S!アプリ開発ガイド

[Tips 編]

Version 1.0.2 / Apr.23,2010

ソフトバンク モバイル 株式会社

本書は情報提供を目的として作成されたものです。ソフトバンクモバイル株式会社は本書の記載内容に関して明示的にも、黙示的にも何ら保証するものではありません。

本書に記載されている事柄は、予告なしに変更する可能性があります。

本書の使用、または本書を使用した結果については、ユーザ各位がその責任を負うものとしますのでご了承ください。

1. ドキュメントの一部または全部を改版、引用することを禁じます。
2. ドキュメントを第三者に複製し、頒布することを禁じます。
3. ドキュメントを運用した結果の影響については、いっさいの責任を負いかねますのでご了承ください。

［商標］

- Powered by JBlend™, (C)1997-2010 Aplix Corporation. All rights reserved.
- JBlend および JBlend に関する商標は、日本およびその他の国における株式会社アプリックスの商標または登録商標です。
- S!アプリ対応のソフトバンク携帯電話は、株式会社アプリックスが開発し、Java™アプリケーションの実行速度が速くなるように設定された JBlend ®を搭載しています。
- S!アプリは、Java™に対応したアプリケーションです。
- Java および Java に関する商標は、米国およびその他の国における米国 Sun Microsystems, Inc.の商標または登録商標です。
- SMAF はヤマハ株式会社の商標または登録商標です。
- SOFTBANK およびソフトバンクの名称、ロゴは日本国およびその他の国におけるソフトバンク株式会社の商標または登録商標です。
- S!アプリはソフトバンクモバイル株式会社の商標または登録商標です。

その他、記載されている会社名、製品名は、各社の商標または登録商標です。

■修正履歴

Version	日付	内容
1.0.0	2006/10/1	新規文書
1.0.1	2008/8/1	1.1.1 startApp() 制限を追記 1.2 RecordStore 文言を修正 1.3 通信 MIDP1.0、MIDP2.0について追記 1.4 Thread 文言を修正 3. VM 共通の制限、注意について追記 4.8. 端末上の制限 新規追加 4.9. タイマー起動アプリ/待受アプリ 新規追加 5.4. PNG 新規追加
1.0.2	2010/4/23	MIDP 1.0対応端末に関する記載を削除 以下、既存の他ドキュメントの記載を移行し、新規追加 ・ 1.9. SpriteCanvas ・ 1.19. RecordStore ・ 1.20. Image 以下、既存の他ドキュメントの記載を移行し、制限または注意を追記 ・ 1.3. 通信 ・ 1.7. Graphics ・ 1.8. Canvas ・ 1.10. Display ・ 2.6. LocalizedTextBox,LocalizedTextField,TextBox, TextField 以下、項目削除 ・ 1.18. Timer ・ 2.5. ResorceOperator ・ 2.6. ResorceOperatorManager 誤記載、修正

0. イントロダクション

0.1. 目的

本書はS!アプリを提供するにあたり、必要な技術情報を提供するものである。

0.2. 前提

本書は以下の技術について熟知していることを前提とする。

- Java™2
- CLDC ： Connected Limited Device Configuration
- MIDP ： Mobile Information Device Profile

加えて、弊社提供の各種開発ガイドを既読であること。

0.3. 参考文献

“Configurations and Profiles Architecture Specification, Java™2 Platform Micro Edition (J2ME)”, Sun Microsystems, Inc.
“Java™2 Platform: Micro Edition, A White Paper”, Sun Microsystems, Inc.
“The K Virtual Machine (KVM), A White Paper”, Sun Microsystems, Inc.
“Connected, Limited Device Configuration, Java™2 Platform Micro Edition”, Sun Microsystems, Inc.
“Mobile Information Device Profile (JSR-37), Java™2 Platform Micro Edition”, Sun Microsystems, Inc.

0.4. 本書の構成

本書は以下の構成である。

1章　CLDC/MIDP：CLDC/MIDPの範疇におけるtipsを説明する。
2章　MEXA/JSCL：MEXA/JSCLの範疇におけるtipsを説明する。
3章　VM：現VMを利用する上での留意点を説明する。
4章　その他：その他、端末上でのS!アプリを実現する上でのtipsを説明する。
5章　データ：S!アプリで使用するデータについての注意点を説明する。

1. CLDC/MIDP

CLDC/MIDP での規定事項を含め、弊社端末で S!アプリを作成する際の tips を記載する。

1.1. MIDlet

MIDlet のサブクラスを実装する上での留意点を記載する。

1.1.1. startApp()

注意：

- MIDlet#startApp ()の処理完了前に一時停止すると、S!アプリが異常終了する可能性が高い。
- MIDlet#startApp ()の実装は簡素かつ短時間の処理が望ましい。

制限：

- MIDlet#startApp ()内では以下の処理は避けること。

作業メモリの確保
Canvas インスタンスの生成
MediaPlayer インスタンスの生成
ネットワーク入出力(別スレッドで行うなら可)

- 端末上では同時に複数個の S!アプリを動作させることはできない。端末上では、常に単一の S!アプリをロードし、かつ、ロードした単一 S!アプリのみを実行することができる。

1.1.2. pauseApp()

制限：

- MIDlet#pauseApp()は起動後１秒経過すると強制終了する。時間のかかる処理は実装しないこと。

1.1.3. destroyApp()

