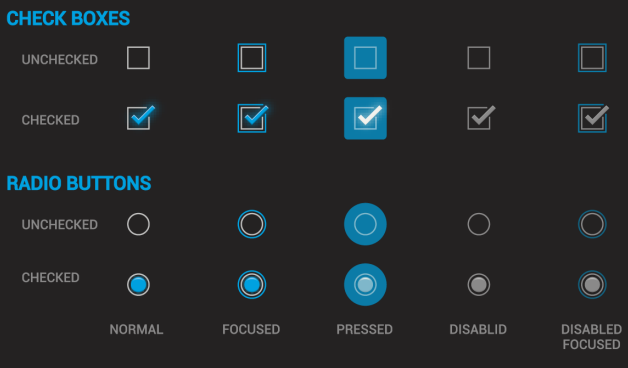
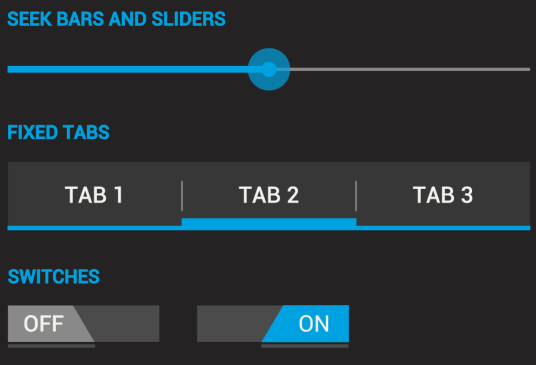
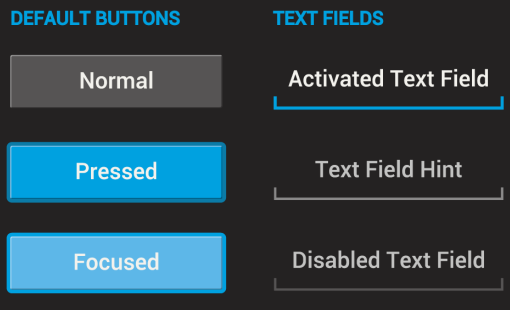
Android UI Cookbook for 4.0

あんざいゆき [著]

ICS (Ice Cream Sandwich) アプリ開発術







» 単に動けば良い、仕様が満たされていれば良いということでなく、エレガントさ、見た目の良さに関する一定の考えを結果に反映させる姿勢は、筆者ならではです。そうしたセンスが反映された技術書は、おそらく唯一の存在でしょう。

バイドゥ株式会社
アンドロイド向け日本語入力システム Simeji 開発担当 足立昌彦 デザイン担当 矢野りん

インプレスジャパン

WARNING

はじめにお読みください

Android UI Cookbook for 4.0 ICS

(Ice Cream Sandwich)

アプリ開発術

あんざいゆき [著]

インプレスジャパン

本書で使用したサンプルプログラムのソースは以下の URL からダウンロード可能です。

http://www.impressjapan.jp/books/3174

※ 本書で用いた Roboto type family や Ice Cream Sandwich の UI elements は、the Android Open Source Project が作成、提供しているコンテンツをベースに複製したもので、クリエイティブ・コモンズの表示 2.5 ライセンスに記載の条件に従って使用しています。本書内でも紹介している Android Design の Web サイト（http://developer.android.com/intl/ja/design/index.html）を参照してください。

※ 本書の内容は、2012 年 1 月の執筆時点のものです。本書で紹介した製品／サービスなどの名前や内容は変更される可能性があります。あらかじめご注意ください。

※ 本書の内容に基づく実施・運用において発生したいかなる損害も、著者ならびに株式会社インプレスジャパンは一切の責任を負いません。

※ 本書に登場する会社名、製品名、サービス名は、各社の登録商標または商標です。本文中では ®、TM、© マークは明記しておりません。

あなたの価値を高める 1 冊

Android の普及は誰もが認めるところとなりました。そのため、スマホ開発は iPhone だけ、Android だけという区切りではなく、" 両方 " が求められます。特に Android は端末の多様性のため、画面デザインにちょっとした癖があります。でもそれは正しい知識と技術があれば簡単に乗り越えられます。本書にはその知識と技術がふんだんに盛り込まれています。「Android ではそのようなデザインはできません」と言ってしまって自分の価値を落としている開発者さん。「Android のデザインって複雑だし面倒そう」と言ってしまって手を出していないデザイナーさん。そんなことを言ってしまっている有象無象から 1 歩も 2 歩も抜け出すチャンスが本書にあります。

バイドゥ株式会社　足立昌彦
（Simeji 開発担当 /Google API Expert（Android）/a.k.a adamrocker）

私は以前、あんざいさんと一緒にかなりの規模のアプリ開発を手がけた経験があります。あらゆる実装手段を知っている彼女は非常に短期間で、込み入った画面を仕上げていきました。Android のアプリ開発ではレイアウトに最も手間と時間がかかるものだということを知っていたので、とても驚きました。彼女の開発スピードは彼女の知識量そのものだと感じたものです。その知識の多くが、本書にまとめてあります。本書を手にするということは、彼女の開発スピードを手にする、ということかもしれません。もう 1 つ彼女のすごいところは、審美性に配慮した実装を指示なしでやるところです。単に動けばよい、仕様が満たされていればよいということでなく、エレガントさ、見た目のよさに関する一定の考えを結果に反映させる姿勢は、彼女ならではです。そうしたセンスが反映された技術書は、おそらく唯一の存在でしょう。

バイドゥ株式会社　矢野りん
（Simeji デザイン担当 /Android 女子部部長）

2011 年 5 月、Google I/O 2011 で Ice Cream Sandwich（以下、ICS もしくは 4.0）が秋頃にリリースされるという発表を聞いて私はワクワクしていました。タブレットとハンドセットの両方に対応した SDK になるという話だったからです。また、Honeycomb（3.x）はリリース後もソースコードが公開されることがありませんでしたが、ICS では公開されるという話もありました。ICS では Honeycomb の機能をどのようにハンドセットに取り込むのか、新しい機能は追加されるのか、どの機能がなくなるのか。いろいろ予想していました。　正直な感想として、ICS では目立った新しい機能はないなと感じました。ソフトキーボードが作れるようになったり、AppWidget が作れるようになったり、ライブウォールペーパーが作れるようになったり、C2DM や NFC が紹介されたときのような、特定の何かができるようになったというものではなかったのです。

その代わり、もっと根本的な部分が大きく変わりました。アプリの設計自体にも大きく関わってくる UI です。

Honeycomb をハンドセット向けに拡張したと言われるだけあって、Honeycomb と ICS では UI に大きな違いはありません。しかし、2.x 以前と 4.0 ではるかに違います。1 つの apk で 2.x と 4.0 の両方に対応しようと考えている方もいるでしょう。しかし、個人的にはあまりお勧めできません。

