モーション制御Activityに移動
70:            intent = new Intent(this, TestMotionActivity.class);
71:            startActivity(intent);
72:            break;
73:        case 2:
74:            // 頭LED制御Activityに移動
75:            intent = new Intent(this, TestLedActivity.class);
76:            startActivity(intent);
77:            break;
78:        case 3:
79:            // TTS Activity 移動
80:            intent = new Intent(this, TestTTSActivity.class);
81:            startActivity(intent);
82:            break;
83:        case 4:
84:            // PSD Activity 移動
85:            intent = new Intent(this, TestPSDSensorActivity.class);
86:            startActivity(intent);
87:            break;
88:        case 5:
89:            // Touch Activity 移動
90:            intent = new Intent(this, TestTouchSensorActivity.class);
91:            startActivity(intent);
```

```
92:         default:
93:             break;
94:         }
95:     }
96:
97:     public class MainAdapter extends BaseAdapter {
98:
99:         public int getCount() {
100:            return mDemoImgList.size();
101:        }
102:
103:        public Object getItem(int position) {
104:            return null;
105:        }
106:
107:        public long getItemId(int position) {
108:            return position;
109:        }
110:
111:        public View getView(int position, View convertView, ViewGroup parent) {
112:            if (convertView == null) {
113:                LayoutInflater li =  (LayoutInflater) getApplicationContext()
114:                        .getSystemService(LAYOUT_INFLATER_SERVICE);
115:                convertView = li.inflate(R.layout.demoitems, null);
116:            }
117:
118:            ImageView icon = (ImageView) convertView.findViewById(R.id.demo_icon);
119:            TextView name = (TextView) convertView.findViewById(R.id.demo_name);
120:
121:            icon.setBackgroundResource(mDemoImgList.get(position));
122:            name.setText(mDemoNameList.get(position));
123:
124:            return convertView;
125:        }
126:    }
127: }
```

[Listing 6-4] MainActivity.java

6.2 駆動部の制御

駆動部の制御は、Hovis Genieのオムニホイールの車輪を制御するプロセスをテストするための例です。今回の例では、ロボットが前方に50cm前進した後、元の位置に戻るようにするプログラムを作成します。

駆動部制御のActivityはactivity_testomniwheel.xmlとTestOmniwheelActivity.javaで構成されます。それぞれのファイルを/ res / layoutと/ src/ com/ dongburobot/ genieapiに追加します。

まず、駆動部制御ActivitiyのUIを構成します。activitiy_testomniwheel.xmlファイルを開き、ソースコードを次のように変更します。

```
1: <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     xmlns:tools="http://schemas.android.com/tools"
3:     android:layout_width="match_parent"
4:     android:layout_height="match_parent" >
5:
6:     <!-- テストタイトル -->
7:     <TextView
8:         android:id="@+id/tv_title"
9:         android:layout_width="match_parent"
10:        android:layout_height="85dp"
11:        android:text="Omniwheel test"
12:        />
13:
14:    <!-- ロボットのイメージ -->
15:    <ImageView
16:        android:id="@+id/iv_image"
17:        android:layout_width="match_parent"
18:        android:layout_height="320dp"
19:        android:layout_below="@id/tv_title"
20:        android:contentDescription="イメージ "
21:        />
22:
23:    <!-- 開始と終了ボタン -->
24:    <LinearLayout
25:        android:layout_width="match_parent"
26:        android:layout_height="wrap_content"
```

```
27:         android:layout_alignParentBottom="true">
28:         <Button
29:             android:id="@+id/btn_start"
30:             android:layout_width="160dp"
31:             android:layout_height="wrap_content"
32:             android:text="開始"
33:             />
34:         <Button
35:             android:id="@+id/btn_quit"
36:             android:layout_width="160dp"
37:             android:layout_height="wrap_content"
38:             android:text="終了"
39:             />
40:     </LinearLayout>
41:
42: </RelativeLayout>
```

[Listing 6-5] activity_testomniwheel.xml

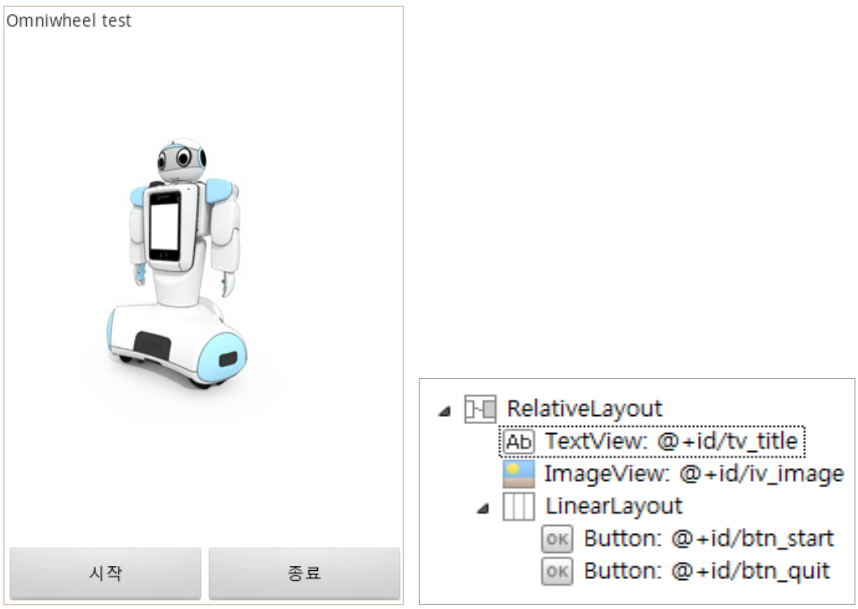
activity_testomniwheel.xmlの構成は<RelativeLayout>を最上位のタグにして<TextView>、<ImageView>、<LinearLayout>を含んでいます。ここで<LinearLayout>は再び<Button>ウィジェットの二つを含んでいます。

<RelativeLayout>を使用すると、各ウィジェットごとに位置を相対的に指定するプロパティを必ず記述します。もし、相対的な位置プロパティを指定しない場合、親の左上を基準にウィジェットが配置されます。前の節の例の場合、<RelativeLayout>に<GridView>ウィジェットだけが存在しているため省略をしましたが、今回の例のように、多数のウィジェットがある場合には位置プロパティ技術の省略はエラーの原因になります。

<RelativeLayout>の存在するウィジェットのうち<TextView>には、位置プロパティがありません。したがって、左上に配置されます。<ImageView>の位置プロパティは、19行目にあります。このラインは、<ImageView>をidがtv_titleな<TextView>の下に配置します。<LinearLayout>の位置プロパティは27行目で、<LinearLayout>は<RelativeLayout>の一番下に配置されます。

```
19:        android:layout_below="@id/tv_title"
…
27:        android:layout_alignParentBottom="true"
…
```

このように設定されたUI画面は、最終的に以下のとおりです。



[図 6-3] 駆動部の制御

続いて、EclipseのPackage Explorerのsrc> com.dongburobot.genieapiでTestOmniwheelActivity.javaファイルを選択し、[Listing6-6]を参照してソースコードを追加します。作成方法は次のとおりです。今回の例は、wheel_1.pngが必要です。このファイルをres/ drawable-mdpiにコピーします。（www.dongburobot.com資料室を参照）

1) BaseActivity を受ける

```
public class TestOmniwheelActivity extends BaseActivity …
```

2) ウィジェットに関連するメンバ変数を宣言する

```
private ImageView mIvWheelImg;
private Button mBtnStart;
private Button mBtnQuit;
```

3) onCreate()の作成

onCreate()でレイアウトXMLファイルを接続して、各メンバ変数にウィジェットのリソースを設定します。21行目のsetContentView()はactivity_testomniwheel.xmlで定義されたUIを現在のActivityの画面に表現しています。23～25行目は[図6-2]のロボットのイメージを設定するコードです。27～30行目は、ボタンにタッチイベントを設定するためのコードです。

4) OnClickListener インターフェイスを多重に受けた後、onClick()を作成

今回の例は、”開始”と “終了”の二つのボタンがあります。このボタンを押したときに、タッチイベントを処理するためのソースコードを作成します。まずOnClickListenerインターフェイスを多重に受けます。

```
public class TestOmniwheelActivity extends BaseActivity implements OnClickListener …
```

“implements OnClickListener”を追加して、 “TestOmniwheelActivity”の上にマウスカーソルを置くと、onClick()を追加ができます。

5) moveOmniWheel(), stopOmniWheel() 作成

moveOmniWheel()はロボットを50cm前進させて、所定の位置に戻す関数であり、stopOmniWheel()はロボットを停止するコードです。詳細は、再度説明します。

6) onPause()の作成

onPause()はTestOmniwheelActivityが画面から消える瞬間に呼び出される関数です。44行目のsuper.onPause()はTestOmniwheelActivityの親クラスであるBaseActivity()のonPause()関数を呼び出します。BaseActivityのonPause()は、finish()を呼び出して、Activityを完全に終了させるので、Androidのスタックにactivityを残しません。

7) onKeyDown() 作成

MainActivityもBaseActivityを受けて作成しました。そのためMainActivityのGridViewでアイコンをクリックして、他の例のActivityに切り替えた時にMainActivityもアンドロイドスタックに保存されません。
通常のAppは、A -> B Activityに切り替えながらA ActivityがAndroidスタックに保存されるので、Backボタンを押すと、A Activityに戻ります。しかし、MainActivityはアンドロイドのメモリスタックに保存されていないためBackボタンを押すと、Appが終了します。
34～40行のonKeyDown()はBackボタンを押すと、MainActivityに切り替えるコードです。このコードにより、Backボタンを押すと、Appが終了されず、メイン画面に戻ります。
TestOmniwheelActivity.javaのソースコードは次のとおりです。

```
 1: package com.dongburobot.genieapidemo;
 2:
 3: import android.content.Intent;
 4: import android.os.Bundle;
 5: import android.os.RemoteException;
 6: import android.view.KeyEvent;
 7: import android.view.View;
 8: import android.view.View.OnClickListener;
 9: import android.widget.Button;
10: import android.widget.ImageView;
11:
12: public class TestOmniwheelActivity extends BaseActivity
                                                    implements OnClickListener {
13:
14:     private ImageView mIvWheelImg;
15:     private Button mBtnStart;
16:     private Button mBtnQuit;
17:
18:     @Override
19:     protected void onCreate(Bundle savedInstanceState) {
20:         super.onCreate(savedInstanceState);
21:         setContentView(R.layout.activity_testomniwheel);
22:
23:         mIvWheelImg = (ImageView)findViewById(R.id.iv_image);
24:         mIvWheelImg.setImageResource(R.drawable.wheel_1);
25:         mIvWheelImg.setScaleType(ImageView.ScaleType.FIT_XY);
26:
27:         mBtnStart = (Button)findViewById(R.id.btn_start);
28:         mBtnQuit = (Button)findViewById(R.id.btn_quit);
29:         mBtnStart.setOnClickListener(this);
30:         mBtnQuit.setOnClickListener(this);
31:     }
32:
33:     @Override
34:     public boolean onKeyDown(int keyCode, KeyEvent event) {
35:         if (keyCode == KeyEvent.KEYCODE_BACK) {
```

```
36:             Intent intent = new Intent(this, MainActivity.class);
37:             startActivity(intent);
38:         }
39:         return super.onKeyDown(keyCode, event);
40:     }
41:
42:     @Override
43:     protected void onPause() {
44:         super.onPause();
45:     }
46:
47:     public void onClick(View v) {
48:         switch (v.getId()) {
49:         case R.id.btn_start:
50:             moveOmniwheel();
51:             break;
52:         case R.id.btn_quit:
53:             stopOmniwheel();
54:             break;
55:         default:
56:             break;
57:         }
58:     }
59:
60:     private void moveOmniwheel() {
61:         if (myRobotApp.mBinder == null)
62:             return;
63:
64:         try {
65:             myRobotApp.mBinder.dmel_robot_set_pose(getComponentName(), 0, 0, 0);
66:             myRobotApp.mBinder.dmel_navigate_prepare(getComponentName());
67:             myRobotApp.mBinder.
                  dmel_navigate_add_goal_pose(getComponentName(), 0.5, 0, 0, true, 1);
68:             myRobotApp.mBinder.
                  dmel_navigate_add_goal_pose(getComponentName(), 0, 0, 0, true, 1);
69:             myRobotApp.mBinder.dmel_navigate_start(getComponentName());
```