制限：

- 正常終了時にMIDlet#destroyApp()がコールされる。コール後、5秒で強制終了される。

1.2. RecordStore

制限：

- 端末に RecordStore の内容を保存中、バッテリーを外す、終話ボタンを押す等、保存動作を遮断した場合にアプリは強制終了する場合がある。また、データについては保証できない。

注意：

- RecordStore の不揮発性メモリへの書込みタイミングは、従来の終了時ではなく随時書込みとなる。
  このため、ApplicationManager クラスの flushRMS()メソッドを使用することができない場合がある。
  ただし、RecordStore へ書込み中に S!アプリが終了する(エラー終了も含む)、複数スレッドから同時に書込みを行うなどの場合はデータを保証することができない。

1.3. 通信

制限：

- HttpConnection をはじめとする通信処理は、主スレッドとは別のスレッドで処理すること。主スレッドとは別のスレッドで通信しないと、S!アプリは通信中に一時停止することができない。通信中は他の処理をブロックする為、外部から一時停止を要求されても MIDlet#pauseApp()を実行できず、1 秒以内に MIDlet#pauseApp()処理が完了しないと強制終了する。(1.1.2. pauseApp()を参照のこと)
- open したものについては、HTTP 通信が終了したら、速やかに close する必要がある。
- HTTP 通信でデータを取得する際に、1 度の InputStream.read()で全てのデータを読み込めるとは限らない。そのため、実際に読み込めたサイズの返り値が-1（=読み込み終了）になるまで読み込み操作を繰り返す必要がある。
- HTTP 通信でデータを取得する際に、ContentConnection.getLength()では HTTP ヘッダの Content-Length がない場合は、返り値が-1 になる。この場合は、コンテンツの長さが不明ということなので、InputStream.read()の返り値が-1（=読み込み終了）になるまでデータの読み込み操作を行う必要がある。
- S!アプリよりヘッダを設定することは可能であるが、設定したヘッダの動作については保証できない。

- インターネット上の Web サーバに GET,POST リクエストにより送付する URL のサイズは 1024bytes 以下でなければならない。また、GET,POST のリプライは 12kbytes（Header 情報含）以下、でなければならない。但し、JAR によりサイズを圧縮することができるデータであれば、300kbytes（Header 情報含）まで、可能である。

注意：

- 通信を行っている最中は CPU をはじめとするハードウェア資源を通信処理に優先的に割り当てる為、通信を行っているスレッドとは別のスレッドで平行して実行中の処理(描画等)が正常に行われない可能性がある。

1.4. Thread

制限：

- MIDP では通常の Java™2 に含まれる、Thread#stop(), Thread#suspend(), Thread#interrupt()がない。即ち、あるスレッドから別のスレッドに対して、終了、停止、割込みをかけることはできない。
- スレッドを停止するには自発的に停止しなければならない。
- スレッドのスケジューリングに対して、タイミングに依存したプログラミングをしている S!アプリついては、動作を保証しない。従って、何がどのスレッドで動いているかを常に把握しておく必要がある。
- Canvas#serviceRepaints()を頻繁に呼ぶスレッドの場合は、S!アプリ一時停止時に停止または一度停止するような作りにすること。
- スレッドの同期をとる場合には、synchronized を用いたプログラミングを行うこと。弊社で提供する端末では、機種ごとに VM のスレッドの切り替えタイミングが異なるため、synchronized 以外の手段でスレッドの同期をとろうとした場合は、期待通りの動作をしないことがある。

1.5. イベントリスナー

制限：

- イベントリスナーの中で別のイベントの発生を待つような実装をしてはいけない。そのような実装を行うと当該リスナーがデッドロックする。

注意：

- イベントリスナーでは重い処理を実装しないこと。処理が重いと他のイベント割込みの処理を取りこぼすことがある。

1.6. 計時(Timer と Thread)

注意：

- TimerTask を Timer でスケジュールし、TimerTask の起動までの間に S!アプリが一時停止状態となった場合には、Timer はその時間を含めてスケジュールされた時間を計時する。即ち、TimerTask にとって、S!アプリの実行状態の如何に関わらず時計は常に動いているものとみなす。そのため、複数個の TimerTask をスケジュールしておくと、一時停止状態から再開した際に複数個の TimerTask が順次 run することがありうる。

- `Thread#sleep()`の実行中に S!アプリが一時停止状態にある場合にはその時間は `Thread#sleep()`には加算しない。即ち、`Thread#sleep()`にとって、一時停止状態では時計が止まっているものとみなす。

1.7. Graphics

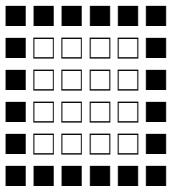
注意：

- Graphics#draw～()とGraphics#fill～()では描画する領域のサイズが異なる。以下のメソッドでそれぞれ描画域が異なっている。

draw～()	fill～()
drawRect	fillRect
drawRoundRect	fillRoudRect
drawArc	fillArc

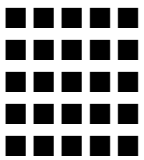
⇒ Graphics#draw～()では与えた幅と高さのそれぞれに１ピクセル分を加えた領域の枠線を描画する。

例：drawRect (0, 0, 5, 5)



⇒ Graphics#fill～()では与えた幅と高さの領域を塗りつぶす。

例：fillRect (0, 0, 5, 5)