4.0 で追加されたリッチな UI 機能を使わないのはもったいないですし、2.x と 4.0 ではシステムの標準的な振る舞いが異なるので、本来はそれぞれのシステムと同じ振る舞いをするようにアプリを作るべきです。また、Support package を使った後方互換性の維持には限界があります。フラグメントは Support package 使うことで 1.6 からも対応できますが、アクションバーは完全にサポートされているわけではありません。さらに、リフレクションや API レベルによる分岐が多いコードになりがちなので、メンテナンス性も下がります。

現状としては、通信など共通できる部分をライブラリプロジェクトとし、2.x と 4.0 向けで別々のプロジェクトを作り、Android Market の Multiple apk 機能を使ってリリースするのが現実的ではないかと思います。

本書では、4.0 向けのアプリを作る上で必要となる UI に関連する項目を取り上げています。第 1 章では、ICS での UI の基礎、新しく追加された項目やウィジェット、4.0 向けのアプリを作る上で押さえておきたいポイント、4.0 向けのアプリのデザインガイドラインである Android Design（http://

developer.android.com/design/index.html）の内容をピックアップして紹介しています。第 2 章以降は、フラグメントやアクションバー、アニメーション、ドラッグ & ドロップなど、3.0 以降に追加された機能のうち、4.0 向けのアプリの UI でも基礎になっている項目について解説しています。

Android Design をはじめ、さまざまなサイトや書籍でモバイルデバイスの UI パターンについて解説されています。しかし、どんなにすばらしいデザインの UI でも、それを実現するコードが書けなければ意味がありません。本書がみなさんのすばらしいアプリの UI を実現する一助になれば幸いです。

あんざいゆき

＊本書は各章を個別に読み進められるような構成になっています。そのため、一部のクラスやメソッドについて説明が重複しているところもありますが、ご了承ください。

＊本書で使用したサンプルプログラムのソースは以下の URL からダウンロード可能です。
http://www.impressjapan.jp/books/3174

＊本書を通して作成した各種サンプルを収録したデモアプリを以下のサイトで公開しています。
https://market.android.com/details?id=yanzm.products.icsuicookbook

下記の QR コードからもアクセスできます。
QR コードを読み込み、マーケットアプリを起動させてください。

Android UI Cookbook for 4.0 | CONTENTS

第 6 章　アプリケーションウィジェット

第 7 章　コピー&ペースト

第 8 章　ドラッグ&ドロップ

第 9 章　ローダー

第1章

Android UI Cookbook for 4.0

ON

Android 4.0 (ICS)

2011 年 10 月、Google は Android の最新版である Ice Cream Sandwich（以下、ICS もしくは 4.0）を発表しました。これまでは、ハンドセット向けの SDK として Android 2.x が、タブレット向け SDK として Android 3.x が提供されていましたが、1 つの SDK でハンドセットとタブレットに対応するためにこれらを統合して作られたのが Android 4.0、通称 ICS です。

ICS は Android 3.0 の特徴を多く引き継いでいます。まるで Android 3.0 をハンドセット向けに拡張したかのようです。

そのため、フラグメントやアクションバーなど、Android 3.0 からサポートされた機能が継承されている一方で、Android 2.x で利用できていた方法が使えなくなっていることもあります。たとえば、Android 2.x で多くのアプリケーションに利用されてきた、複数のアクティビティを使ったタブレイアウトは、Android 4.0 では利用できません。代わりに、複数のフラグメントを使ってタブレイアウトを実現するようになっています。

このように、Android 4.0 では、アプリケーション設計の基礎的な部分に大きな変更が入りました。本書では、フラグメントなどのアプリケーションの基礎部分や、アクションバーなどのレイアウトに関連する部分を深く取り上げます。

1.1　システムユーザーインタフェース

Android 4.0 のシステムユーザーインタフェース（UI）は、全体的に半透明が多用されたデザインになっています。ディズニー映画「トロン：レガシー」に登場する青い半透明のデザインによく似ています。また、これまでハンドセットでハードキーとして割り当てられていたホームキーやバックキーがソフトキーとして画面上に配置されるようになりました。

1.1.1　UI 上のバー

Android 3.0 では、システムの UI としてホームキーやバックキーなどのボタンとステータスバーが一体になったバーが下部に表示されるようになりました。Android 4.0 では、これがハンドセットにも拡張され、ホームキーやバックキーなどのボタンが下部にバーとして、ステータスが上部にバーとして表示される UI になりました（図 1-1）。これらのバーにはそれぞれ名称があります。

- **ステータスバー**
  左側にノーティフィケーションが表示され、右側に時間、バッテリーレベル、電波強度など

図 1-1　Android 4.0 のシステム UI

のステータスが表示されるバー。下方向にスワイプすることでノーティフィケーションの詳細を見ることができます。Android 3.0 以前からもある UI です。

- ナビゲーションバー
  Android 4.0 でハンドセット向けに新しく採用されたバー。ホームキーやバックキーなど、これまでのハードキーがないデバイスでのみ表示されます。ここには、バック、ホーム、最近のアプリケーション用のコントロールが表示されます。Android 2.3 以前向けに書かれたアプリケーションでは、メニュー用のコントロールも表示されます（図 1-2）。

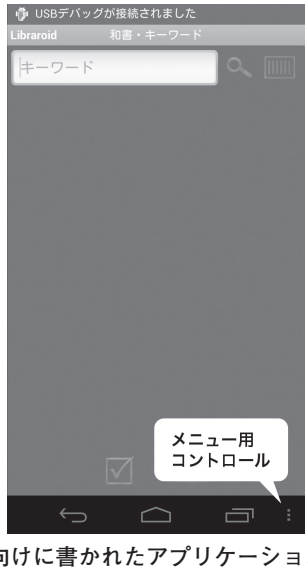


図 1-2　Android 2.x 向けに書かれたアプリケーションでのナビゲーション

- システムバー
  タブレットの画面用のナビゲーションバーとステータスバーが組み合わされたバー。Android 3.0 から採用されている UI です。

1.1.2　システム UI の状態

ナビゲーションバーなどのシステム UI の状態には次の 3 つがあり、それぞれ View クラスの定数が割り当てられています。それぞれの状態を切り替えるには、View クラスの setSystemUiVisibility() メソッドを使って、割り当てられている定数を指定します。