```
70:         } catch (RemoteException e) {
71:             e.printStackTrace();
72:         }
73:     }
74:
75:     private void stopOmniwheel() {
76:         if (myRobotApp.mBinder == null)
77:             return;
78:
79:         try {
80:             myRobotApp.mBinder.dmel_navigate_stop(getComponentName());
81:         } catch (RemoteException e) {
82:             e.printStackTrace();
83:         }
84:     }
85: }
```

[Listing 6-6] TestOmniwheelActivity.java

今回の例で、最も重要な部分は、開始と終了ボタンをタッチしたときに呼び出されるmoveOmniwheel()とstopOmniwheel()関数です。この二つの関数は、ロボットサービスを利用して駆動を制御します。moveOmniwheel()はロボットを50cm前進させ、所定の位置に戻るように動作させ、stopOmniwheel()は、ロボットの駆動を停止します。

moveOmniwheel()は、ロボットサービスを利用しています。ロボットサービスはGenieApiDemoAppが実行される時に、バインディングされます。バインドされたサービスは、IHovisGenieServiceのオブジェクトであるmBinderを介してロボットのAPIを呼び出すことができます。mBinderは次のようにGenieApiApplicationに宣言されています。

GenieApiDemoApplication.java 14行目:

```
public IHovisGenieService mBinder = null;
```

GenieApiDemoApplication
クラスに定義されたmBinderにアクセスするためには、GenieApiApplicationのオブジェクトが必要です。このオブジェクトは、BaseActivityに宣言されています。

BaseActivity.java 10行目:

```
protected GenieApiDemoApplication myRobotApp;
```

現在TestOmniwheelActivityクラスはBaseActivityの子クラスであるためmyRobotAppを宣言しなくても使用できます。そのため最終的にロボットのサービスはmyRobotApp.mBinderを使用してアクセスできます。

moveOmniwheel()の61行目は、myRobotApp.mBinderのオブジェクトがnullかどうかをチェックします。ロボットサービスは、外部で動作するので、たとえAppを開始した時にサービスバイドしたとしても、中間バインディングが切断されることがあります。ロボットサービスがバインドされた状態であれば、ロボット制御時にエラーが発生するので、このコードを使用してエラーを未然に防止します。

64～72行目は、ロボットサービスを利用するコードです。ロボットサービスは、遠隔のプロセスと相互の通信をする構造であるため、RemoteExceptionが発生することがあります。ロボットサービスは、例外の処理のために必ずtry-catchの文を使用する必要があります。

try-catchの文に包まれた部分には、次のように実際のロボットサービスのAPI関数を呼び出す部分があります。各ロボットサービス関数は、次のとおりであり、行ごとに説明します。

dmel_robot_set_pose

```
boolean dmel_robot_set_pose(ComponentName cn,
                            double x,
                            double y,
                            double theta)
                           throws android.os.RemoteException
```

ロボットの現在位置をx、y、thetaで設定する。ロボットの現在位置を(0、0、0)に設定すると、現在の位置が基準点になる。

Parameters:
cn – Component 名前
x – (meter)
y – (meter)
theta – (Radian -PI ~ +PI)
Returns:
成功した場合は true, **失敗した場合は**false **返す**
Throws:
android.os.RemoteException

dmel_navigate_prepare

```
void dmel_navigate_prepare(ComponentName cn)
                    throws android.os.RemoteException
```

ロボットのナビゲーションのための初期化を実行する。初期化する時に、ロボットの現在位置を(0、0、0)に設定し、現在のHeadingを0度に設定する。
Parameters:
cn – Component 名前
Throws:
android.os.RemoteException

dmel_navigate_add_goal_pose

```
boolean dmel_navigate_add_goal_pose(ComponentName cn,
                                    double x,
                                    double y,
                                    double theta,
                                    boolean rflag,
                                    int time)
                                    throws android.os.RemoteException
```

ロボットが訪問するポイントを追加する。ロボットに設定された開始点(0、0、0)を基準に訪問するポイントを決定する。(dmel_navigate_prepare()を参照)ロボットは設定された訪問ポイントをdmel_navigate_start()を介して訪問を開始する。
Parameters:
cn – Componentの名前
x – 始点の基準x(meter)
y – 始点の基準y(meter)
theta – 始点からロボットのHeadingを基準にしたロボットの方向(Radian–PI〜+ PI)
rflag – true時にロボットが訪問ポイントに到着した時にtheta方向を眺める
time – ロボットが訪問ポイントに到着した時の待機時間(sec)
Returns:
Throws:
android.os.RemoteException

dmel_navigate_start

```
boolean dmel_navigate_start(ComponentName cn)
                    throws android.os.RemoteException
```

ロボットのナビゲーションを実行する。ロボットのナビゲーションは事前設定されたgoal_poseポイントずつ訪問する。
Parameters:
cn – Componentの名前
Returns:
成功した場合はtrue、失敗した場合falseを返す
Throws:
android.os.RemoteException

dmel_robot_set_pose()関数は、ロボットの現在位置をx、y、thetaで設定します。ここでは、0、0、0に設定したので、現在のロボットが置かれた位置を座標平面上のx座標0、y座標0、そしてロボットの見ている方向を0度に設定します。引数の値の単位はx、yがmであり、thetaの値はradianです。

dmel_navigate_prepare()は、ロボットのナビゲーションのための初期化を実行する関数です。その機能はdmel_robot_set_pose()と同じです。(上記の例では、両方のラインの一つは省略可能)
dmel_navigate_add_goal_pose()は、ロボットの移動ポイントを指定する役割をします。dmel_navigate_add_goal_pose()は移動ポイントの情報を追加するだけで、実際のロボットの駆動はdmel_navigate_start()関数が呼び出される時に開始します。

最初のdmel_navigate_add_goal_pose()の引数は、ComponentNameの最初の引数を除いたxメートル、yメートル、theta(radian)を意味します。コードの0.5、0、0は50cm前進したポイントで、ロボットが動く前眺めていた方向を意味します。この関数の5番目の引数は、ロボットが到着した時に3番目の引数で指定されたtheta方向を眺めるかを指定するため、true、falseを指定します。6番目の引数は、ロボットが到着した後、到着ポイントでの待機時間(sec)を意味します。このコードでは、true、1なので、ロボットの到着した後に、0度を眺めて1秒待機します。

2番目のdmel_navigate_add_goal_pose()の引数は、0、0、0、true、1です。前の3つの引数は、元の位置を意味します。4番目の引数がtrueなのでロボットが元の位置に戻ると、前方を眺めるように回転をして、5番目の引数によって、到着後1秒待機します。

その後、dmel_navigate_start()関数が呼び出されると、実際のロボットが駆動します。

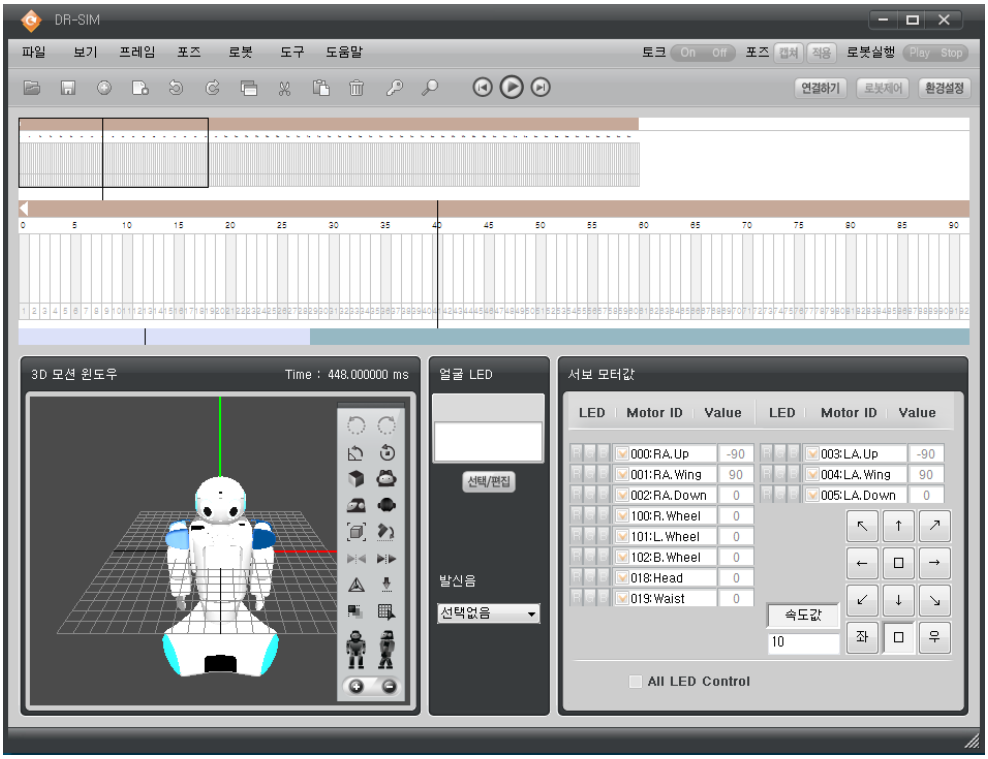
```
myRobotApp.mBinder.dmel_robot_set_pose(getComponentName(), 0, 0, 0);
myRobotApp.mBinder.dmel_navigate_prepare(getComponentName());
myRobotApp.mBinder.dmel_navigate_add_goal_pose(getComponentName(), 0.5, 0, 0, true, 1);
myRobotApp.mBinder.dmel_navigate_add_goal_pose(getComponentName(), 0, 0, 0, true, 1);
myRobotApp.mBinder.dmel_navigate_start(getComponentName());
```

6.3 モーション制御

今回の例は、Hovis Genieでモーションを動作させる例です。モーション制御とは、ロボットの駆動部と関節を構成するモーターを動作させて、ロボットの動きを制御することを意味します。Hovis Genieは全11個のモーターを搭載しています。各腕に3つのモーターがあり、腰モーター、頭モーター、そして3つのオムニホイール駆動部モーターがあります。

モーション制御をするために、それぞれのモーター制御が必要です。しかし、モーターを一つずつ制御する場合、制御が難しくなるだけでなく、効率的にモーションを制作することも困難です。そこで、Dongbu Robotは、ロボットのモーションエディタツールであるDR-SIMを提供しています。ユーザーは、DR-SIMを利用してHovis Genieのモーションを編集できます。

http://www.dongburobot.com/jsp/cms/view.jsp?code=100122&isSkin=Y&cmd=view&boardCode=100074&bseq=3703



[図 6-4] DR-SIM モーションエディタプログラム

DR-SIMは、仮想的に表現された3Dロボットをベースにモーションを制作し、タイムラインを提供して、時間の流れに沿ったモーションを編集できるプログラムです。DR-SIMを用いて作成されたモーションファイルは、Hovis　Genieモーション制御に使用され、拡張子はDMTです。(DR-SIMの使用方法は、他のマニュアルを参照)

Hovis Genieのモーションファイルは MIDのSDカードに搭載され、ロボットのモーションファイルの位置は/ dongbu/ motionです。

ロボットのモーションフォルダへのアクセスにはMIDとUSBメモリデバイスを使用します。MID上で設定>アプリケーション>開発>USBデバッグの項目のチェックを解除してPCと接続し、MIDのSD Cardを開きます。そして/ dongbu/ motionにアクセスします。

MIDのモーションフォルダにはすでに様々なモーションが搭載されています。このモーションファイルは、基本的なモーションでリストは次のとおりです。

ファイル名	内容
01m_turn_l.dmt	“元の位置回転 (右)
01m_turn_r.dmt	“元の位置回転 (左)
01n_adbomen0.dmt	片手でお腹を触る
01n_adbomen1.dmt	両手でお腹を触る
01n_aging.dmt	モーター Aging
01n_bend_weist.dmt	腰を下げる
01n_body.dmt	体磨き
01n_clap.dmt	胸の前で拍手
01n_command1.dmt	指揮する1
01n_command2.dmt	指揮する2
01n_command3.dmt	指揮する3
01n_command4.dmt	指揮する4
01n_dance.dmt	踊り
01n_face.dmt	洗面
01n_front_hand.dmt	前へならえしながら頭キョロキョロ
01n_goodjob.dmt	片腕を前へ出し (最高だよ!)

ファイル名	内容
01n_gymnastics.dmt	国民体操
01n_hide.dmt	“両腕で顔を隠す
01n_hurrah0.dmt	歩きながら万歳
01n_hurrah1.dmt	万歳したまま両腕を左右に振る:他の方向
01n_hurrah2.dmt	万歳したまま両腕を左右に振る:同じ方向
01n_kiss0.dmt	チュー1 (片手を口に付けてはなし)
01n_kiss1.dmt	チュー2 (両手を口に付けてはなし)
01n_love.dmt	元の位置で愛してる(両腕を上に上げてハート[円形]の作成)
01n_muscle	筋肉を自慢
01n_pos1	ポーズ1
01n_scrape.dmt	目をこする(目の前の片手を左右に動かす)
01n_scratch	頭を傷付ける
01n_self_msg.dmt	自分でマッサージ
01n_shake_head0.dmt	片腕を持ち上げて振る
01n_shake_head1.dmt	両腕を広げて振る (腕を上げずに)
01n_shake_head2.dmt	両腕を交互に振る(アップアップする)
01n_shake_head.dmt	頭を早く振る
01n_sidebyside.dmt	片腕を前へならえ
01n_sorrow.dmt	泣く(目の前で両手を左右に動かす)
01n_stare0.dmt	元の位置でキョロキョロ
01n_stare1.dmt	片手を頭に当ててキョロキョロ
01n_stare2.dmt	片手を耳に当ててキョロキョロ
01n_stretching.dmt	伸び
01n_tekwuando.dmt	テコンドー
01n_tekwuando_wide.dmt	テコンドー (MID横の時)

ファイル名	内容
01n_tooth.dmt	うがい
01n_turn_360_0.dmt	元の位置で360度回転(1回)
01n_turn_360_1.dmt	元の位置で360度回転(繰り返し)
01n_turn_head_l.dmt	頭の回転 (右)
01n_turn_head_r.dmt	頭の回転 (左)
01n_turn_love.dmt	回転しながら愛してる (両腕を上に上げてハート[円形]を作成)
01n_turn_shake.dmt	胸の前で両腕を折って車輪を左右に振る(体を振るように)
01n_up_hand.dmt	片腕を上に上げる
01n_watch_clock.dmt	手首の時計を見るようなモーション(肘を折り顔の近くに))
01n_wave.dmt	両腕ウェーブ
basicpose.dmt	正しい姿勢
mid_rotate.dmt	MIDが横のとき縦にする
motion_highwalk.dmt	早く歩く
motion_slow_walk.dmt	ゆっくり歩く
motion_standby.dmt	準備姿勢
motion_walk.dmt	普通に歩く

今回の例では、すでに搭載されている基本的なモーションを使用して、ロボットのモーションを実行します。モーション制御は、activity_testmotion.xmlとTestMotionActivity.javaで構成されているので、この二つのファイルをGenieApiDemoプロジェクトに追加します。そしてactivity_testmotion.xmlを次のように変更してモーション制御のUIを構成します。

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3:     android:layout_width="match_parent"
4:     android:layout_height="match_parent" >
5:
6:     <!-- テストのタイトル-->
7:     <TextView
```

```
 8:        android:id="@+id/tv_title"
 9:        android:layout_width="match_parent"
10:        android:layout_height="85dp"
11:        android:text="Motion test"
12:        />
13:
14:    <!-- モーションリスト -->
15:    <ListView
16:        android:id="@+id/lv_motions"
17:        android:layout_width="match_parent"
18:        android:layout_height="wrap_content"
19:        android:layout_below="@id/tv_title"
20:        />
21:
22: </RelativeLayout>
```