- RoundRectangle の引数値が小さい場合には角が丸くならずに、見た目がRectangle と等価となる。引数が幾つ未満から見た目が Rectangle と等価になるかは、端末メーカーの実装に依存する。

- Anchor Point 指定する際、縦方向だけ、または横方向だけの指定ではIllegalArgumentException が発生する。描画するには、縦方向・横方向の両方を指定する必要がある。

```
g.drawChar ('A', 0, 0, Graphics.LEFT|Graphics.TOP);
```

また、Graphics.VCENTER を指定可能なメソッドはイメージ描画系のメソッド（drawImage()、copyArea()、drawRegion()）のみであり、文字列描画系メソッド（drawString()、drawSubstring()、drawChar()、drawChars()）に指定した場合には、IllegalArgumentException が発生する。

1.8. Canvas

注意：

- Canvas#isDoubleBuffered()の値により実装を作り分けると、描画性能が向上することがある。下記を試みてみるとよい。
  - ✧ 真値ならばオフスクリーンを利用せずに直書きする。
  - ✧ 偽値ならばオフスクリーンを用意して描画する。

- repaint();
  // (A)

  (A)の部分のコードが実行される時点で repaint()を契機とした画面更新が終わっていることは保証できない。 (A)の部分に画面更新が終わったことを前提とするコードが必要な場合は、repaint() と(A)のコードの間にserviceRepaints()を呼び出して repaint()による画面更新の実行を待ち合わせる必要がある。

- Canvas クラスを継承した S!アプリで paint()メソッドを実行中、セキュリティダイアログ表示対象 API (HTTP/HTTPs 通信など) を実行した場合、セキュリティダイアログの表示前に実行した描画メソッドの結果を破棄する。

  例)

```
protected void paint( Graphics g ) {
    // ここから
    g.setclip( 0, 0, 100, 100 );
    g.setColor( 0x0ffffff );
    g.fillRect( 0, 0, getWidth(), getHeight() );
    // ここまでの描画結果は無効になる
    try{
            HttpConnection conn =
            (HttpConnection)Connector.open(url, Connector.READ,true);
            // セキュリティダイアログ表示
            conn.setRequestMethod(HttpConnection.GET);
            InputStream in = conn.openInputStream();
            :
```

```
            } finally {
                    conn.close();
            }
        }
        catch(SecurityException e){
        // ユーザが HTTP 通信を許可しなかった場合の処理を記述
        }
        // 以降の描画結果は有効
        g.drawLine(0,0,20,30);
        g.drawString("Hello world!",20,20,Graphics.TOP|Graphics.LEFT);
        :
    }
```

セキュリティダイアログ表示対象 API (SecurityException を送出する可能性のあるメソッド) を Canvas 継承クラスの paint()メソッド内で使用する場合、S!アプリでの描画は、セキュリティダイアログ表示対象 API の後に実行するように記述すること。また、その際には try～catch により、SecurityException を適切に処理すること。

- GameAction に該当するキーの正しい判定をするためには、S!アプリで Canvas#getGameAction(keycode) および getKeyCode(gameAction) により GameAction とキーコードの対応を取得する必要がある。

  例１)

```
protected void keyPressed(int keyCode) {
    int gameAction = getGameAction (keyCode);
    if(gameAction == UP) {
            System.out.println ("keyPressed("+ keyCode +"): UP");
    }
    else if(gameAction == FIRE) {
            System.out.println ("keyPressed("+ keyCode +"): FIRE");
    }
}
```

  下記方法は間違っているわけではないが、複数のキーに同じ GameAction が割り当てられているプラットフォームの場合、意図しない振る舞いになる可能性がある。

  例２)

```
protected void keyPressed(int keyCode) {
    if(keyCode == getKeyCode(UP)) {
            System.out.println ("keyPressed("+ keyCode +"): UP");
    }
    else if(keyCode == getKeyCode(FIRE)) {
            System.out.println ("keyPressed("+ keyCode +"): FIRE");
    }
}
```

1.9. SpriteCanvas

注意：

- Canvas クラスの paint()メソッドに渡す Graphics オブジェクトについて、paint()を呼び出した時点でのクリップ領域に何らかの仮定を設けて明示的にクリップ領域を設定していない場合、意図したとおりの画面描画を行なわない場合がある。paint()メソッドの先頭など、必要なタイミングで Graphics#setClip()を実行する必要がある。

  例)

```
public void paint(Graphics g) {
        g.setClip(0, 0, getVirtualWidth(), getVirtualHeighit() );
        :
```

  特に、SpriteCanvas を使用し、実画面サイズを越える（仮想画面）領域への描画を行なう場合、描画メソッドを実行する前に適切にクリップ領域を設定する必要がある。

1.10. Display

注意：

- Display#setCurrent() と Displayable#addCommand () または Displayable#removeCommand()を非同期に実行しないこと。非同期に実行した場合、ソフトキーの配置が乱れることがある。

- Display.setCurrent(Displayable) 後に Displayable.isShown() が true になるタイミングは、setCurrent()後の初回の paint()実行直後となる。

- MIDP 2.0 対応端末では Display#setCurrent()は非同期メソッドとして実装しているため、以下の点に注意すること。
  例１）

```
// (A)
setCurrent(d);
// (B)
```