- システム UI が常に表示されている状態
  デフォルトの状態であり、ナビゲーションバーやステータスバーなどが常に表示されます。
  View クラスの定数：SYSTEM_UI_FLAG_VISIBLE
- システム UI の見た目が控えめになっている状態
  ナビゲーションバーやステータスバーは表示されているものの、内部のコントロールが目立たなくなっている状態です。ユーザーがアプリケーション部分の画面に触れてもこの状態は解除されませんが、ナビゲーションバーやステータスバーに触れた場合は直ちにナビゲーションコントロールが目立つ状態に戻ります。ゲーム、電子書籍リーダー、ビデオプレイヤーなど、没入型のアプリケーションに適しています。
  View クラスの定数：SYSTEM_UI_FLAG_LOW_PROFILE
- システムナビゲーションが一時的に非表示になっている状態
  システムナビゲーションが非表示になるため、SYSTEM_UI_FLAG_LOW_PROFILE よりも目立たなくなります。ハンドセットではナビゲーションバーが一時的に非表示になりますが、ステータスバーは非表示になりません。Window クラスのフラグである FLAG_FULLSCREEN や FLAG_LAYOUT_IN_SCREEN と組み合わせることで、ステータスバーも非表示にした全画面表示が実現できます。ただし、ナビゲーションコントロールは非常に重要なため、ユーザーが画面に触れるなどのインタラクションが発生すると直ちにナビゲーションバーが表示されます。
  View クラスの定数：SYSTEM_UI_FLAG_HIDE_NAVIGATION

では、これらの状態を切り替えるコードを書いてみましょう。

リスト 1-1　MainActivity.java

```
public class MainActivity extends Activity implements View.OnClickListener{

  @Override
```

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main1_1);

  findViewById(R.id.low_profile_btn).setOnClickListener(this);
  findViewById(R.id.hide_navigation_btn).setOnClickListener(this);
  findViewById(R.id.visible_btn).setOnClickListener(this);

  ToggleButton toggleButton = (ToggleButton)
                              findViewById(R.id.fullscreen_btn);
  toggleButton.setOnCheckedChangeListener(
      new CompoundButton.OnCheckedChangeListener() {

    @Override
    public void onCheckedChanged(CompoundButton buttonView,
                                 boolean isChecked) {
      Window window = getWindow();
      WindowManager.LayoutParams winParams = window.getAttributes();

      if (isChecked) {
        // ステータスバーを非表示
        window.addFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN);
      } else {
        // ステータスバーを表示
        window.clearFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN);
      }
      window.setAttributes(winParams);
    }
  });
}

@Override
public void onClick(View v) {
  Window window = getWindow();
  View view = window.getDecorView();

  int id = v.getId();
  int visibility = 0;

  if(id == R.id.hide_navigation_btn) {
    // 一時的にナビゲーションバーを非表示
    visibility = View.SYSTEM_UI_FLAG_HIDE_NAVIGATION;
  }
  else if(id == R.id.low_profile_btn) {
    // ナビゲーションコントロールを暗くする
    visibility = View.SYSTEM_UI_FLAG_LOW_PROFILE;
  }
  else if(id == R.id.visible_btn) {
    // ナビゲーションバーおよびコントロールを表示
    visibility = View.SYSTEM_UI_FLAG_VISIBLE;
  }

  view.setSystemUiVisibility(visibility);
```

```
    }
}
```

レイアウトはリスト 1-2 のようになっています。

リスト 1-2　res/layout/main1_1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical"
              android:padding="16dip">

  <LinearLayout android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="horizontal"
                android:gravity="center_vertical">

    <TextView android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="FLAG_FULLSCREEN" />

    <ToggleButton android:id="@+id/fullscreen_btn"
                  android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
                  android:layout_margin="16dip" />
  </LinearLayout>

  <Button android:id="@+id/hide_navigation_btn"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:text="SYSTEM_UI_FLAG_HIDE_NAVIGATION" />

  <Button android:id="@+id/low_profile_btn"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:text="SYSTEM_UI_FLAG_LOW_PROFILE" />

  <Button android:id="@+id/visible_btn"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:text="SYSTEM_UI_FLAG_VISIBLE" />

</LinearLayout>
```

結果は図 1-3 ～ 1-6 のようになります。

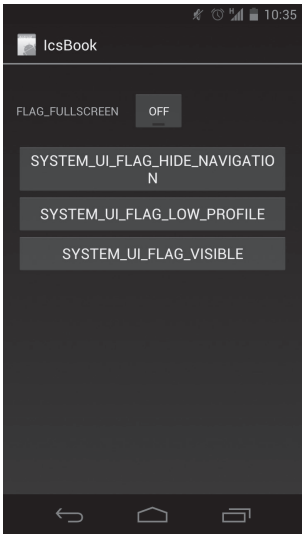


図 1-3　システム UI が常に表示されている状態

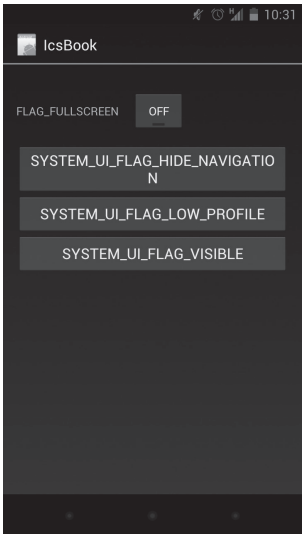


図 1-4　システム UI の見た目が控えめになっている状態

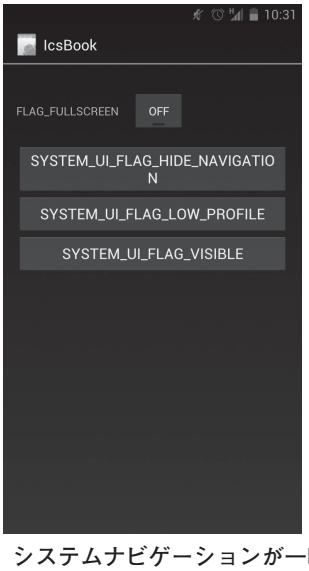


図 1-5　システムナビゲーションが一時的に非表示になっている状態

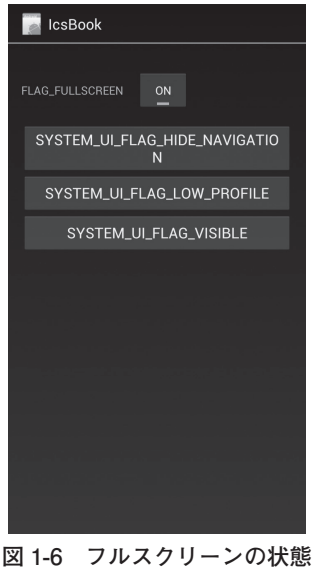


図 1-6　フルスクリーンの状態

コラム

ナビゲーションバーによる画面サイズのコンフィグレーションを考慮する

Android 3.0 から使われているシステムバーや、Android 4.0 から採用されたナビゲーションバーは、縦画面のときと横画面のときで占める領域サイズが変わることがあります。そのため、画面回転時にコンフィグレーションの変化が発生します。