[Listing 6-7] activity_testmotion.xml

activity_testmotion.xmlの最上位<RelativeLayout>は、テストタイトルを表現する<TextView>とモーションリストを表現するための<ListView>が含まれています。これは、UI構成を画面で表現すると図6-5と同じです。



[図 6-5] モーション制御

TestMotionAcitivity.javaで、ListViewの項目の設定の核心は、Adapterです。 6.1節では、メイン画面のGridViewを構成する時に、様々なデータソースを受け入れるためにAdapterを使用したように、ここでもAdapterを使用します。
ただし、6.1節ではイメージとテキストで構成された多数のデータソースを受け入れるためにdemoitems.xmlにサブウィジェットを定義してMainAdapterという名前のクラスを作成しましたが、ここでは“Walk”、 “tekwuando”など一つのTextViewのみを使用するので、アンドロイドが提供する基本的なArrayAdapterを使用します。ただし、一つのTextViewで構成されたサブウィジェットの定義は必要なので、/ res / layoutにlistitems.xmlを追加して、ソースを追加します。

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3:     android:layout_width="match_parent"
4:     android:layout_height="match_parent"
5:     android:orientation="vertical" >
6:
7:     <TextView
8:         android:id="@+id/tv_name"
9:         android:layout_width="wrap_content"
10:         android:layout_height="40dp"
11:         android:gravity="center_vertical"
12:         />
13:
14: </LinearLayout>
```

[Listing 6-7] listitems.xml

listitems.xmlは横が<TextView>に割り当てられている文字列のサイズで、縦が40dpであるTextViewを示しており、図6-4の”Walk”と “tekwuando”に該当するウィジェットです。

アンドロイドのサイズの単位

px – ピクセル(pixel)

dp – deviece-independent pixel

アンドロイドのサイズを表すいくつかの単位がありますが、主にdp(dpi)を使用しています。その理由は、px単位で指定されたウィジェットは、機器の解像度の違いによって、画面が異なる場合があるからです。dpを単位として指定すると、どのような画面でも同じ比率のサイズで見えるので、dpを使用することをお勧めします。

最後にTestMotionActivity.javaを[Listing6-8]を参照して、次のような手順で作成します。

1) BaseActivity を受ける

```
public class TestMotionActivity extends BaseActivity …
```

2) ウィジェットに関するメンバ変数の宣言

```
private ListView mMotionListView;
```

3) onCreate()の作成 – ListViewとAdapter

TestMotionActivityが、最初の画面に表示される時に、実行されるコードです。24行目のset-ContentView()関数は、activity_testmotion.xmlに定義されたUIを呼び出します。26行目では、ListView IDを介してメンバ変数mMotionListViewにリソースを割り当てます。

28～30行は、ArrayListに文字列を割り当てます。32～35行では、ArrayAdapterのオブジェクトadapterを生成します。この部分では最も核心のコードです。 ArrayAdapterのコンストラクタとそのソースコードは次のとおりです。

```
ArrayAdapter(Context context, int resource, int textViewResourceId, List<T> objects)


ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                                        R.layout.listitems, R.id.tv_name, arrayList);
```

上記のコードの2番目の引数は、少し前に宣言したlistitems.xmlのリソースであり、3番目の引数は、listitems.xmlに宣言されたTextViewのIDです。そして最後の4番目の引数は、28～30行目に割り当てられたリストの文字列です。この引数の値を介してadapterはListViewに供給するデータの型と値を維持します。最後に、36行目のListViewには、adapterを付けてListViewにデータ割り当てます。37行目では、ListViewにタッチイベントを登録します。

4）onItemClickListenerインターフェイスを多重に受けた後、onItemClick()を作成

ユーザーのタッチイベントを処理する関数です。ListViewの位置は0からカウントされるので、この例では、"Walk"は0であり、"tekwuando"は1です。

```
 1: package com.dongburobot.genieapidemo;
 2:
 3: import java.util.ArrayList;
 4:
 5: import android.content.Intent;
 6: import android.os.Bundle;
 7: import android.os.RemoteException;
 8: import android.util.Log;
 9: import android.view.KeyEvent;
10: import android.view.View;
11: import android.widget.AdapterView;
12: import android.widget.AdapterView.OnItemClickListener;
13: import android.widget.ArrayAdapter;
14: import android.widget.ListView;
15:
16: public class TestMotionActivity extends BaseActivity
                                                implements OnItemClickListener {
17:
18:     private ListView mMotionListView;
19:
20:     @Override
21:     protected void onCreate(Bundle savedInstanceState) {
22:         // TODO Auto-generated method stub
23:         super.onCreate(savedInstanceState);
24:         setContentView(R.layout.activity_testmotion);
```

```
25:
26:         mMotionListView = (ListView)findViewById(R.id.lv_motions);
27:
28:         ArrayList<String> arrayList = new ArrayList<String>();
29:         arrayList.add("Walk");
30:         arrayList.add("tekwuando");
31:
32:         ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
33:                 R.layout.listitems,
34:                 R.id.tv_name,
35:                 arrayList);
36:         mMotionListView.setAdapter(adapter);
37:         mMotionListView.setOnItemClickListener(this);
38:     }
39:
40:     @Override
41:     public boolean onKeyDown(int keyCode, KeyEvent event) {
42:         if (keyCode == KeyEvent.KEYCODE_BACK) {
43:             Intent intent = new Intent(this, MainActivity.class);
44:             startActivity(intent);
45:         }
46:         return super.onKeyDown(keyCode, event);
47:     }
48:
49:     public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
50:         if (myRobotApp.mBinder == null)
51:             return;
52:
53:         try {
54:             myRobotApp.mBinder.dmel_motion_stop(getComponentName());
55:
56:             switch (position) {
57:             case 0:
58:                 myRobotApp.mBinder.
                        dmel_motion_play(getComponentName(), "motion_walk.dmt");
```

```
59:                 Log.i("motion", "0 clicked");
60:                 break;
61:             case 1:
62:                 myRobotApp.mBinder.
                        dmel_motion_play(getComponentName(), "01n_tekwuando_wide.dmt");
63:                 Log.i("motion", "1 clicked");
64:             default:
65:                 break;
66:             }
67:
68:             Thread.sleep(1000);
69:         } catch (RemoteException e) {
70:             e.printStackTrace();
71:         } catch (InterruptedException e) {
72:             e.printStackTrace();
73:         }
74:     }
75: }
```

[Listing 6-7] listitems.xml

onItemClick()は、ユーザーがリストから項目をタッチするときに実行されます。リストの項目がタッチされると、50行目のmyRobotApp.mBinderをチェックして、サービスが異常なく接続されているかどうかを確認します。異常がなければ、現在実行されているモーションを停止する命令を下した後、モーションを開始します。