  (A) カレント切り替え前を前提としたコード
  (B) カレント切り替え前を前提としたコード

  MIDP 2.0 対応端末の実装(非同期実装)では、(B)の部分のコードが実行される時点で setCurrent()を契機としたカレント切り替えが終わっていることは保証しない。つまり、(B)の部分のコードは、カレントがまだ切り替わってない事を前提としたコードでなければならない。カレント切り替え後に実行する必要のあるコードは、showNotify()が呼び出された以降、または isShown() が true を返す状態になってから実行されるように記述する必要がある。

  例２）

```
setCurrent(d);
serviceRepaints();      //(C)
```

  (C)serviceRepaints()でのカレント切り替えの待ち合わせはできない。

  (C)が実行される時点でカレント切り替え後、repaint()呼び出し済みであることは保証されない。よって、(C)の serviceRepaints()で画面更新まで処理がブロックすることは期待できない。 このように setCurrent()によるカレント切り替えを待ち合わせるには､以下のような待ち合わせコードが必要です。

```
setCurrent( d );
while( !d.isShown() ){
    Thread.sleep(100);
}
```

1.11. Alert

注意：

- Alert をモーダルで利用する場合には、AlertType を FOREVER に設定することに加え、Display の切り替えに留意すること。
- Alert をモーダル(modal dialog)として利用する際には画面の切り替えは、Display#setCurrent(Displayable nextDisplayable) ではなく **Display#setCurrent (Alert Alert, Displayable nextDisplayable)**を利用する。

1.12. プラットフォーム名称

注意：

- System.getProperty("Platform")で返す値が端末のプラットフォーム名称である。(端末の正式名称ではない。)

  ◎ 例

  System.getProperty ("Platform") ⇒ "944SH"

- 尚、弊社提供の開発支援ツールではツール独自の名称を返す。

  ◎ 例

  System.getProperty ("Platform") ⇒ "microJBlend emulator"

1.13. ライブラリバージョンの取得

注意：

- System.getProperty("microedition.profiles")で返す値がライブラリバージョンである。

◎　例

System.getProperty ("microedition.profiles")

⇒ "MIDP-2.0"

1.14. MIDlet#getAppProperty()

注意：

- MIDlet#getAppProperty()を使用することで、JAD,Manifest から S!アプリ固有の情報を取得可能である。JAD,Manifest の展開、情報の検索に使用されるエリアはヒープ上から確保されるが、処理終了時に開放される。

1.15. 標準出力、標準エラー出力

制限：

- 標準出力、標準エラー出力へ印字する場合の振る舞いは、端末の実装に依存する。商用サービスではより多くの端末での稼動を確実にする為、標準出力、標準エラー出力への印字は行わないこと。
- 特に、下記のメソッドに注意されたい。

メソッド	例
java.io.PrintStream#print()	System.out.print() System.err.print()
java.io.PrintStream#println()	System.out.print() System.err.print()
java.lang.Throwable#printStackTrace()	

1.16. コマンドリスナー

注意：

- コマンドリスナー内では Display#setCurrent()を 2 回以上呼ぶのは望ましくない。どうしても 2 回以上呼びたい場合は、コマンド処理を別スレッドにすると 1 回目の Display#setCurrent()が無視されることはなくなる。

- コマンドリスナー内でDisplay#setCurrent ()を2回呼ぶと以下のようになる。
  - ✧ 1 回目の Display#setCurrent ()は無視されて、画面がクリアされる。
  - ✧ 2 回目の Display#setCurrent ()が実行されると画面が切り替わる。

1.17. フォント

注意：

描画関数内でフォントサイズの切り替えを行う場合は、以下に注意する。

- フォントサイズが以下の端末の場合の例

・デフォルト及び Font.SIZE_MEDIUM・・・12 ドット
・Font.SIZE_LARGE ・・・16 ドット

Canvas クラスにて以下のコードを実行すると、

```
public void paint ( Graphics g ){
        //文字列を描画
        g.drawString ( "ABC", 0, 0, g.TOP|g.LEFT );      ・・・①
        //フォントサイズ変更  MEDIUM → LARGE
        g.setFont ( Font.getFont( Font.FACE_SYSTEM,
        Font.STYLE_PLAIN, Font.SIZE_LARGE ) );   ・・・②
        //文字列を描画
        g.drawString ( "DEF", 0, 20, g.TOP|g.LEFT );     ・・・③
    }
```

✧ 1 度目の関数呼び出しでは、①はデフォルトの 12 ドットで描画される。③は 16 ドットで描画される。
✧ 2 度目以降の関数呼び出しでは、①は 16 ドットで描画される。

①を 12 ドット、③を 16 ドットで描画したい場合は、①の前に以下のコードを挿入すること。

```
//デフォルトフォントを設定
g.setFont( Font.getDefaultFont() );
```

- 文字フォントの縦幅、横幅は端末により異なる。S!アプリにて Canvas などに文字を描画する場合にはこれに留意する必要がある。実画面サイズと文字フォントのサイズを取得するメソッドを利用し、文字を動的に描画すること。

尚、各端末のフォントサイズについては S!アプリ開発ガイド「端末情報（各種)」および、コンテンツ開発ガイド[端末情報（各種)]を参照のこと。

1.18. List

制限：

- 現象

  Display#setCurrent()前に Screen#setTicker()または Screen#setTitle()を実行すると、スクロールした時にカーソルが画面から消えてしまう。