たとえば、縦画面と横画面に同じレイアウトで対応する場合、AndroidManifest.xml の <activity> タグの android:configChanges 属性を使って、画面回転時のアクティビティの再生成を行わないようにすることがあります。

```
<activity android:label="@string/app_name"
          android:name=".section1.example1.MainActivity"
          android:configChanges="orientation" />
```

アプリケーションのターゲット API が API レベル 13 以上の場合に、画面回転時にアクティビティを再生成させたくないときは、"orientation" に加えて "screenSize" も指定しましょう。ターゲット API が API レベル 13 よりも低い場合、screenSize コンフィグレーションは自動で処理され、アクティビティの再生成は行われません。

```
<activity android:label="@string/app_name"
          android:name=".section1.example1.MainActivity"
          android:configChanges="orientation|screenSize" />
```

また、外部出力装置を接続した場合など、画面の短辺の長さが変わる場合は、"smallestScreenSize" コンフィグレーションも変化します。ただし、アプリケーションのターゲット API が API レベル 13 よりも低い場合、smallestScreenSize コンフィグレーションは自動で処理され、アクティビティの再生成は行われません。

1.1.3　ノーティフィケーション

ステータスバーを引き出したときに表示されるノーティフィケーション（通知）は、これまでの 2.x 系と同じです。しかし、拡張された点がいくつかあります。まず、デフォルトのノーティフィケーションで表示できる項目が増えました。これまではタイトルとメッセージ、アイコンだけでしたが、大きいアイコン（コンタクト画像など）やノーティフィケーション数（メールの

未読数など）も表示できるようになりました。さらに、ノーティフィケーション全体のレイアウトをオリジナルのものにすることもできます。たとえば、音楽プレイヤーアプリで再生や一時停止のボタンが付いたノーティフィケーションを出すことなどができます（図 1-7）。詳しくは「第 4 章　ノーティフィケーション」を参照してください。

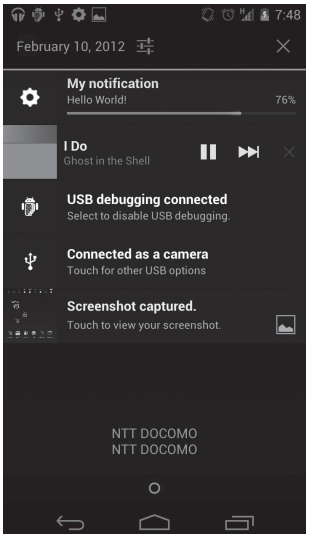


図 1-7　ノーティフィケーション

1.2　アプリケーションの UI

Android 4.0 では、これまでのタイトルバーの代わりにアクションバーが標準で表示されます（図 1-8）。タイトルバーとは異なり、アクションバーはアプリケーションのコントロールを表示する領域として利用されます。これまでメニューキーを押して表示されていた項目は、アクションバー上にオーバーフローアイテムやアクションアイテムとして表示されます。また、タブやドロップダウンリストをアクションバー上に表示して画面を切り替えるなどのコントロールとしても利用できます。詳しくは「第 3 章　アクションバー」を参照してください。

1.2.1　48dp 単位のレイアウト

ナビゲーションバーのサイズは 48dp になっています。また、デフォルトではアクションバーの高さも 48dp に指定されています。このようにシステムが用意しているタッチ可能な UI コン

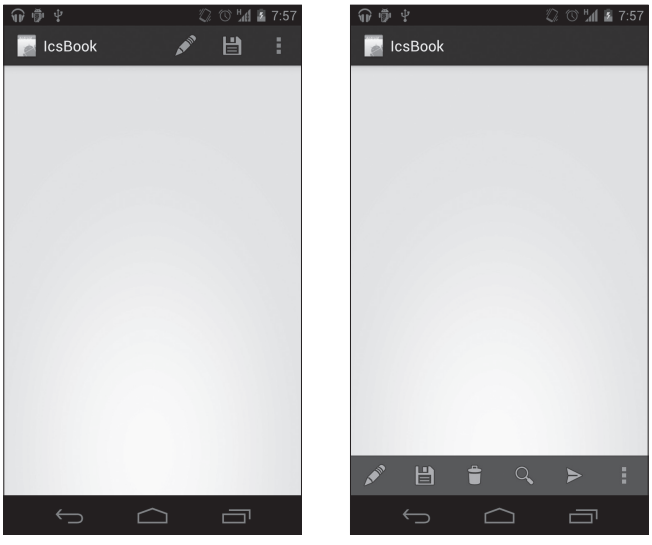


図 1-8　アクションバー

ポーネントの多くは一般的に 48dp を単位として作成されています。

48dp は物理的なサイズに換算すると約 9mm になります。この大きさはタッチスクリーン上のオブジェクトとして推奨されるターゲットサイズ（7 ～ 10mm）に合い、ユーザーが確実かつ正確に指でタッチできるサイズでもあります。

画面に配置するエレメントを 48dp 以上にした場合、次のことが保証できます。

- どのデバイスの画面に表示されているかにかかわらず、推奨される最小のターゲットサイズ（7mm）よりも大きくなる
- UI エレメントのターゲットとしての機能と情報密度との間の適切な折り合いをつけられる

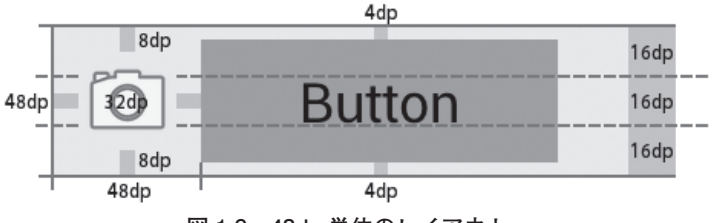


図 1-9　48dp 単位のレイアウト

このとき、エレメント間の間隔を 4dp もしくは 8dp 単位にすると、48dp の約数であるため、レイアウト上のリズムが整います。

たとえば、Android Design サイトでは、図 1-10 のように 4、8、16、32、48dp を単位として利用することが推奨されています [※1]。

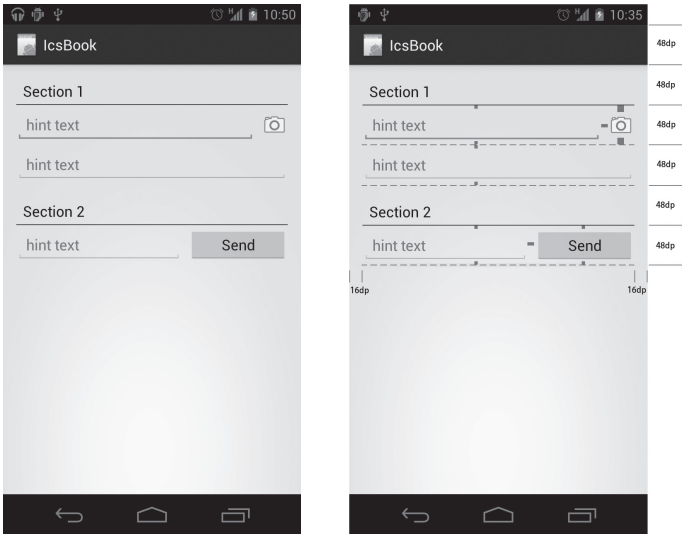


図 1-10　48dp のグリッドに合わせたレイアウト例

1.2.2　タイポグラフィ

Android 4.0 から、デフォルトのフォントが Roboto というタイプファミリーになりました。これまでは Droid Sans というフォントでした。Roboto でも bold、italic、bold italic のスタイルをサポートしています（図 1-11）。