```
myRobotApp.mBinder.dmel_motion_stop(getComponentName());
…
myRobotApp.mBinder.dmel_motion_play(getComponentName(), ファイル名);
```

dmel_motion_play

void dmel_motion_play(ComponentName cn,
java.lang.String motion)
throws android.os.RemoteException
ロボットモーションファイルを再生する。(基本経路は/ sdcard/ dongbu/ motion/に指定されている)
Parameters:
cn - Componentの名前
motion - モーションファイルの名前 ex) "01n_shake_head.dmt"
Throws:
android.os.RemoteException

dmel_motion_stop

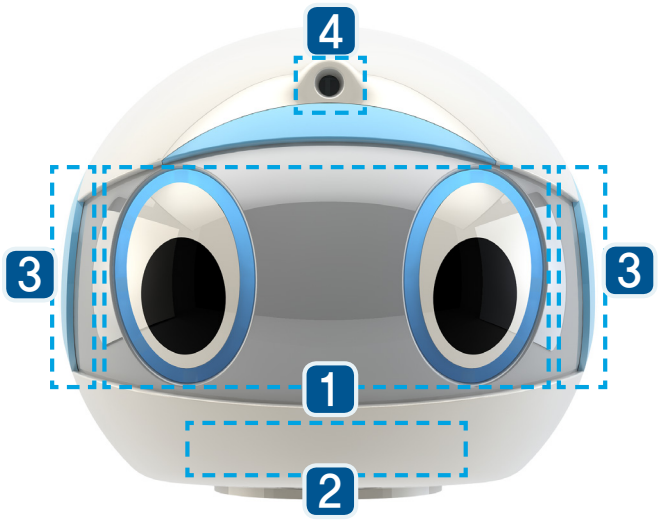
void dmel_motion_stop(ComponentName cn)
throws android.os.RemoteException
現在、実行しているモーションを中断する
Parameters:
cn - Componentの名前
Throws:
android.os.RemoteException

例56～66行目はリスト上でタッチされた位置に応じてモーションを実行するコードです。このコードによって、"Walk"を選択するとmotion_walk.dmt、"tekwuando"を選択すると01n_tekwuando_wide.dmtのモーションファイルを再生します。

ここで注意する点はモーションファイルが必ずSDカードの/ dongbu/ motionに存在することです。皆さんは、追加のファイルが存在するかどうかという例外を処理してください。また、他の基本的なモーションリストから増やしながら、練習してDR-SIMで作成したモーションを制御してください。

6.4 頭のLED制御

今回の節では、Hovis Genieの頭LEDの制御について説明します。Hovis Genieの頭には、LEDを制御することができるボードがあり、目、口、耳、額のLEDをオンまたはオフにできます。具体なLEDの構成は次の図のとおりです。



[図 6-6] Hovis Genie頭のLEDの配置(1 - 目、2 - 口、3 - 耳、4 - 額))

1番は目のLEDです。片方の目には、赤と青のLEDが各8個ずつ配置され、16個のLEDを持ち、両方の目には32個のLEDが付いています。目のLEDの場合、青色、赤色をそれぞれ設定することができ、青と赤を同時にオンにして紫の効果を出すことができます。
2番の位置は口のLEDです。口のLEDは3個で構成されています。 3番は耳のLEDで両方にあります。耳の場合、片方の耳に各4個ずつLEDを持っており、すべてオンまたはオフにするなど一括した制御が可能です。4番の位置は額の上のLEDで明るさの強度を調節できるLEDです。そして、口と耳と額のLEDは単色です。

ロボットサービスは頭のLED制御を効率的にするために、それぞれLEDを制御するのではなく、モジュールごとの効果を中心に関数を提供しています。この関数は、LEDを介してロボットの感情表現と視覚効果に使用されます。

頭のLED制御に必要なactivity_testled.xmlとTestLedActivity.javaをGenieApiDemoプロジェクトに追加して、activity_testled.xmlを開いて例のためのUIを構成します。

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3:     android:layout_width="match_parent"
4:     android:layout_height="match_parent" >
5:
6:     <!-- テストのタイトル -->
7:     <TextView
8:         android:id="@+id/tv_title"
9:         android:layout_width="match_parent"
10:         android:layout_height="85dp"
11:         android:text="LED test"
12:         />
13:
14:     <!-- LED 制御リスト -->
15:     <ListView
16:         android:id="@+id/lv_ledcontrols"
17:         android:layout_width="match_parent"
18:         android:layout_height="wrap_content"
19:         android:layout_below="@id/tv_title"
20:         />
21:
22: </RelativeLayout>
```

[Listing 6-7] activity_testmotion.xml

頭のLED制御のUIは以前の6.3節のモーション制御とほぼ同じであり、画面は[図6-7]と同じです。



[図 6-7] 頭のLED制御

今回の例も同じようにListViewに項目となるサブウィジェットを作成して、onCreate()内の文字列データを追加してAdapterを作成します。そしてAdapterをListViewに登録して、画面に表示します。最終的にListViewにタッチイベントを設定してonItemClickListenerインターフェイスを受けてonItemClick()を作成します。

ListViewのサブウィジェットを指定するlistitems.xmlは6.3節ですでに作成したので省略します。TestLedActivity.javaの作成法も似ているので、6.3節を参照しています。

```
1: package com.dongburobot.genieapidemo;
2:
3: import java.util.ArrayList;
4:
5: import android.content.Intent;
6: import android.os.Bundle;
7: import android.os.RemoteException;
8: import android.view.KeyEvent;
9: import android.view.View;
10: import android.widget.AdapterView;
11: import android.widget.AdapterView.OnItemClickListener;
12: import android.widget.ArrayAdapter;
13: import android.widget.ListView;
```

```
14:
15: public class TestLedActivity extends BaseActivity implements OnItemClickListener {
16:
17:     private ListView mLedControlListView;
18:
19:     @Override
20:     protected void onCreate(Bundle savedInstanceState) {
21:         // TODO Auto-generated method stub
22:         super.onCreate(savedInstanceState);
23:         setContentView(R.layout.activity_testled);
24:
25:         mLedControlListView = (ListView)findViewById(R.id.lv_ledcontrols);
26:
27:         ArrayList<String> arrayList = new ArrayList<String>();
28:         arrayList.add("すべてオフ");
29:         arrayList.add("すべてオン");
30:
31:         ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
32:                 R.layout.listitems,
33:                 R.id.tv_name,
34:                 arrayList);
35:         mLedControlListView.setAdapter(adapter);
36:         mLedControlListView.setOnItemClickListener(this);
37:
38:     }
39:
40:     @Override
41:     public boolean onKeyDown(int keyCode, KeyEvent event) {
42:         if (keyCode == KeyEvent.KEYCODE_BACK) {
43:             Intent intent = new Intent(this, MainActivity.class);
44:             startActivity(intent);
45:         }
46:         return super.onKeyDown(keyCode, event);
47:     }
48:
49:     public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
50:         turnAllLedOff();
```

```
51:         try {
52:             Thread.sleep(300);
53:         } catch (InterruptedException e) {
54:             e.printStackTrace();
55:         }
56:
57:         switch (position) {
58:         case 0:
59:             turnAllLedOff();
60:             break;
61:         case 1:
62:             turnAllLedOn();
63:             break;
64:         default:
65:             break;
66:         }
67:
68:     }
69:
70:
71:     // beam 0:オフ、1:弱めにオン、2:中間の強さでオン、3:明るくオン
72:   // eyeCode – 0:既存の状態を維持、1:くるくる、2:眉毛点滅、3:眉毛停止、
73:   //4:完全オン、5:完全点滅、6:左右の効果、7:上下の効果、8:オフ
74:     // earCode – 0 : 既存の状態を維持、1:オン、2:オフ、3:点滅
75:     // colorCode – 1 : RED LED, 2 : BLUE LED, 3 : PURPLE (RED + BLUE) LED
76:   // mouthCode – 0 : 既存の状態を維持、1:中央をオン、2:三つをオン、3:中を点滅、
                              4:三つを点滅、5:話す、6:オフ
77:     private void turnAllLedOff() {
78:         if (myRobotApp.mBinder == null)
79:             return;
80:
81:         try {
82:             myRobotApp.mBinder.dmel_hri_set_mouth_led(getComponentName(), 6);
83:             myRobotApp.mBinder.dmel_hri_set_ear_led(getComponentName(), 2);
84:             myRobotApp.mBinder.dmel_hri_set_brow_beam(getComponentName(), 0);
```

```
85:             myRobotApp.mBinder.dmel_hri_set_eye_led(getComponentName(), 8, 1);
86:         } catch (RemoteException e) {
87:             e.printStackTrace();
88:         }
89:
90:     }
91:
92:     private void turnAllLedOn() {
93:         if (myRobotApp.mBinder == null)
94:             return;
95:
96:         try {
97:             myRobotApp.mBinder.dmel_hri_set_mouth_led(getComponentName(), 2);
98:             myRobotApp.mBinder.dmel_hri_set_brow_beam(getComponentName(), 2);
99:             myRobotApp.mBinder.dmel_hri_set_ear_led(getComponentName(), 1);
100:             myRobotApp.mBinder.dmel_hri_set_eye_led(getComponentName(), 4, 2);
101:         } catch (RemoteException e) {
102:             e.printStackTrace();
103:         }
104:     }
105: }
```

[Listing 6-10] TestLedActivity.java

今回の例のListView項目は、“すべてオフ”、そして “すべてオン”があります。それぞれのタッチはonItemClick()の57～65行目でturnAllLedOff()とturnAllLedOn()を呼び出して、頭のLEDをすべてオンまたはオフにします。
turnAllLedOff()とturnAllLedOn()は、次の4つのロボットサービス関数を呼び出します。ただし、引数の値に応じてLEDの動作に違いがあります。それぞれの関数は、次のとおりです。

dmel_hri_set_brow_beam

```
void dmel_hri_set_brow_beam(ComponentName cn,
                            int brightness)
                            throws android.os.RemoteException
```

額LEDを設定する。
Parameters:
cn – Componentの名前
brightness – 0:オフ、1:弱めにオン、2:中間の明るさでほどオン、3:明るくオン
Throws:
android.os.RemoteException

dmel_hri_set_eye_led

```
void dmel_hri_set_eye_led(ComponentName cn,
                          int eyeCode,
                          int colorCode)
                          throws android.os.RemoteException
```

目のLEDを設定する。
Parameters:
cn – Componentの名前
eyeCode – 0:既存の状態を維持、1:くるくる、2:眉毛点滅、3:眉毛停止、4:完全オン、5:完全点滅、6:左右の効果、7:上下の効果、8:オフ
colorCode – 1 : RED LED, 2 : BLUE LED, 3 : PURPLE (RED + BLUE) LED
Throws:
android.os.RemoteException

dmel_hri_set_ear_led

```
void dmel_hri_set_ear_led(ComponentName cn,
                          int earCode)
                          throws android.os.RemoteException
```

耳のLEDを設定する。
Parameters:
cn – Componentの名前
earCode – 0:既存の状態を維持、1:オン、2:オフ、3:点滅
Throws:
android.os.RemoteException

dmel_hri_set_mouth_led

```
void dmel_hri_set_mouth_led(ComponentName cn,
                            int mouthCode)
                            throws android.os.RemoteException
```

口のLEDを設定する。
Parameters:
cn - Componentの名前
mouthCode - 0:既存の状態を維持、1:中をON、2:三つをオン、3:中を点滅、4:三つを点滅、5:話す
Throws:
android.os.RemoteException

口、額、耳のLEDは単色でロボットサービス関数に色を指定する引数がありませんが、目LEDの制御関数であるdmel_hri_set_eye_led()のみ目の色の値を指定する引数があります。前に説明したように目は片方あたり8個ずつ赤、青のLEDがあります。(紫は赤と青のLEDが点灯して色が合成されて表示される色です。)目のLEDの場合32個のLEDをそれぞれ制御すると複雑さが増大するので、事前に定義した効果を中心にLEDを動作させます。

各関数の引数に対応するコード値は、[表6-3]と同じです。

コード値	目のLED	目の色	耳	口	額
0	状態を維持		状態を維持	状態を維持	オフ
1	くるくる	赤	オン	中をオン	弱
2	眉毛点滅	青	オフ	3つをオン	中
3	眉毛停止	紫	点滅	中を点滅	強
4	完全オン			3つを点滅	
5	完全点滅			話す	
6	左右の効果			オフ	
7	上下の効果				
8	オフ				

すべてのLEDを消す82～85行のturnAllLedOff()を表[6-3]を参照してみると次のとおりです。

```
myRobotApp.mBinder.dmel_hri_set_mouth_led(getComponentName(), 6);
myRobotApp.mBinder.dmel_hri_set_ear_led(getComponentName(), 2); myRobotApp.mBinder.
dmel_hri_set_brow_beam(getComponentName(), 0); myRobotApp.mBinder.dmel_hri_set_eye_
led(getComponentName(), 8, 1)
```

82行目のdmel_hri_set_mouth_led()関数の引数の値で指定されたコード値は6です。[表6-2]で口の6はオフです。したがって、口のLED 3個すべてオフになります。 83行目のhri_set_ear_led()のコード値は2です。このコードの値は、耳のLEDをすべて消します。次の行のdmel_hri_set_brow_beam()の額のLED制御関数にする値0は、額のLEDを消すことです。 最後の行のdmel_hri_set_eye_led()の引数は、目LEDコードの値が8であり、目の色コードの値は1です。目のLEDコード値8は、LEDを消します。したがって、目の色値1は影響を与えません。ただし、すべての目のLEDを消しても目の色の値は、任意の色で、必ず指定する必要があります。