  解決策

  ① setTicker ()またはScreen#setTitle ()を setCurrent ()後に行うこと。

  ② setTicker ()または Screen#setTitle ()後に List に対して要素を List#append()してから setCurrent ()を行うこと。

1.19. RecordStore

注意：

- RecordStore への書き込み内容をヒープ領域にて保持せず、随時端末内部メモリへの書き込み処理を開始する。ただし、端末内部メモリへの書き込み完了タイミングについては端末実装依存となるため注意すること。

1.20. Image

注意：

- Image#createImage()を利用する際、絶対パスにて指定する標準の方法となる。

  Image#createImage(/image);

2. MEXA/JSCL

2.1. MediaPlayer

制限：

- MediaPlayer のインスタンスで画像を表示中に、同時に Canvas のインスタンスを使用することはできない。
- MediaPlayer のインスタンスと PhrasePlayer のインスタンスを単一の S!アプリの中に共存させることはできない。
- FEP での文字入力中に MediaPlayer のコントロールを行なわないこと。
- MediaPlayer のインスタンスで表示系 SMAF、JPEG、PNG を再生中には、Display#setCurrent()は実行しないこと。

注意：

- MediaPlayer のインスタンスの実行中に Display#setCurrent()により他の Displayable を与えると、MediaPlayer の実行は一時停止する。一時停止した MediaPlayer を Display#setCurrnet()に与えると MediaPlayer の実行を再開する。Display#setCurrent()で MediaPlayer の実行状態を遷移させた場合には、MediaPlayerListener が呼ばれることはない為、MediaPlayerListener の中で PAUSED,PLAYED,STOPPED のいずれの状態も取得することができない。
- 他の Displayable を Display#setCurrent()して MediaPlayer のインスタンスを一時停止させている際にMediaPlayer#play()を実行すると例外が発生する。
- MediaPlayer のインスタンスで音曲を再生している際には、Canvas 等の Displayable 派生クラスのインスタンスを同時に利用することが可能である。
- MediaPlayer のインスタンスを制御するには、必ず、MediaPlayer が PAUSED, PLAYED, STOPPED のいずれの状態にあるかを確認した上で､行うこと｡例えば、MediaPlayer#play()を実行し、MediaPlayerListener が PLAYED を取得する前に MediaPlayer#pause()を実行しても一時停止しない。
- メディアデータが静止画の場合、MediaPlayer#play ()の直後に再生完了のイベント(MediaPlayerListener.STOPPED)があがり、その後、静止画の再生に対して MediaPlayer#pause (),MediaPlayer#stop()を行っても無視する。
- MediaPlayer 鳴動開始と鳴動停止の処理を確実に行う為に、MediaPlayerListner#mediaStateChanged イベントハンドラにて MediaPlayer の状態を管理する必要がある。
  特に下記の挙動に注意を払うこと。

① 再生状態前(MediaPlayerListner#mediaStateChanged の played イベントを受ける前)では、MediaPlayer#stop ()メソッドの呼出を無視する。

② 再生状態(MediaPlayerListner#mediaStateChanged の played イベントを受けた後)では、別の MediaPlayer#play()メソッドの呼出が発生した場合に RuntimeException が発生する。

- ②のような例外が発生してから再び再生するには MediaPlayer#stop ()メソッドを呼び出してから MediaPlayer#play ()メソッドを呼び出す必要がある。
- MediaPlayer サウンドの停止から次の開始には、少なくとも 400msec 程度の間隔をあけること。但し、サウンドデータの大きさや CPU 負荷状態依存する為、実機にて十分に確認すること。

2.2. PhrasePlayer

制限：

- 同期設定するフレーズデータの長さを揃える必要がある。フレーズデータの長さが違った場合、同期設定時に Exception が発生する。

注意：

- PhraseTrack#resume()した場合、pause()した位置から再生する。
- AudioPhraseTrack に Audio データをセットしていない状態で、setVolume() または mute() を行っても無効になる。
- AudioPhraseTrack#play(int repeat)が下記の場合意図した動作をしない。

play 再生中に stop して play を呼んだ場合に限り、stop 後の play の回数が無効となる。但し、play 再生完了後に play を行った場合は問題ない。

✧ play ()→再生中にstop ()→play(int repeat)と実行した場合に繰り返し再生を行わない。
✧ play (int repeat)→再生中にstop ()→play()と実行した場合に繰り返し再生を行う。

上記現象は stop ()→play ()間において、データを再度セットし直す(removePhrase()→setPhrase())と回避可能である。

- 着信時に com.jblend.media.smaf.phrase.PhrasePlayer#kill()メソッドをコールした場合、RuntimeException が発生する。

2.3. DeviceControl

注意：

- DeviceControl#setKeyRepeatState()の振る舞いは端末の実装に依存する。

2.4. FixedPoint

注意：

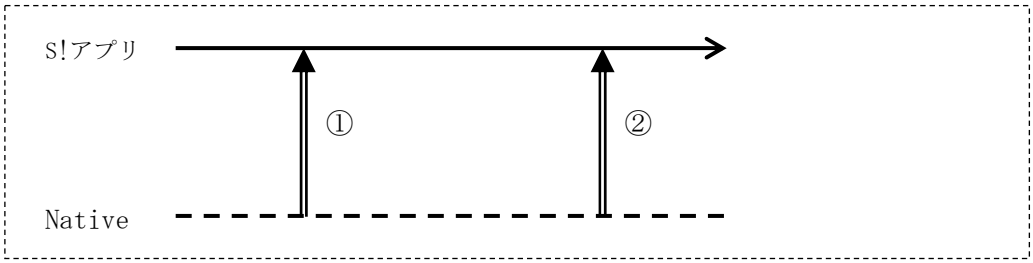
- FixedPoint でオーバーフローを起こしても例外を発生しない為、オーバーフローする可能性がある演算の実行に際しては事前に FixedPoint#isInfinit()によりオーバーフローしないことを確認すること。