システムデフォルトで用意されているテーマには、それぞれのテーマに合わせてテキストの色と大きさが指定されています。この色と大きさはアプリケーションからも参照できます。

システムで用意されている色には主に表 1-1 と表 1-2 のスタイルがあり、テーマによってそれぞれ定義されている色が異なります。

表 1-1　Theme.Holo.Dark

textColorPrimary	#fff3f3f3
textColorSecondary	#bebebe
textColorPrimaryInverse	#ff000000
textColorSecondaryInverse	#323232

※1　http://developer.android.com/design/style/metrics-grids.html

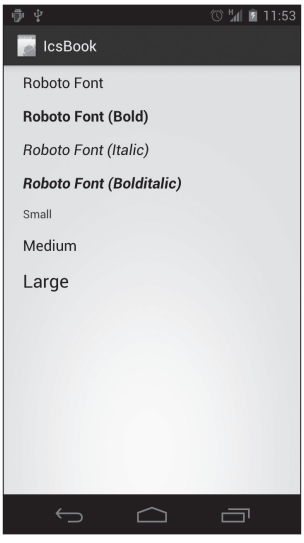


図 1-11　Android 4.0 のデフォルトフォント

表 1-2　Theme.Holo.Light

textColorPrimary	#ff000000
textColorSecondary	#323232
textColorPrimaryInverse	#fff3f3f3
textColorSecondaryInverse	#bebebe

上記のカラーコードは通常状態のテキストカラーであり、タップされた状態、無効状態、選択状態などのカラーコードは別途指定されています。詳しくは Android SDK の中のプラットフォームの `theme.xml` および `style.xml` を参照してください。

これらのスタイルをアプリケーションで利用するには、以下のように属性として参照します。

```
<TextView
  ...
  android:textColor="?android:attr/textColorPrimary"
  />
```

タイトルのテキストサイズと内容のテキストサイズが同じ大きさになっていたり、重要な内容と補足的な内容のテキストサイズが同じになっていたりすると、画面上の情報を捉えにくくなってしまいます。一方、あまりにもたくさんの種類のテキストサイズが画面にあると、一貫性がなくうるさい画面になってしまいます。

そのため、システムでは以下の 3 つのテキストサイズがスタイルとして用意されています。

textAppearanceSmall textAppearanceSmallInverse	14sp 14sp
textAppearanceMedium textAppearanceMediumInverse	18sp 18sp
textAppearanceLarge textAppearanceLargeInverse	22sp 22sp

これらのスタイルは以下のようにアプリケーションから参照できます。

```
<TextView
  ...
  android:textAppearance="?android:attr/textAppearanceLarge"
  />
```

Android 4.0 から、ユーザーが設定アプリでシステム全体のテキスト拡大率を指定できるようになりました。sp（scale-independent pixels）単位で指定されたテキストサイズには、この拡大率が適用されます。たとえば、10sp というテキストサイズは、10dp にシステム全体のテキスト拡大率を掛けた大きさになります。

1.2.3 色

Android 4.0 の標準テーマである Holo テーマでは、タップされた状態や選択された状態などのアクセントとして青が使われています。また、複数選択の画面では緑が使われています。タップされた状態の色や長押ししたときの色は表 1-3 のスタイルとして定義されています。

表 1-3 Theme.Holo および Theme.Holo.Light

colorPressedHighlight	@color/holo_blue_light	#ff33b5e5
colorLongPressedHighlight	@color/holo_blue_bright	#ff00ddff
colorFocusedHighlight	@color/holo_blue_dark	#ff0099cc
colorMultiSelectHighlight	@color/holo_green_light	#ff99cc00
colorActivatedHighlight	@color/holo_blue_dark	#ff0099cc

これらのスタイルは次のように参照できます。

```
<TextView
  ...
  android:textColor="?android:attr/colorLongPressedHighLight"
  />
```

または

```
<TextView
  ...
  android:textColor="@android:color/holo_blue_dark
  />
```

逆に、これらのスタイルをオリジナルテーマで指定することで、アプリケーション全体でこの色が使われている部分を変えることもできます。

さらに、この青と同じようなカラースキームとして紫、緑、オレンジ、赤もシステムで定義されています。

@color/holo_blue_light @color/holo_blue_dark @color/holo_blue_bright	#ff33b5e5 #ff0099cc #ff00ddff
@color/holo_purple	#ffaa66cc
@color/holo_green_light @color/holo_green_dark	#ff99cc00 #ff669900
@color/holo_orange_light @color/holo_orange_dark	#ffffbb33 #ffff8800
@color/holo_red_light @color/holo_red_dark	#ffff4444 #ffcc0000

1.2.4　アイコン

ランチャーアイコン

サイズは 48×48dp（Web 版 Android Market 用のアイコンは 512×512px）です。アイコンは 48×48dp いっぱいに作成します。3 次元で、正面少し上から見たデザインが推奨されています（図 1-12）[※2]。

図 1-12　ランチャーアイコン

※2　http://developer.android.com/design/style/iconography.html

アクションバーアイコン

サイズは32×32dpです。アイコンは32×32dpの内側24×24dpいっぱいに作成します（図1-13）。

図1-13　アクションバーアイコン

アクションバーのアイコンは、アプリケーション内でユーザーが行う最も重要なアクションを表す画像ボタンです。そのため、一目でユーザーが理解できるよう単一の機能を表すものにします。

"refresh" や "share" などのデフォルトで用意されているアイコンは、Android Developersサイト[※3]からダウンロードできます。

絵文字、平面的、詳細すぎず、滑らかなカーブもしくはシャープな形が推奨されています。アイコンの絵が細い場合は斜め45°に傾けて24×24dpの領域を占めるようにします。また、細い部分の線や領域は少なくとも2dp以上になるようにします。