同じように97～100行のturnAllLedOn()を[表6-2]から参照してみると、すべてのLEDを点灯することが分かります。

```
myRobotApp.mBinder.dmel_hri_set_mouth_led(getComponentName(), 2);
myRobotApp.mBinder.dmel_hri_set_brow_beam(getComponentName(), 2); myRobotApp.
mBinder.dmel_hri_set_ear_led(getComponentName(), 1); myRobotApp.mBinder.dmel_hri_set_
eye_led(getComponentName(), 4, 2);
```

今回の例でも、画面のリストに他の項目を追加して、様々なLED効果を直接作ってみてください。時々耳と口のLEDが制御がされない場合があります。この場合は、口の中のLEDが点滅します。これは、LEDを制御する頭ボードのエラー状態を示すものであり、ロボットの自律モードで自動的に回復します。

6.5 TTS 制御

TTSとはText-to-Speechの略で、テキストをロボットのボイスに変える機能です。たとえば、ロボットにテキストとして"こんにちは"を入力すると、ロボットが"こんにちは"と言う機能です。TTS機能を使用すると、ロボットが人のように話す効果を出すことができるので、現在ほぼすべてのロボットで使用されています。
Hovis Genieは日本語TTSエンジンを搭載しています。今回の例は、日本語のTTSを利用して、ロボットが話す機能を実行します。

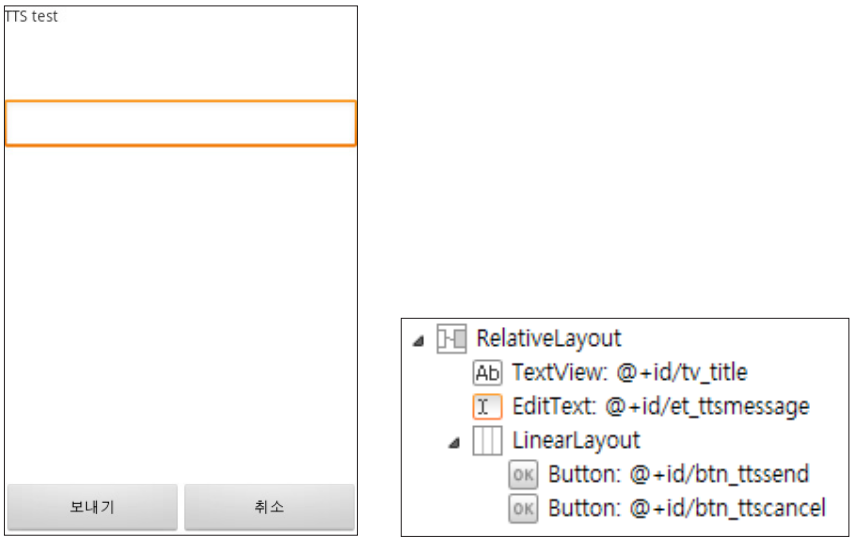
TTS制御ActivityにUIを構成するactivity_testtts.xmlを/ res / layoutに追加し、これに対応するTestTTSActivity.javaを/ src/ com/ dongburobot/ genieapidemoに追加します。続いてactivity_testtts.xmlを開き、[Listing6-11]を参照してコードを変更します。

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3:     android:layout_width="match_parent"
4:     android:layout_height="match_parent" >
5:
6:     <!-- テストのタイトル -->
7:     <TextView
8:         android:id="@+id/tv_title"
9:         android:layout_width="match_parent"
10:        android:layout_height="85dp"
11:        android:text="LED test"
12:        />
13:
14:    <!-- LED 制御リスト -->
15:    <EditText
16:        android:id="@+id/et_ttsmessage"
17:        android:layout_width="match_parent"
18:        android:layout_height="wrap_content"
19:        android:hint="文字を入力してください"
20:        android:maxLength="140"
```

```
20:         android:maxLength="140"
21:         android:layout_below="@id/tv_title"
22:         />
23:
24:         <!-- 開始と終了ボタン -->
24:         <!-- 開始と終了ボタン -->
25:     <LinearLayout
26:         android:layout_width="match_parent"
27:         android:layout_height="wrap_content"
28:         android:layout_alignParentBottom="true">
29:         <Button
30:             android:id="@+id/btn_ttssend"
31:             android:layout_width="160dp"
32:             android:layout_height="wrap_content"
33:             android:text="送る"
34:             />
35:         <Button
36:             android:id="@+id/btn_ttscancel"
37:             android:layout_width="160dp"
38:             android:layout_height="wrap_content"
39:             android:text="キャンセル"
40:             />
41:     </LinearLayout>
42: </RelativeLayout>
```

[Listing 6-7] activity_testmotion.xml

TTS制御画面のレイアウト構成は、最上位が<RelativieLayout>で構成され、各ウィジェットの配置によってウィジェット間の相対的な位置を指定します。<TextView>の場合、相対位置プロパティがないため、画面の左の上に配置されます。<EditText>は21行目に指定されたプロパティによって<TextView>の下に配置されます。<LinearLayout>は28行目のプロパティによって<RelativeLayout>の左の下に配置されます。ここで<LinearLayout>はサブの2つの<Button>を持ち、このウィジェットが<LinearLayout>の影響を受けて最下段に配置されます。この配置されたウィジェットの画面構成は、図6-8のとおりです。



[図 6-8] TTS 制御画面

TTS制御画面でユーザの入力は<EditText>を介して行われます。<EditText>ウィジェットは[図6-8]の黄色の枠線で囲まれた部分です。<EditText>にタッチをすると、キーボードが画面に出て、テキストの入力ができ、Backボタンを押すと、キーボードが消えます。
<EditText>の入力は20行目に記述されたandroid:maxLength="140"プロパティによって140文字まで入力できます。<EditText>のテキスト入力が多くて行が越えた場合android:layout_height="wrap_content"で<EditText>のサイズは内容だけ増加します。

例のためのUIが構成されています。TestTTSActivity.javaファイルを開き、ソースコードを次のように変更します。

1) BaseActivityを受ける

```
public class TestOmniwheelActivity extends BaseActivity …
```

2) ウィジェットに関するメンバ変数の宣言

```
private EditText mTTSEditText;
private Button mBtnTTSSend;
private Button mBtnTTSCancel;
```

3) onCreate() 作成

onCreate()でレイアウトXMLファイルをロードして、各ウィジェットのメンバー変数に、リソースを割り当てます。そして、各ボタンにタッチイベントを設定します。

4) OnClickListener インターフェイスを多重に受けた後、onClick()を作成

今回の例は、”送る”と“キャンセル”の二つのボタンがあります。このボタンを押した時に、タッチイベントを処理するためのソースコードを作成します。まずOnClickListenerインターフェイスを多重に受けます。

```
public class TestTTSActivity extends BaseActivity implements OnClickListener …
```

“implements OnClickListener”を追加して、“TestTTSActivity”上にマウスカーソルを置くと、onClick()を追加できます。

5) playTTS(), stopTTS() 作成

TTSを再生して停止するメソッドであるplayTTS()とstopTTS()を作成します。この2つのメソッドは、ユーザーが”送る”、“キャンセル”ボタンを押すと、onClick()によって呼び出されます。

6) onBackPressed() 作成

前の例まではonKeyDown()をオーバーライドして、Backボタンの処理をしました。onKeyDown()の場合、どのキーが押されても呼び出され、関数の内部で押されたボタンがBackボタンあることを確認して処理します。onBackPressed()はただBackボタンが押された場合にのみ呼び出されます。

ロボットAppは常に一つのActivityのみで維持しており、他のActivityがスタックにたまらないように設計されています。そのためBackボタンを押すと、Appを終了できますが、onBackPressed()は、これを防止して、メイン画面に戻る役割をします。

```
1: package com.dongburobot.genieapidemo;
2:
3: import android.content.Intent;
4: import android.os.Bundle;
5: import android.os.RemoteException;
6: import android.view.View;
7: import android.view.View.OnClickListener;
8: import android.widget.Button;
9: import android.widget.EditText;
10:
11: public class TestTTSActivity extends BaseActivity implements OnClickListener {
12:
13:     private EditText mTTSEditText;
14:     private Button mBtnTTSSend;
15:     private Button mBtnTTSCancel;
16:
17:     @Override
18:     protected void onCreate(Bundle savedInstanceState) {
19:         super.onCreate(savedInstanceState);
20:         setContentView(R.layout.activity_testtts);
21:
22:         mTTSEditText = (EditText)findViewById(R.id.et_ttsmessage);
23:
24:         mBtnTTSSend = (Button)findViewById(R.id.btn_ttssend);
25:         mBtnTTSCancel = (Button)findViewById(R.id.btn_ttscancel);
26:         mBtnTTSSend.setOnClickListener(this);
27:         mBtnTTSCancel.setOnClickListener(this);
28:     }
29:
30:     @Override
31:     public void onBackPressed() {
32:         super.onBackPressed();
33:         Intent intent = new Intent(this, MainActivity.class);
34:         startActivity(intent);
35:     }
36:
```

```
37:    public void onClick(View v) {
38:        switch (v.getId()) {
39:        case R.id.btn_ttssend:
40:            playTTS(mTTSEditText.getText().toString());
41:            break;
42:        case R.id.btn_ttscancel:
43:            stopTTS();
44:            break;
45:
46:        default:
47:            break;
48:        }
49:    }
50:
51:    private void playTTS(String msg) {
52:        stopTTS();
53:
54:        if (myRobotApp.mBinder == null)
55:            return;
56:
57:        try {
58:            myRobotApp.mBinder.dmel_tts_speak(getComponentName(), msg);
59:        } catch (RemoteException e) {
60:            e.printStackTrace();
61:        }
62:        return;
63:    }
64:
65:    private void stopTTS() {
66:        if (myRobotApp.mBinder == null)
67:            return;
68:
69:        try {
70:            myRobotApp.mBinder.dmel_tts_stop(getComponentName());
71:            Thread.sleep(200);
72:        } catch (RemoteException e) {
```

```
73:             e.printStackTrace();
74:         } catch (InterruptedException e) {
75:             e.printStackTrace();
76:         }
77:         return;
78:     }
79: }
```

[Listing 6-12] TestTTSActivity.java

今回の例の核心は、51～79のonClick()によって呼び出されるplayTTS()とstopTTS()です。

playTTS()はString型の文字列を引数として受け取ります。例では、ユーザーが"送る"　ボタンを押すと、EditTextに入力した文字列をString型に変換して引数として受け取ります。40行目に該当します。

```
40: playTTS(mTTSEditText.getText().toString());
```

これは、playTTS()58行目のTTSに関するロボットサービスの関数であるdmel_tts_speak()に渡され、テキストを音に変換します。