2.5. イベントリスナー

注意：

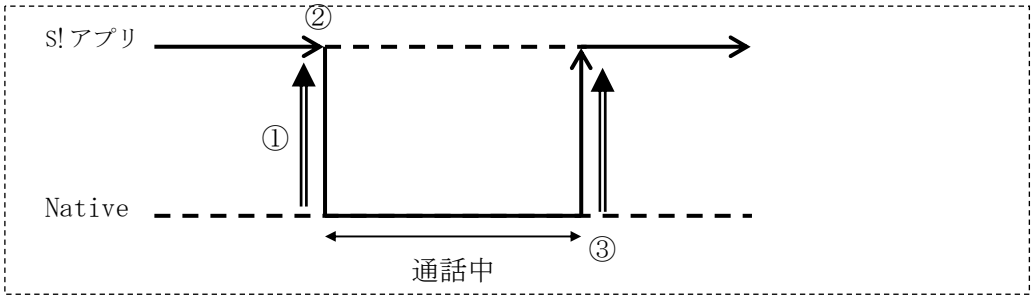
- イベントリスナーに渡す文字列にNULLが入る場合がある。(TelephonyListener, ScheduledAlarmListener)
- TelephonyListener#ignored() （切断イベント）は必ず、TelephonyLis**tener#ring()**(音声着信イベント)と対で発生する。動作例を以下に示す。

< Pattern 1 >



- 音声着呼により、TelephonyListener#ring ()を呼ぶ。
- 応答せずに切断すると、TelephonyListener#ignored ()を呼ぶ。
- (簡易留守録に切り替わった瞬間にもTelephonyListener#ignored ()を呼ぶ)

< Pattern 2 >



- 音声着呼により、TelephonyListener#ring ()を呼ぶ。
- 応答する為Nativeに切り替わる。
- 通話終了してS!アプリを再開すると、MIDlet#startApp()、TelephonyListener#ignored()が並列に呼ばれる。

2.6. LocalizedTextBox,LocalizedTextField, TextBox,TextField

制限：

- 着信イベント発生時、TextBox,TextField の FEP 中に S!アプリが自ら一時停止した場合、S!アプリ再開時に startApp()は呼ばれない。

注意：

- コンストラクタに渡す最大文字数は、0 より大きな値にすること。TextBox,TextField に割り当てることができる最大文字数の上限は、取得するテキストエリア分のメモリをヒープから取得可能な限りとなる。ヒープより大きなメモリ(指定文字数分のテキストエリア)を確保しようとした場合 OutOfMemoryError が発生する。

- TextField、TextBox にて NUMERIC を利用している場合、“-”を入力すると Exception が発生する。

2.7. StorageConnection

制限：

- StorageConnection#openInputStream()にて InputStream から 1 バイトごとにデータを読み込む場合には、InputStream#read() (引数無し) ではなく、InputStream()#read(byte [] b) (b＝1) を利用すること。

2.8. MailData

制限：

- DataElement.setString()にて送信先メールアドレス（To、Cc、Bcc）が最大まで登録している状態で空きフィールド指定（引数 index＝-1）による追加、設定要求（削除要求（引数 value = null）を除く）を行おうとした場合、IoException(“set maximum”)が発生する。

  ＜例＞　to を 5 件設定した状態

  setString(TO_ADDRESS_INFO,0,” to0”);

  setString(TO_ADDRESS_INFO,1,” to1”);

  setString(TO_ADDRESS_INFO,2,” to2”);

  setString(TO_ADDRESS_INFO,3,” to3”);

  setString(TO_ADDRESS_INFO,4,” to4”);

  （問題）

  setString(TO_ADDRESS_INFO,1,” 変更”);

  ⇒　IOException(“set maximum”)発生

  (回避例)

  setString(TO_ADDRESS_IFFO,1,null);

  setString(TO_ADDRESS_INFO,1,” 変更”);

  ⇒　index 1 の内容を変更する場合、index 1 を削除し、その後 index 1 を登録する。

3. VM

制限：

- System.gc()をコールしたタイミングでガベージコレクションが実行されるとは限らない。VM にガベージコレクションの実行を促すのみとなる。実際のガベージコレクションが処理されるタイミングは実装依存である。

注意：

- 頻繁に System.gc()をコールした場合、アプリの動作が遅くなる場合がある。
- ヒープの量が大きくなればなるほどガベージコレクションやメモリコンパクションの処理に時間がかかるようになる。そのため、動作速度を重視するような S!アプリでは、クラスインスタンスをできるだけ使いまわすなどの工夫をして、不要にインスタンスの生成/開放を行わないように注意する必要がある。