色は次のとおりです。

- アクションバーの背景が白系（`Theme.Holo.Light`など）
  有効状態：`#99333333`（`#333333`の灰色で不透明度60%）
  無効状態：`#4c333333`（`#333333`の灰色で不透明度30%）
- アクションバーの背景が黒系（`Theme.Holo`など）
  有効状態：`#ccffffff`（`#ffffff`の白で不透明度80%）
  無効状態：`#4cffffff`（`#ffffff`の白で不透明度30%）

図1-14　アクションバーアイコンの色

※3　http://developer.android.com/design/static/download/action_bar_icons-v4.0.zip

小さいアイコン / 文脈上のアイコン

アプリケーションの本体でアクションを明確化したり、ステータスを表したりするのに小さいアイコンを使うことがあります。たとえば、Gmail アプリでは重要なメールを表す星アイコンが利用されています。

サイズは 16×16dp です。アイコンは 16×16dp の内側 12×12dp いっぱいに作成します（図 1-15）。

図 1-15　小さいアイコン

平面でシンプル、中立的なデザインが推奨されています。細い線よりも塗りつぶされた形のほうが見やすくなります。ユーザーがその目的を簡単に気づいて理解できるように単一の視覚的メタファーを表現するようにします。

色は、非中間色を目的に合わせて控えめに使います。動作が可能なことを表すアイコンの場合は、背景と十分なコントラストを確保したほうがよいでしょう。

ノーティフィケーションアイコン

アプリケーションがノーティフィケーションを発行する場合、新しいノーティフィケーションが来たときにステータスバーに表示されるアイコンを指定できます。

サイズは 24×24dp です。アイコンは 24×24dp の内側 22×22dp いっぱいに作成します（図 1-16）。

図 1-16　ノーティフィケーションアイコン

平面的でシンプルなデザインが推奨されています。ランチャーアイコンと同じメタファーを簡素化したものにするとよいでしょう。

色は白（`#ffffff`）です。システムが自動的に必要に応じて縮小したり暗くしたりするため、アイコンの色は白にします（図 1-17）。

図 1-17　ノーティフィケーションアイコンの例

1.3　新しいスタイルとテーマ

Android 4.0 では、ベーステーマとして Honeycomb と同じく Holo テーマが使われています。アプリのテーマとして指定したり、オリジナルのテーマを作成するときにベースとして継承するために、次の 3 種類が用意されています。

- Holo Dark
- Holo Light
- Holo Light with dark action bar

Holo Dark

アクションバーの背景色が黒系で、コンテンツエリアの背景色が黒系、文字色が白系のテーマです（図 1-18）。次のように指定します。

```
android:theme="@android:style/Theme.Holo"
```

Holo Light

アクションバーの背景色が白系で、コンテンツエリアの背景色が白系、文字色が黒系のテーマです（図 1-19）。次のように指定します。

```
android:theme="@android:style/Theme.Holo.Light"
```

Holo Light with dark action bar

アクションバーの背景色が黒系で、コンテンツエリアの背景色が白系、文字色が黒系のテーマです（図 1-20）。API レベル 14 から追加されています。次のように指定します。

```
android:theme="@android:style/Theme.Holo.Light.DarkActionBar"
```

図 1-18　Holo Dark テーマ

図 1-19　Holo Light テーマ

図 1-20　Holo Light with dark action bar テーマ

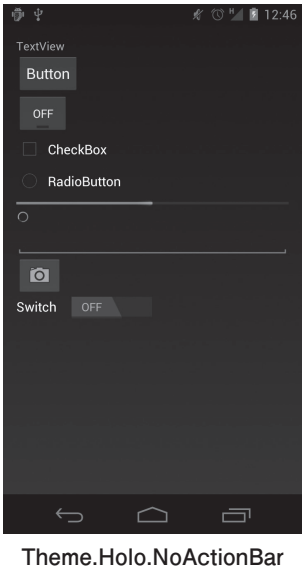
アクティビティのテーマとして利用できるプラットフォームで定義済みのテーマを表 1-4 にまとめました。

表 1-4　定義済みのテーマ

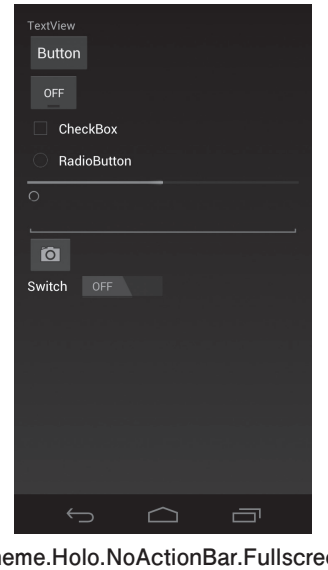
値	説明
黒系のテーマ	
Theme.Holo	黒系のホログラフィックテーマ
Theme.Holo.NoActionBar	アクションバーを表示しないテーマ
Theme.Holo.NoActionBar.Fullscreen	アクションバーとステータスバーを表示しないテーマ
Theme.Holo.Dialog	ダイアログ表示のテーマ
Theme.Holo.Dialog.MinWidth	ダイアログの最小サイズが指定されているテーマ
Theme.Holo.Dialog.NoActionBar	タイトルバーがないダイアログ表示のテーマ
Theme.Holo.Dialog.NoActionBar.MinWidth	タイトルバーがなく、最小サイズが指定されているテーマ
Theme.Holo.DialogWhenLarge	small、normal のデバイスではフルスクリーン、large、xlarge のデバイスではダイアログで表示されるテーマ
Theme.Holo.DialogWhenLarge.NoActionBar	タイトルバーがなく、small、normal のデバイスではフルスクリーン、large、xlarge のデバイスではダイアログで表示されるテーマ

（続き）

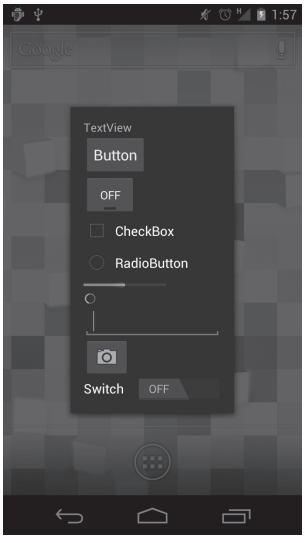
値	説明
Theme.Holo.Panel	すべてのデコレーションがないテーマ
白系のテーマ	
Theme.Holo.Light	白系のホログラフィックテーマ
Theme.Holo.Light.NoActionBar	アクションバーを表示しないテーマ
Theme.Holo.Light.NoActionBar.Fullscreen	アクションバーとステータスバーを表示しないテーマ
Theme.Holo.Light.Dialog	ダイアログ表示のテーマ
Theme.Holo.Light.Dialog.MinWidth	ダイアログの最小サイズが指定されているテーマ
Theme.Holo.Light.Dialog.NoActionBar	タイトルバーがないダイアログ表示のテーマ
Theme.Holo.Light.Dialog.NoActionBar.MinWidth	タイトルバーがなく、最小サイズが指定されているテーマ
Theme.Holo.Light.DialogWhenLarge	small、normal のデバイスではフルスクリーン、large、xlarge のデバイスではダイアログで表示されるテーマ
Theme.Holo.Light.DialogWhenLarge.NoActionBar	タイトルバーがなく、small、normal のデバイスではフルスクリーン、large、xlarge のデバイスではダイアログで表示されるテーマ
Theme.Holo.Light.Panel	すべてのデコレーションがないテーマ
黒系の ActionBar ＋白系	
Theme.Holo.Light.DarkActionBar	アクションバーが黒の白系のホログラフィックテーマ
その他	
Theme.Holo.Wallpaper	背景画像として壁紙を使うテーマ
Theme.Holo.Wallpaper.NoTitleBar	アクションバーがなく、背景画像として壁紙を使うテーマ