```
private void playTTS(String msg) {
...
58:     myRobotApp.mBinder.dmel_tts_speak(getComponentName(), msg);
…
}
```

dmel_tts_speak
void dmel_tts_speak(ComponentName cn, java.lang.String text) throws android.os.RemoteException TTS(Text-to-Speech)をPlayする。 Parameters: cn - Componentの名前 text - 音声に変換する文字列String ex)"こんにちは。ホビスジニです。" Throws: android.os.RemoteException

playTTS()52行目では、TTSの正確な動作を保証するためにstopTTS()を実行します。TTS命令の優先順位は、ロボットの資源をまず先に取得したTTSが高くなります。したがって、TTSがすでに実行されている場合は、ロボットサービスは、今後のTTS命令を無視します。

65～78行のstopTTS()は、現在実行されているTTS命令を中断する役割をします。 stopTTS()は内部的にdmel_tts_stop()を実行します。 dmel_tts_stop()は、TTSを実質的に中断するようにしますが、中断まで少し時間がかかることがあります。したがって、プログラムが安全に動作するようにThread.sleep(200)を利用して少しのディレイを与えます。

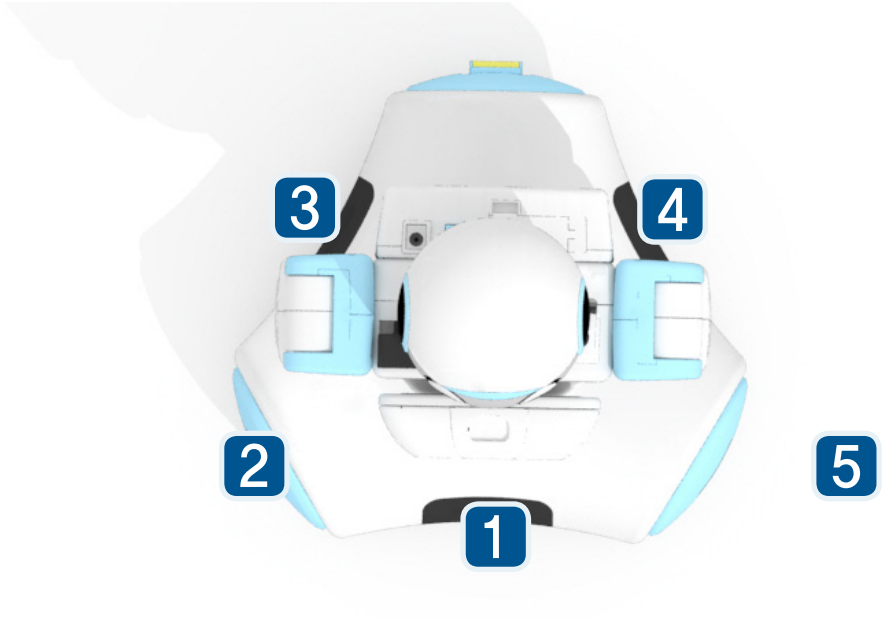
```
myRobotApp.mBinder.dmel_tts_stop(getComponentName());
Thread.sleep(200);
```

dmel_tts_stop
void dmel_tts_stop(ComponentName cn) throws android.os.RemoteException TTS(Text-to-Speech)を中止する。 Parameters: cn - Componentの名前 Throws: android.os.RemoteException

今までのロボットサービス関数を使用したplayTTS()とstopTTS()を介してTTS制御をしました。TTSは利用率が高く、ロボットをスマートに見せることができます。ただし、TTSに人と同じ発音を期待することには無理があります。この場合、テキストを発音に沿って書いてみるのもTTSを介したロボットの声の質を高める一つのノウハウになることがあります。

6.6 PSD センサ制御

Hovis Genieは周辺環境を認知するために5つのPSD(Position Sensitive Device)センサを搭載しています。PSDセンサは最大80cmまで周辺の情報を取得して障害物を認知します。ロボットはPSDセンサを使用して障害物を回避したり、複数の走行戦略を作成できます。



[図 6-9] 距離センサの配置

Hovis　Genieは正面を基準に時計の回りに1～5個のPSDセンサがあります。今回の例では、各PSDの値をリアルタイムで取得して画面に出力します。

PSDセンサ制御のUIファイルはactivity_testpsdsensor.xmlであり、そのソースコードは、TestPSDActivity.javaです。そのファイルをGenieApiDemoプロジェクトに追加します。
続いて、activity_testpsdsensor.xmlを開き、次のように変更します。

```
1: <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     xmlns:tools="http://schemas.android.com/tools"
3:     android:layout_width="match_parent"
4:     android:layout_height="match_parent" >
5:
6:     <!-- テストのタイトル -->
7:     <TextView
8:         android:id="@+id/tv_title"
9:         android:layout_width="match_parent"
10:         android:layout_height="85dp"
11:         android:text="PSD Sensor test"
12:         />
13:
14:     <LinearLayout
15:         android:layout_width="match_parent"
16:         android:layout_height="wrap_content"
17:         android:layout_below="@id/tv_title"
18:         android:orientation="vertical"
19:         >
20:         <!-- 1番のPSD -->
21:         <LinearLayout
22:             android:layout_width="match_parent"
23:             android:layout_height="wrap_content"
24:             >
25:             <TextView
26:                 android:layout_width="70dp"
27:                 android:layout_height="wrap_content"
28:                 android:text="前方"
29:                 />
30:             <TextView
31:                 android:id="@+id/tv_psd1"
32:                 android:layout_width="40dp"
33:                 android:layout_height="wrap_content"
34:                 />
35:             <SeekBar
36:                 android:id="@+id/seekbar_psd1"
```

```
37:             android:layout_width="200dp"
38:             android:layout_height="wrap_content"
39:             android:max="80"
40:             />
41:     </LinearLayout>
42:
43:     <!-- 2番のPSD -->
44:       <LinearLayout
45:        android:layout_width="match_parent"
46:        android:layout_height="wrap_content"
47:        >
48:        <TextView
49:             android:layout_width="70dp"
50:             android:layout_height="wrap_content"
51:             android:text="前方右側"
52:             />
53:        <TextView
54:             android:id="@+id/tv_psd2"
55:             android:layout_width="40dp"
56:             android:layout_height="wrap_content"
57:             />
58:        <SeekBar
59:             android:id="@+id/seekbar_psd2"
60:             android:layout_width="200dp"
61:             android:layout_height="wrap_content"
62:             android:max="80"
63:             />
64:     </LinearLayout>
65:
66:       <!-- 3番のPSD -->
67:       <LinearLayout
68:        android:layout_width="match_parent"
69:        android:layout_height="wrap_content"
70:        >
71:        <TextView
```

```
72:             android:layout_width="70dp"
73:             android:layout_height="wrap_content"
74:             android:text="後方右側"
75:             />
76:         <TextView
77:             android:id="@+id/tv_psd3"
78:             android:layout_width="40dp"
79:     }
80:
81:         <SeekBar
82:             android:id="@+id/seekbar_psd3"
83:             android:layout_width="200dp"
84:             android:layout_height="wrap_content"
85:             android:max="80"
86:             />
87:     </LinearLayout>
88:
89:         <!-- 4番のPSD -->
90:         <LinearLayout
91:          android:layout_width="match_parent"
92:          android:layout_height="wrap_content"
93:          >
94:          <TextView
95:             android:layout_width="70dp"
96:             android:layout_height="wrap_content"
97:             android:text="後方左側"
98:             />
99:          <TextView
100:             android:id="@+id/tv_psd4"
101:             android:layout_width="40dp"
102:             android:layout_height="wrap_content"
103:             />
104:          <SeekBar
105:             android:id="@+id/seekbar_psd4"
106:             android:layout_width="200dp"
107:             android:layout_height="wrap_content"
```

```
108:                android:max="80"
109:                />
110:        </LinearLayout>
111:
112:          <!-- 5番のPSD -->
113:          <LinearLayout
114:           android:layout_width="match_parent"
115:           android:layout_height="wrap_content"
116:           >
117:           <TextView
118:               android:layout_width="70dp"
119:               android:layout_height="wrap_content"
120:               android:text="前方左側"
121:               />
122:           <TextView
123:               android:id="@+id/tv_psd5"
124:               android:layout_width="40dp"
125:               android:layout_height="wrap_content"
126:               />
127:           <SeekBar
128:               android:id="@+id/seekbar_psd5"
129:               android:layout_width="200dp"
130:               android:layout_height="wrap_content"
131:               android:max="80"
132:               />
133:       </LinearLayout>
134:   </LinearLayout>
135:
136:   <!-- 開始と終了ボタン -->
137:   <LinearLayout
138:       android:layout_width="match_parent"
139:       android:layout_height="wrap_content"
140:       android:layout_alignParentBottom="true">
141:       <Button
142:           android:id="@+id/btn_start"
143:           android:layout_width="160dp"
```

```
144:            android:layout_height="wrap_content"
145:            android:text="開始"
146:            />
147:        <Button
148:            android:id="@+id/btn_stop"
149:            android:layout_width="160dp"
150:            android:layout_height="wrap_content"
151:            android:text="終了"
152:            />
153:    </LinearLayout>
154:
155: </RelativeLayout>
```

[Lisiting 6-11] activity_testpsdsensor.xml

今回のPSD制御画面では、UIのソースコードが多少長くなります。しかし、繰り返されるウィジェットがありますので構造は単純です。

最上位のタグは<RelativeLayout>です。<RelatvieLayout>のサブタグは7～12行目の<TextView>、14～134行目の<LinearLayout>、そして137～153行目の<LinearLayout>です。ここで<TextView>は"PSD Sensor test"を出力し、<LinearLayout>5つのPSDファイルの情報を表現し、最後の<LinearLayout>は"開始"と "終了"ボタンを表現します。

<RelativeLayout>に含まれるタグは、各ウィジェットの位置を相対的に表現します。7～12行目の<TextView>は、位置指定のプロパティがないため、親タグの左上に配置されます。14～134行目の<LinearLayout>は17行目のプロパティによって<TextView>下部に配置されます。137～153行の<LinearLayout>は140行目のプロパティによって、親である<RelativeLayout>の最下段に配置されます。

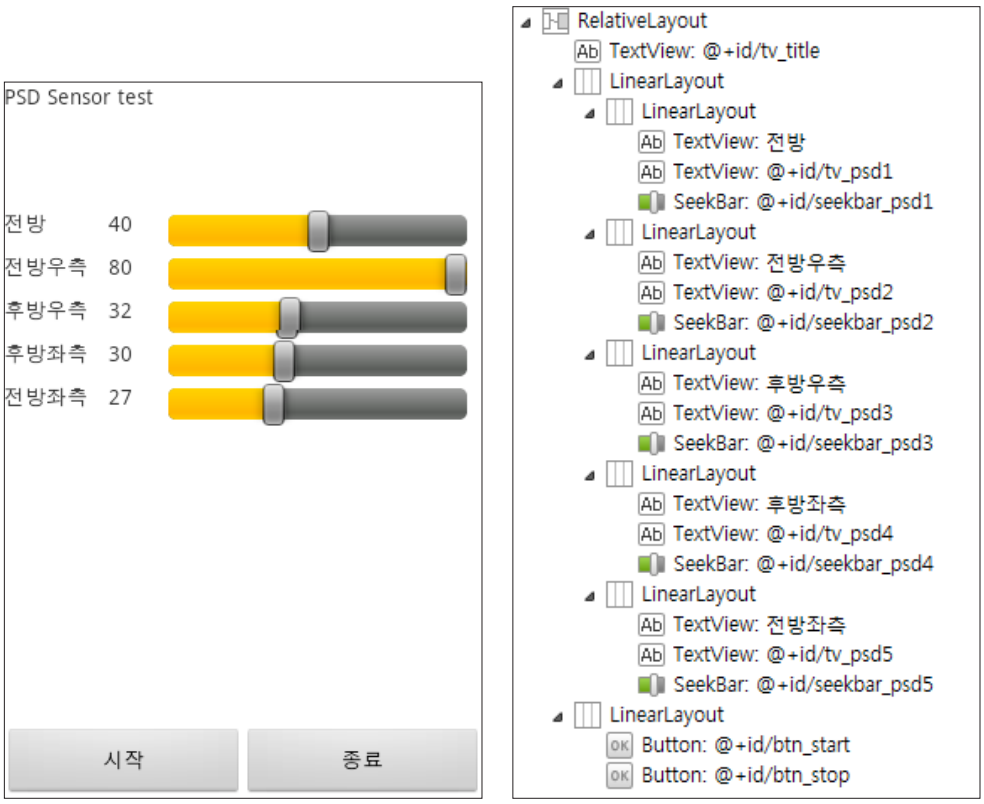
<RelativeLayout>のサブタグである14～134行目の<LinearLayout>は1番から5番のPSDセンサの情報を出力する<LinearLayout>5つを含んでいます。5つの<LinearLayout>は垂直に配置されますが、これは親タグである14～134行目の<LinearLayout>の18行目のプロパティによって決定されます。

5つの<LinearLayout>それぞれは、実際のPSDの情報を出力する3つのウィジェットである<TextView>、<TextView>、そして<SeekBar>をサブタグとして持ちます。それぞれの<LinearLayout>は水平、垂直の配置について設定をしていないので、含まれている3つのウィジェットは水平に配置されます。

5つの<LinearLayout>に含まれている3つのウィジェットは、IDのみが異なり、完全に同じなので、一つの<LinearLayout>のみ説明します。21～41行目の<LinearLayout>は1番のPSDについて情報を出力します。最初のサブタグである<TextView>は、センサの配置位置をテキストで表現します。二番目の<TextView>は、PSDセンサで測定された距離の値をcm単位で出力します。最後の<SeekBar>は距離値をグラフィカルに表現します。<SeekBar>にはandroid:max="80"という名前のプロパティがあります。これは<SeekBar>の表現範囲を0～80で指定し、80はPSDセンサが測定することができる最大距離の値です。

最後に、一番下の<RelativeLayout>のサブタグの<LinearLayout>があります。137～153行目の<LinearLayout>には二つの<Button>ウィジェットが含まれています。この<LinearLayout>では水平、垂直の配置プロパティが指定されていないので、ボタンは水平に配置されます。

構成されたUIは、[図6-10]のとおりです。



[図 6-10] PSD センサ制御画面

UIの設定が完了したので、TestPSDActivity.javaを開いてPSDセンサを制御するためのプログラミングを開始します。以下は、ソースコードの記述のガイドラインです。

1) BaseActivityを受ける