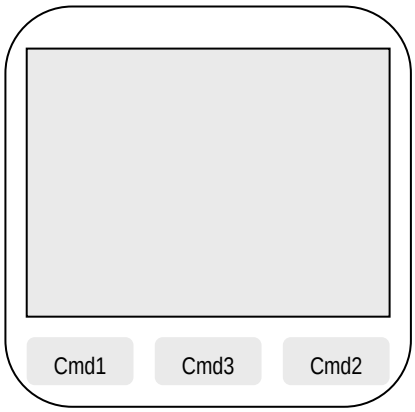
4. その他

4.1. ソフトキー

注意：

- MIDP で規定されるソフトキー(Command)の表示位置は、CommandType 毎に端末の実装に依存する為全ての端末で同じ位置に表示されるとは限らない。表示位置を確定するには Command.SCREEN を使用すること。(推奨) Command.SCREEN の Priority は数字が小さいほうから、左、右、真ん中の順に表示される。SCREEN に関しては全ての端末(ソフトキー表示領域が一つの端末は除く) で表示位置が共通となる。

例：ソフトキー表示領域が 3 つの場合




```
Command Cmd1 = new Command(“Cmd1”, Command.SCREEN, 1);
Command Cmd2 = new Command(“Cmd2”, Command.SCREEN, 2);
Command Cmd3 = new Command(“Cmd3”, Command.SCREEN, 3);
```

- 下記のように空文字列を Command の Label に設定しないこと。

```
Command Cmd2 = new Command("", Command.SCREEN, 1);
Command Cmd2 = new Command(" ", Command.SCREEN, 1);
```

制限：

- コマンドボタンのラベルに記載できる文字数は端末の実装に依存する。
- コマンドボタンのラベルに記載できる文字としては制御文字以外の印字可能文字(弊社固有の絵文字含む)を指定できる。但し、弊社固有の絵文字のうち動画絵文字を指定した場合について、絵文字がアニメーション動作をするか否かは端末の実装に依存する。
  制御系エスケープシーケンス(“¥n”など)は全て無効となる。
- ソフトキーのイベントは DeviceControl#getDeviceState()にて取得しないこと。

4.2. コマンドメニュー

注意：

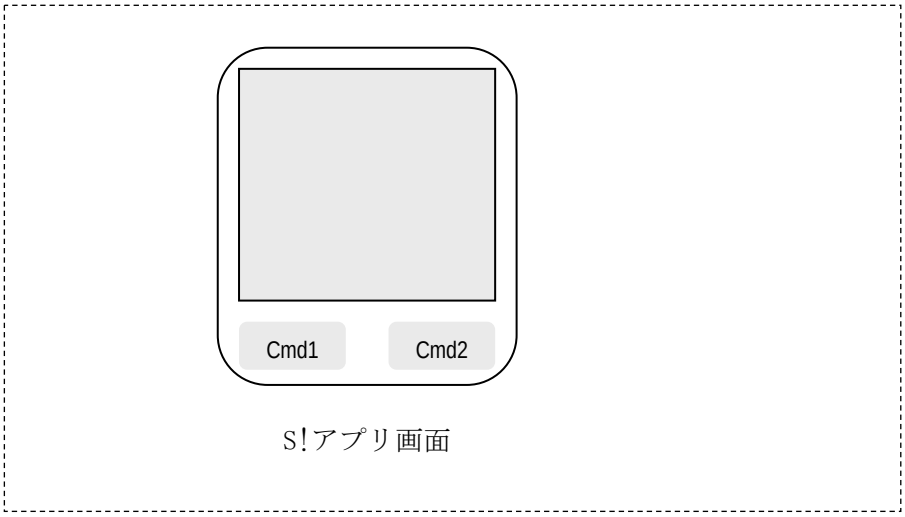
- １画面に表示できるソフトキーの個数より多くの Command インスタンスを同時に利用する際には、Command インスタンスはコマンドメニューで制御する。尚、低レベルグラフィックスを利用する際には、コマンドメニューの表示前後で画面の更新が発生することを考慮すること。
  - S!アプリからコマンドメニューに制御が移る時(Menu 表示時)
  - コマンドメニューから S!アプリに制御が戻る時(Menu 選択後)

  １画面に表示できるソフトキーの個数は全ての端末で同一とは限らない。そのため、全てのソフトキーを１画面で表示できる場合と、コマンドメニュー表示する場合とでは、描画タイミングが異なる点を留意する必要がある。以下に例を示す。