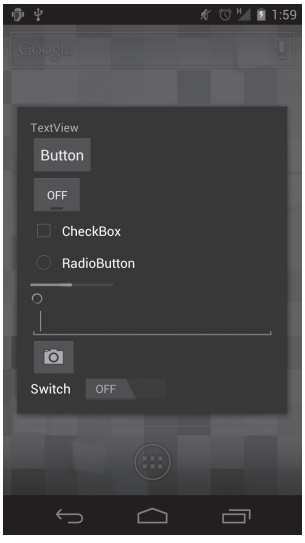
Theme.Holo.NoActionBar



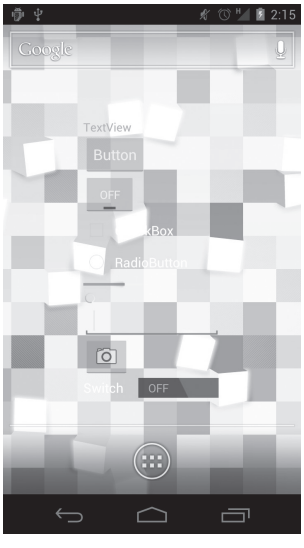
Theme.Holo.NoActionBar.Fullscreen



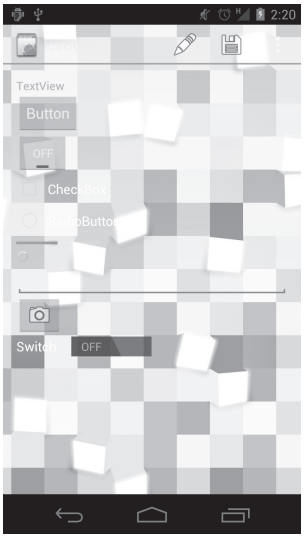
Theme.Holo.Dialog.NoActionBar



Theme.Holo.Dialog.NoActionBar.MinWidth



Theme.Holo.Panel



Theme.Holo.Wallpaper

Theme.Holo.Wallpaper.
NoActionBar

コラム

Theme.Holo.Dialog と Theme.Holo.Dialog.MinWidth

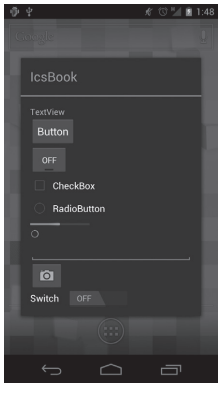
Theme.Holo.Dialog.MinWidth テーマは Theme.Holo.Dialog を継承したテーマです。具体的には次のように定義されています。

```
<!-- Variation of Theme.Holo.Dialog that has a nice minumum width for a regular
 dialog. -->
<style name="Theme.Holo.Dialog.MinWidth">
<item name="android:windowMinWidthMajor">@android:dimen/dialog_min_width_major
</item>
<item name="android:windowMinWidthMinor">@android:dimen/dialog_min_width_minor
</item>
</style>
```

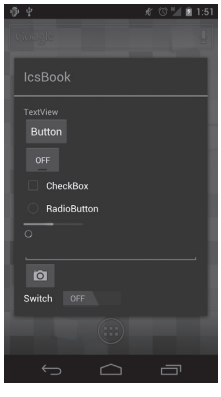
```
<!-- The platform's desired minimum size for a dialog's width when it
     is along the major axis (that is the screen is landscape).  This
     may be either a fraction or a dimension. -->
<item type="dimen" name="dialog_min_width_major">65%</item>

<!-- The platform's desired minimum size for a dialog's width when it
     is along the minor axis (that is the screen is portrait).  This
     may be either a fraction or a dimension. -->
<item type="dimen" name="dialog_min_width_minor">95%</item>
```

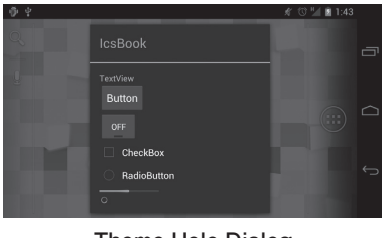
つまり、ダイアログの幅が、横画面では画面の横幅の 65% 以上に、縦画面では画面の縦幅の 95% 以上になるようなテーマです。



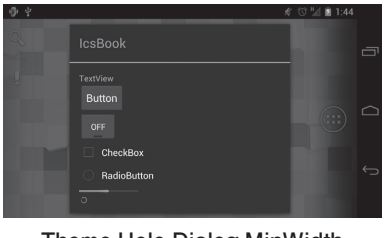
Theme.Holo.Dialog



Theme.Holo.Dialog.MinWidth



Theme.Holo.Dialog



Theme.Holo.Dialog.MinWidth

1.3.1　Device.Default テーマ

Android 4.0 では、AndroidManifest.xml で明示的にテーマを指定していない場合、targetSdkVersion によって適用されるテーマが決まります。targetSdkVersion が 11 よりも小さい場合は @android:style/Theme、11 から 13 の場合は @android:style/Theme.Holo、14 以上の場合は @android:style/Theme.DeviceDefault が適用されます（表 1-5）。

表 1-5　デフォルトテーマ

targetSdkVersion	テーマ
＜ 11	@android:style/Theme
11 ～ 13	@android:style/Theme.Holo
>= 14	@android:style/Theme.DeviceDefault

Theme.DeviceDefault は Android 4.0 から用意されたテーマで、デバイスの各メーカーがシステムデフォルトとしてカスタマイズするためのものです。つまり、Theme.DeviceDefault を指定すると、個々のデバイスに特有のシステムテーマ（たとえば、ボタンのデザインや色など）と同じテーマが適用されます。個々のデバイス固有のテーマではなく、どのデバイスでも Holo テーマを適用したい場合は、明示的に Theme.Holo を指定します。