```
public class TestPSDActivity extends BaseActivity …
```

2) ウィジェットに関連するメンバ変数の宣言とHandler宣言

17～23行を参照してウィジェットとHandlerオブジェクトをメンバ変数として宣言します。mTvPSDは、各PSDの距離値をTextViewに出力するために使用されるTextViewオブジェクト変数であり、mSbPSDは距離値をSeekBarに出力するためのオブジェクト変数です。　mBtnStartとmBtnStopはボタンを制御するためのボタンオブジェクトです。

```
private TextView[ ] mTvPSD = new TextView[5];
private SeekBar[ ] mSbPSD = new SeekBar[5];
private Button mBtnStart;
private Button mBtnStop;
private Handler mHandler;
```

今回の例の核心は、Handlerの使用です。 Handlerは、PSDセンサ距離値をリアルタイムに取得して、画面を更新する役割をします。
HandlerはThreadと同様の機能をしています。Threadはプロセス内で独自の作業を並列的に実行します。今回の例の場合、PSDセンサの値を持続的に読んで、画面に更新する作業をリアルタイムに実行しながら、ユーザーのタッチ入力を処理する必要があるので、Threadの機能を使用する必要があります。
しかし、決定的なThreadの欠点は、GUI（Graphic User Interface）で安全ではないという点です。アンドロイド上でThreadで画面に関する作業をすると、多くの場合、エラーが発生します。これを防止するためにHandlerを使用しています。
Handlerの内部には、ThreadとThread Queueを持っています。詳細については、Androidの開発者サイトを参照してください。ここでは、”UIに関するリアルタイム処理は無条件にHandlerを使用する”という程度にまとめます。

3) onCreate() 作成

onCreate()の28～45行目でレイアウトXMLファイルをロードして、各ウィジェットのメンバー変数に、リソースを割り当てます。そして、各ボタンにタッチイベントを設定します。

onCreate()で最も重要な部分は、47～71行目のHandlerです。登録されたHandlerは、メッセージによって実行を開始したり、終了します。これは、ソースコードを作成した後に再度説明します。

4) OnClickListener インターフェイスを多重に受けた後、onClick()を作成

今回の例では、”開始”と “終了”の二つのボタンがあります。 “開始”は、PSDセンサの値の読み取りを開始する部分であり、”終了”は、これを終了します。onClickListenerインターフェイスを多重に受けて、onClick()を作成します。

```
public class TestPSDActivity extends BaseActivity implements OnClickListener …
```

Eclipseで”implements OnClickListener”を追加して、 “TestPSDActivity”上のマウスカーソルを置くと、onClick()を追加できます。 onClick()は、押されたボタンに応じて、Handlerにメッセージを送信したり、中断する役割をします。

5) onPause()を作成

onPause()はActivityが終了する時に呼び出されるメソッドです。今回のPSD制御Activityは終了する時にHandlerの実行を終了するコードを追加します。

6) onBackPressed()を作成する。

ロボットAppは常に一つのActivityのみ維持し、他のActivityがスタックにたまらないように設計されており、Backボタンを押すと、Appを終了ができますが、onBackPressed()は、これを防止して、メイン画面に戻る役割をします。

```
 1: package com.dongburobot.genieapidemo;
 2:
 3: import android.annotation.SuppressLint;
 4: import android.content.Intent;
 5: import android.os.Bundle;
 6: import android.os.Handler;
 7: import android.os.RemoteException;
 8: import android.util.Log;
 9: import android.view.View;
10: import android.view.View.OnClickListener;
11: import android.widget.Button;
12: import android.widget.SeekBar;
13: import android.widget.TextView;
14:
15: public class TestPSDSensorActivity extends BaseActivity implements OnClickListener {
16:
17:     private TextView[] mTvPSD = new TextView[5];
18:     private SeekBar[] mSbPSD = new SeekBar[5];
19:
20:     private Button mBtnStart;
21:     private Button mBtnStop;
22:
23:     private Handler mHandler;
24:
25:     @Override
26:     protected void onCreate(Bundle savedInstanceState) {
27:         super.onCreate(savedInstanceState);
28:         setContentView(R.layout.activity_testpsdsensor);
29:
30:         mTvPSD[0] = (TextView)findViewById(R.id.tv_psd1);
31:         mTvPSD[1] = (TextView)findViewById(R.id.tv_psd2);
32:         mTvPSD[2] = (TextView)findViewById(R.id.tv_psd3);
33:         mTvPSD[3] = (TextView)findViewById(R.id.tv_psd4);
34:         mTvPSD[4] = (TextView)findViewById(R.id.tv_psd5);
35:
36:         mSbPSD[0] = (SeekBar)findViewById(R.id.seekbar_psd1);
```

```
37:         mSbPSD[1] = (SeekBar)findViewById(R.id.seekbar_psd2);
38:         mSbPSD[2] = (SeekBar)findViewById(R.id.seekbar_psd3);
39:         mSbPSD[3] = (SeekBar)findViewById(R.id.seekbar_psd4);
40:         mSbPSD[4] = (SeekBar)findViewById(R.id.seekbar_psd5);
41:
42:         mBtnStart = (Button)findViewById(R.id.btn_start);
43:         mBtnStop = (Button)findViewById(R.id.btn_stop);
44:         mBtnStart.setOnClickListener(this);
45:         mBtnStop.setOnClickListener(this);
46:
47:         mHandler = new Handler() {
48:             @SuppressLint("HandlerLeak")
49:             public void handleMessage(android.os.Message msg) {
50:                 if (myRobotApp.mBinder == null)
51:                     return;
52:
53:                 double[] psdValue = null;
54:
55:                 try {
56:                     psdValue =
                                myRobotApp.mBinder.dmel_robot_get_obs(getComponentName());
57:                 } catch (RemoteException e) {
58:                     e.printStackTrace();
59:                 }
60:
61:                 //UI 反映
62:                 for (int i = 0; i < 5; i++) {
63:                     if ( (int)(psdValue[i]*100) != 100 ) {
64:                         mTvPSD[i].setText( String.valueOf((int)(psdValue[i]*100)) );
65:                         mSbPSD[i].setProgress((int)(psdValue[i]*100));
66:                     }
67:                 }
68:                 mHandler.sendEmptyMessageDelayed(0, 100);
69:             };
70:         };
71:     }
72:
```

```
73:     @Override
74:     public void onBackPressed() {
75:         super.onBackPressed();
76:
77:         Intent intent = new Intent(this, MainActivity.class);
78:         startActivity(intent);
79:     }
80:
81:     @Override
82:     protected void onPause() {
83:         if (mHandler != null) {
84:             if (mHandler.hasMessages(0))
85:                 mHandler.removeMessages(0);
86:
87:             mHandler = null;
88:         }
89:
90:         super.onPause();
91:     }
92:
93:     public void onClick(View v) {
94:         switch (v.getId()) {
95:         case R.id.btn_start:
96:             mHandler.sendEmptyMessage(0);
97:             break;
98:
99:         case R.id.btn_stop:
100:            if (mHandler.hasMessages(0))
101:                mHandler.removeMessages(0);
102:            break;
103:
104:        default:
105:            break;
106:        }
107:    }
108: }
```

[Listing 6-14] TestPSDActivity.java

TestPSDActivityが画面に表示される時に、onCreate()を呼び出します。onCreate()でHandlerを登録します。

```
mHandler = new Handler() {
…
public void handleMessage(android.os.aMessage msg) {
…
};
};
```

mHandlerにHandlerオブジェクトを割り当てる時には必ず再定義する必要があるhandleMessage()メソッドがあります。handleMessage()は、Handlerがメッセージを受信した時に呼び出され、メッセージを区別できるようにandroid.os.Message msgを引数として持ちます。

例では、登録されたHandlerにメッセージを送る主体は、“開始”ボタンです。“開始”ボタンの処理は、onClick()の96行目に定義されており、次のとおりです。

```
96: mHandler.sendEmptyMessage(0);
```

mHandler.sendEmptyMeesage()は、登録されたHandlerにメッセージを送信します。sendEmptyMessage()の引数の値である0は、Handlerで多数のメッセージを送信する時に、各メッセージを区別する役割をします。この例では、Handlerは1つのメッセージを送信するので、大きな意味はありません。ただし、メッセージが多数ある場合handleMessage()のmsg.whatで区別できます。sendEmptyMessage()に加えて、メッセージを送信する方法は、いくつかの方法があり、以下のリンクを参照します。 (http://developer.android.com/reference/android/os/Handler.html)

“開始”ボタンによってHandlerにメッセージが送信されると、handleMessage()が動作します。50行目でサービスが異常なくバインドされていることを確認した後、56行目でPSDセンサ距離値を返すロボットサービス関数dmel_robot_get_obs()を呼び出します。

dmel_robot_get_obs

double[ ] dmel_robot_get_obs(ComponentName cn)
throws android.os.RemoteException
前方検出PSDセンサの値を返している。前方から時計回りに0、1、2、3、4
Parameters:
cn - Componentの名前
Returns:
前方感知センサ5つの値をdouble[0]～double[4]に戻す
Throws:
android.os.RemoteException

dmel_robot_get_obs()は、サイズが5であるdouble型の配列を返します。 1番～5番PSDセンサの値は、それぞれ0～4の配列に代入されます。この時に各配列に返される値は、単位がmの距離の値です。各PSDセンサの値の情報を出力するTextViewとSeekBarでは、cm単位の距離の値を使用します。62～67行では、mをcmに変換します。

handleMessage()の68行目では、再びHandlerにメッセージを送って、今までと同じようなルーチンを繰り返すようにします。これは、継続的にPSDセンサの値を画面に更新するためです。

```
68: mHandler.sendEmptyMessageDelayed(0, 100);
```

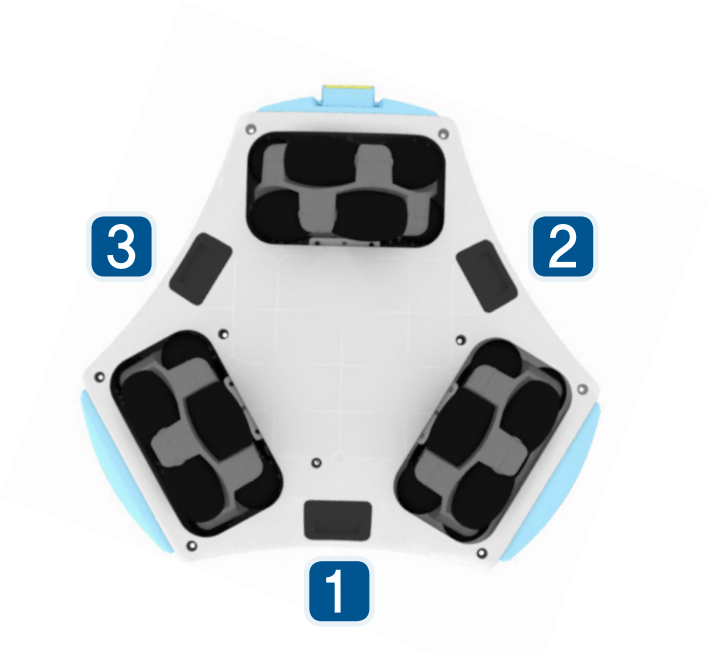
mHandlerにHandlerオブジェクトを割り当てる時には必ず再定義する必要があるhandleMessage()メソッドがあります。 handleMessage()は、Handlerがメッセージを受信した時に呼び出され、メッセージを区別できるようにandroid.os.Message msgを引数として持ちます。

例では、登録されたHandlerにメッセージを送る主体は、 “開始”ボタンです。 “開始”ボタンの処理は、onClick()の96行目に定義されており、次のとおりです。