1.3.2　Android 2.x のサポートと Holo テーマ

Android 2.x をサポートしているアプリケーションでは、デバイスの API レベルに応じて提供されるテーマを変えることができます。つまり、2.x 系のデバイスでは @android:style/Theme を基にしたテーマを、3.0 以降のデバイスでは @android:style/Theme.Holo を基にしたテーマを適用できます。API レベルに応じてテーマを切り替えるには、API レベルによるリソース識別子を使います。たとえば、2.x 系と 3.0 以降でテーマを切り替えるには、次のように values/themes.xml と values-v11/themes.xml に同じ名前のテーマを定義します。

リスト 1-3　**res/values/themes.xml**

```
<resources>

  <style name="MyTheme" parent="@android:style/Theme">
    <!-- 2.x 系のデバイス用のテーマ -->
  </style>
```

```
</resources>
```


リスト 1-4 res/values-v11/themes.xml

```
<resources>

  <style name="MyTheme" parent="@android:style/Theme.Holo">
    <!-- 3.0 以降のデバイス用のテーマ -->
  </style>

</resources>
```

1.4 新しいウィジェット

ここでは主に Android 3.0 以降に追加されたウィジェットを紹介します。

1.4.1 GridLayout と Space

GridLayout は Android 4.0 から新しく追加された ViewGroup です。グリッドをベースにしたレイアウトを簡単に作成できるようにすることを目的に用意されました。また、これまで TableLayout と TableRow で実現していたレイアウトや、LinearLayout を複数入れ子にしていたレイアウトなどを GridLayout に置き換えることで、ビュー階層を少なくすることができます。つまり、GridLayout には以下のような利点があります。

- 縦軸、横軸に揃えるコントロールを両軸同時に行える
- ビュー階層を浅くすることでパフォーマンスを改善する
- グリッドを基準とするデザインツールと相性がよい

たとえば、図 1-21 のようなレイアウトで、ラベルの文字数が変わっても縦横両方のビューに対してベースラインを揃えたい場合、LinearLayout の入れ子では実現できません。この場合は、RelativeLayout もしくは TableLayout を使って実現することになります。

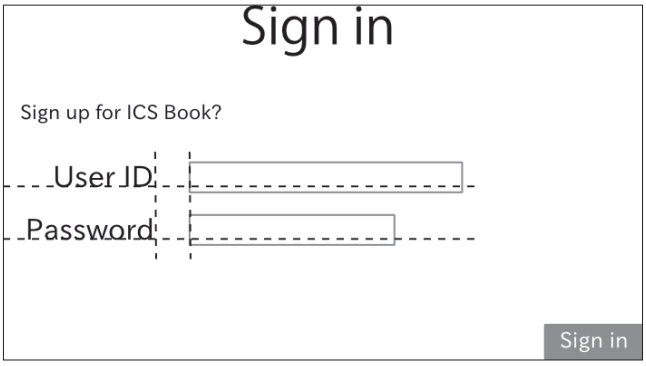


図 1-21　レイアウトの例

では、実際に LinearLayout と RelativeLayout を組み合わせてこのレイアウトを作成してみましょう。

リスト 1-5　**res/layout/main1_4_1_1.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical">

  <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:text="Sign in"
            android:textSize="26dip" />

  <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginBottom="16dip"
            android:layout_marginLeft="8dip"
            android:layout_marginTop="16dip"
            android:text="Sign up for ICS Book?" />

  <RelativeLayout android:layout_width="fill_parent"
                  android:layout_height="wrap_content">

    <TextView android:id="@+id/userIdLabel"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:layout_alignRight="@+id/passwordLabel"
              android:padding="8dip"
              android:text="User ID" />

    <TextView android:id="@id/passwordLabel"
```

```
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_below="@id/userIdLabel"
                android:padding="8dip"
                android:text="Password" />

    <EditText android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignBottom="@id/userIdLabel"
                android:layout_toRightOf="@id/userIdLabel"
                android:ems="7" />

    <EditText android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignBottom="@id/passwordLabel"
                android:layout_toRightOf="@id/passwordLabel"
                android:ems="5" />
  </RelativeLayout>

  <Button android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_gravity="right"
          android:layout_marginTop="16dip"
          android:text="Sign in" />

</LinearLayout>
```

結果は図 1-22 のようになります。

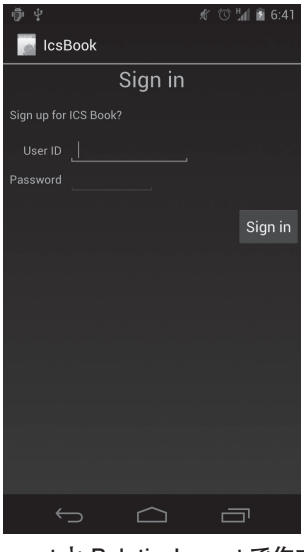


図 1-22　LinearLayout と RelativeLayout で作成したレイアウト

この方法では解決できない問題があります。"User ID" という文字列を表示する TextView の右端を、"Password" という文字列を表示する TextView の右端に揃えるように指定しているため、たとえば "User ID" を "User Account" に変えても、TextView の幅が伸びるわけではないので、2 行になってしまいます。この問題は、GridLayout では解決できます。

では、GridLayout で同じようなレイアウトを作成してみましょう。GridLayout では、描画する領域を行（row）、列（column）およびセル（cell）に分けます。そして、子ビューそれぞれをどのセルに配置するか指定できます。また、列もしくは行に沿って複数のセルにまたがって配置することもできます（図 1-23）。

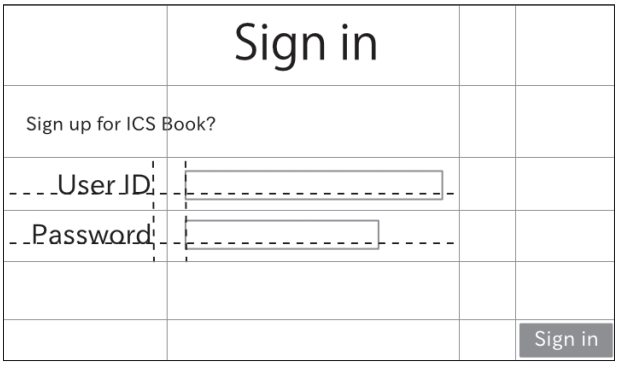


図 1-23　レイアウトの例

このレイアウトを GridLayout を使って作成するコードはリスト 1-6 のようになります。

リスト 1-6　res/layout/main1_4_1_2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:alignmentMode="alignBounds"
            android:columnCount="4"
            android:columnOrderPreserved="false"
             android:padding="8dip"
           android:useDefaultMargins="true">

  <TextView android:layout_columnSpan="4"
            android:layout_gravity="center_horizontal"
            android:text="Sign up"
            android:textSize="26dip" />

  <TextView android:layout_columnSpan="4"
```