```
100:    if (mHandler.hasMessages(0))
101:        mHandler.removeMessages(0);
```

100行目では、Handlerに現在0で区切られたメッセージが配信されていることを確認します。もしメッセージがある場合は、101行目からメッセージを削除します。以降では、これ以上のメッセージがHandlerに渡されず、handleMessage()も呼び出されないためPSD値の取得と画面の更新を終了します。

Hovis Genieには下の感知のための3つのPSDセンサが取り付けられており、ロボット上の配置は次のとおりです。



[図 6-11] 下の感知PSDセンサの配置

下の感知センサのロボットサービス関数は、dmel_robot_get_cliff() です。

dmel_robot_get_cliff
double[] dmel_robot_get_cliff(ComponentName cn) throws android.os.RemoteException 床の感知PSDセンサの値を返します。前方から時計回りに0、1、2 Parameters: cn - Componentの名前 Returns: 床のセンサの3つの値をdouble[0]～double[2]に戻す Throws: android.os.RemoteException

この部分の例については、今回の例と非常に似ているので省略します。
GenieApiDemo XMLファイルとJavaファイルを追加して、新しいActivityを作成して練習してみてください。そしてMainActivityのGridView項目の新しいアイコンとテキストを追加することもしてみてください。最後に、AndroidManifest.xmlに新しいActivity追加も忘れないでください。

6.7 タッチセンサ制御

今回の例は、Hovis　Genieのタッチセンサ制御です。タッチセンサは、ロボットとユーザーとの間の相互作用のために活用されます。ロボットにはタッチセンサは3個あり、センサの位置は、両手のひらと頭です。



[図 6-12] タッチセンサの配置（両手のひら、頭）

タッチセンサ制御ActivityにUIを構成するactivity_testtouchsensor.xmlを/ res / layoutに追加し、これに対応するTestTouchSensorActivity.javaを/ src/ com/ dongburobot/ genieapidemoに追加します。

今回の例では、両手のひらと頭にタッチしているかどうかをTTSで知らせします。例を実行するとタッチしたかどうかをUIに表示しません。そうすることでUIをもっと簡単にできます。<RelativeLayout>を最上位にして<TextView>に”Touch　test”を表示して、現在のActivityがタッチセンサの例であることを示すだけです。activity_testtouchsensor.xmlのソースコードを次のように変更します。

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3:     android:layout_width="match_parent"
4:     android:layout_height="match_parent" >
5:
6:     <!-- テストのタイトル -->
7:     <TextView
8:         android:id="@+id/tv_title"
9:         android:layout_width="match_parent"
10:         android:layout_height="85dp"
11:         android:text="Touch test"
12:         />
13:
14: </RelativeLayout>
```

[Listing 6-15] activity_testtouchsensor.xml



[図 6-13] タッチセンサ制御画面

続いて、TestTouchSensorActivity.javaファイルを開き、ソースコードを次のように作成します。

1) BaseActivityを受ける

```
public class TestTouchSensorActivity extends BaseActivity …
```

2) Handler メンバ変数のオブジェクト宣言

```
private Handler mHandler;
```

3) onCreate() 作成

onCreate()でsetContentView()を利用してレイアウトのXMLファイルをロードします。今回の例ではUIは実行する時に画面の変更がないためactivity_testtouchsensor.xmlに宣言されたリソースのウィジェットメンバ変数の割り当てはありません。

前の節でPSDセンサの値をリアルタイムに取得するためにHandlerが使用されているように、今回の例でも、各タッチセンサがタッチされたかどうかを判断するためにHandlerを使用してリアルタイムにセンサを観察します。　Handlerのオブジェクトを mHandlerに割り当ててhandleMessage()をオーバーライドします。

4) onPause() 作成

Activityの終了する時にHandler終了を保証するためにHanlderのメッセージがある場合削除します。

5) onBackPressed() 作成

ロボットAppは常に一つのActivityのみ維持し、他のActivityがスタックにたまらないように設計されており、Backボタンを押すと、Appを終了ができますが、onBackPressed()は、これを防止して、メイン画面に戻る役割をします。

```
1: package com.dongburobot.genieapidemo;
2:
3: import android.content.Intent;
4: import android.os.Bundle;
5: import android.os.Handler;
6: import android.os.Message;
7: import android.os.RemoteException;
8:
9: public class TestTouchSensorActivity extends BaseActivity {
```

```
10:
11:    private Handler mHandler;
12:
13:    @Override
14:    protected void onCreate(Bundle savedInstanceState) {
15:        super.onCreate(savedInstanceState);
16:        setContentView(R.layout.activity_testtouchsensor);
17:
18:        mHandler = new Handler() {
19:            @Override
20:            public void handleMessage(Message msg) {
21:                super.handleMessage(msg);
22:
23:                if (myRobotApp.mBinder == null)
24:                    return;
25:
26:                boolean[] isHandTouched = null;
27:                boolean isHeadTouched;
28:                boolean isNothingTouched = false;
29:
30:                try {
31:                    isHandTouched =
             myRobotApp.mBinder.dmel_hri_get_hand_touch_info(getComponentName());
32:                    isHeadTouched =
            myRobotApp.mBinder.dmel_hri_get_head_touch_info(getComponentName());
33:
34:                    // 同時にタッチすることは考慮していない。
35:                    if (isHandTouched[0]) {
36:                        myRobotApp.mBinder.dmel_tts_speak(getComponentName(),
                                                        “左手がタッチされています”);
37:                    } else if (isHandTouched[1]) {
38:                        myRobotApp.mBinder.dmel_tts_speak(getComponentName(),
                                                        “右手がタッチされています”);
39:                    } else if (isHeadTouched) {
40:                        myRobotApp.mBinder.dmel_tts_speak(getComponentName(),
                                                        “頭がタッチされています”);
```

```
41:                     } else {
42:                         isNothingTouched = true;
43:                     }
44:                 }
45:                 catch (RemoteException e) {
46:                     e.printStackTrace();
47:                 }
48:
49:                 if (isNothingTouched)
50:                     mHandler.sendEmptyMessageDelayed(0, 100);
51:                 else
52:                     mHandler.sendEmptyMessageDelayed(0, 3000);
53:             }
54:         };
55:
56:         mHandler.sendEmptyMessage(1000);
57:     }
58:
59:     @Override
60:     protected void onPause() {
61:         super.onPause();
62:
63:         if (mHandler != null) {
64:             if (mHandler.hasMessages(0))
65:                 mHandler.removeMessages(0);
66:
67:             mHandler = null;
68:         }
69:     }
70:
71:     @Override
72:     public void onBackPressed() {
73:         super.onBackPressed();
74:
75:         Intent intent = new Intent(this, MainActivity.class);
76:         startActivity(intent);
77:     }
78: }
```

[Listing 6-16] TestTouchSensorActivity.java

MainActivityでタッチセンサアイコンをタッチするとTestTouchSensorActivityが開始します。この時に呼び出されるonCreate()でHandlerのオブジェクトを割り当て、handleMessage()を定義します。そしてonCreate()の最後の56行目でHandlerにメッセージを送って定義されたhandleMessage()が実行されます。

handleMessage()31～32行目のロボットサービス関数dmel_hri_get_head_touch_info()とdmel_hri_get_hand_touch_info()は、ロボットの頭と両手のひらがタッチされたかどうかを返します。各関数の元の形は次のとおりです。

dmel_hri_get_head_touch_info

```
boolean dmel_hri_get_head_touch_info(ComponentName cn)
                                throws android.os.RemoteException
頭がタッチされたかどうかを返す。
Parameters:
cn - Componentの名前
Returns:
頭がタッチされた時true、されてなければfalseを返す
Throws:
android.os.RemoteException
```

dmel_hri_get_hand_touch_info

```
boolean[] dmel_hri_get_hand_touch_info(ComponentName cn)
                                throws android.os.RemoteException
両手がタッチされたかどうかを返す。
Parameters:
cn - Componentの名前
Returns:
タッチされた時true、されてなければfalseを返します。boolean[2](boolean[0]:左手のタッチ、boolean[1]:右手のタッチ)
Throws:
android.os.RemoteException
```

35～43行目では、各関数が返す値を割り当てたisHandTouched[]とisHeadTouched値を調査してタッチされているかどうかをTTSで通知します。ただし、このコードは、ユーザーがタッチセンサを同時にタッチした場合の処理はしません。もし同時にタッチすると、左手、右手、頭の順に優先してTTSを出力します。

ロボットが何のタッチも感知しないと、42行目によってisNothingTouchedがtrueに設定されます。この変数は、49～52行目のif文で使用されますが、isNothingTouchedがtrueの場合100ms後に再度センサの値を読み取るようにHandlerにメッセージを送ってリアルタイムにセンサタッチを確認します。

一方、isNothingTouchedがfalseの場合、どのタッチセンサもタッチを感知したという意味です。この場合、タッチされたセンサによって合うTTSを出力します。 52行目の3000msの時間は再びHandlerにメッセージを送信する前にTTSの出力が終了するまでの時間を保証します。

Dongbu Robotのホビスジニとアプリを利用して、
ロボットプログラミングの原理を基本から楽しく習得できます。

01. アンドロイドロボットHovis Genieの紹介

1.1 Hovis 製作の背景
1.2 Hovis Genieの特徴
(1) Hovis Lite
(2) Hovis Genie
1.3 Hovis Genieのハードウェア構成
(1) ロボット本体
(2) 顔
(3) ホビスジニの仕様
1.4 Hovis Genieのソフトウェアの特徴
(1) MPSU (DRC-005T), OPSU (DRC-004TO), MID Side Board - ファームウェア
(2) MID - Android
1.5 Hovis Genieのアプリケーションを実行する
(1) メイン画面
(2) 自律行動
(3) 遠隔制御
(4) 朝の起床情報/童謡童話などの情報コンテンツのロボットコンテンツ化
(5) ソフトウェア提供の目的と活用

02. アンドロイドとロボットの開発環境を準備する

2.1 JDK インストール
2.2 アンドロイドSDKインストール
2.3 Eclipseのインストールと環境設定

03. アンドロイドロボット(ジニ)プログラミング

3.1 HelloGenie プロジェクトの作成
3.2 HelloGenie プロジェクトの実行
3.3 HelloGenie プロジェクトの基本的な構成
(1) AndroidManifest.xml
(2) MainActivity.javaとactivity_main.xml

04. Hovis Genieロボットのプログラミングを開始する

4.1 GenieApiDemoプロジェクトの作成
4.2 GenieApiDemoApplicationクラス
4.3 ロボットシステム用BaseActivityの作成

05. Hovis Genie API

5.1 駆動部関数
5.2 ナビゲーション関数
5.3 センサの関数
5.4 TTS(Text-to-Speech) 関数
5.5 サウンド(効果音)関数
5.6 マルチメディア(オーディオとビデオ)関数
5.7 モーションとサーボモーターに関連する関数
5.8 頭のLED制御に関する関数

06. Hovis Genie 基本的な例

6.1 メイン画面の構成
6.2 駆動部の制御
6.3 モーション制御
6.4 頭のLED制御
6.5 TTS制御
6.6 PSDセンサ制御
6.7 タッチセンサ制御



Dongbu Robot

(株)Dongbu Robot www.dongburobot.com

本 社 : 11th Floor, Bucheon Techno Park 401, Yakdae-dong, Wonmi-gu, Bucheon-city, Gyunggi-do 420-734, Korea,
TEL : +82-32-329-5551(ext.311), FAX : +82-32-329-5569, E-MAIL : robotsales@dongbu.com
工 場 : 27, 6 Gil, 4 Sandan, Jiksan-eup, Seobuk-gu, Cheonan-city, Chungcheongnam-do 331-814, Korea,
TEL : +82-41-590-1700,FAX : +82-41-590-1701

ISBN 978-89-98873-12-7