  [例]ソフトキーを２つ使用し、ソフトキーを押すと描画を更新(Canvas#repaint())する場合

  ※: ソフトキー表示領域が２つの場合

  ソフトキーを押すと描画を更新(Canvas#repaint ())する。



S!アプリ画面

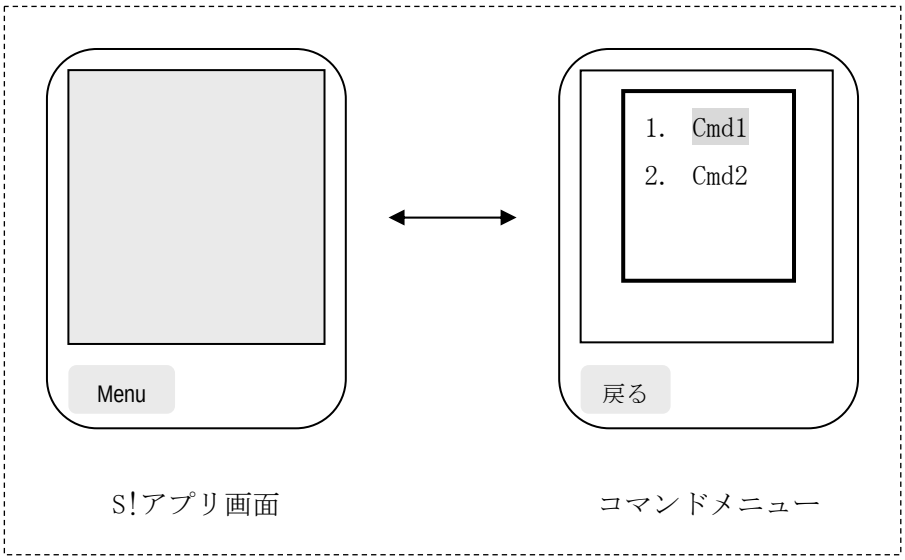
① 「Cmd1」または「Cmd2」を押下する。

② Canvas#repaint()が呼ばれる。

※：ソフトキー表示領域が１つの場合

Canvas#paint()の中でメインのロジックを記述しないこと。

Canvas#repaint ()が呼ばれることがあることを考慮すること。



① コマンドメニューを表示する時、Canvas#repaint()を呼ぶ。
② コマンドメニューを押下する。
③ 「1.Cmd1」または「2.Cmd2」を選択し押下する。Canvas#repaint ()を呼ぶ。
④ コマンドメニューからS!アプリ画面に戻る時Canvas#repaint ()を呼ぶ。
⑤ コマンドメニューを表示する時 Canvas#repaint ()を呼ぶ。
⑥ 「戻る」を押下した時、①が実行される。

- コマンドメニュー表示中はCanvasがカレントでなくなる為S!アプリの描画要求は無視する。S!アプリが一時停止しているわけではないので注意すること。
  コマンドメニューの「Command」を選択押下するもしくは「戻る」を押下すると Canvas がカレントになる為描画を更新する。
  低レベルグラフィックスでは以下の Notification を引き起こす。

  - Canvas がカレントになる　⇒Canvas#showNotify()を呼ぶ。
  - Canvas がカレントでなくなる⇒Canvas#hideNotify()を呼ぶ。

- S!アプリ作成時には、Canvas#showNotify()及びCanvas#hideNotify()を適宜、オーバーライドすることで意図したとおりの描画切り替えを行えるようにCanvas のサブクラスを実装すること。

- キーのリリースイベントを検知して選択処理を行う S!アプリでは、コマンドメニュー押下時のリリースイベント、リピートイベントは、コマンドメニューを表示した Canvas 画面に通知される為、キーのリリースイベントだけを判断して処理を進めず、プレスイベントとリリースイベントの対で判断すること。
- コマンドメニューが表示される時はCanvas#hideNotify()メソッドがコールバックされる。
- コマンドメニューが閉じられた時は Canvas#showNotify ()メソッドがコールバックされる。

4.3. 一時停止とメディアの再生

注意：
＜ PhrasePlayer 、MediaPlayer＞
MIDlet#pauseApp () の後に MediaPlayer または Phraseplayer を停止するので MIDlet#pauseApp () 内での状態を保持する。
MediaPlayer または Phraseplayer の再生中に一時停止を要求される場合は以下のようになる。

例：MediaPlayer 再生中に一時停止を要求される場合

① pauseApp () 内で何もしない
play 状態で一時停止されるので S!アプリ再開時に play 再開
② pauseApp () 内で play ()
play 中の play は無視される。上記と同様になる。
③ pauseApp () 内で pause ()
pause する。S!アプリ再開時も pause 状態である。
④ pauseApp () 内で stop ()
stop する。S!アプリ再開時も stop 状態である。

※ ただし、一部端末にて上記と異なる動作を行う場合がある。

4.4. 制御文字の表示

制限：

- 高レベルグラフィックスにおいて、制御文字 ￥ｎ（改行）が使用できる。改行以外の制御文字を指定した場合に、当該制御文字はないものとして表示する端末と、当該制御文字に替えて空白を挿入する端末がある。

4.5. ネットワークアクセス

注意：

- JAD ファイル、Manifest ファイルに「MIDlet-Permissions」の記述がない場合、あるいは「MIDlet-Permissions」の記述があるが、利用者が端末上でネットワークアクセスを禁止した場合には、SecurityException を発生する。

4.6. MIDlet セレクタ

制限：

- MIDlet を複数含んだ MIDlet スイートの形の場合には、いわゆる「MIDlet セレクタ」経由の起動になる。この場合、各 MIDlet の notifyPaused(), notifyDestroyed()による自力での状態移行の場合、下記の各種資源の後始末は S!アプリ側で行う必要がある。

    - ✧ MediaPlayer、PhrasePlayer の停止
    - ✧ バイブレーションとバックライトの初期化
    - ✧ その他 S!アプリが生成した資源（スレッド、メモリ資源など）

4.7. 音声認識

制限：

- MediaPlayer/PhrasePlayer との同時利用が不可能である。
  音声認識を行う際には、MediaPlayer/PhrasePlayer のインスタンスを破棄すること。

4.8. 端末上の制限

制限：

- S!アプリの作成にあたっては、PC/WS 上で実行するプログラムを作成する場合に比べ、以下に掲げる大きな制約があることを意識した上で作り込まれたい。
  ・端末の処理速度
  ・端末の記憶域サイズ
  ・バッテリーの持ち時間
- 端末では電話機としての必要な性能(連続通話時間、待受時間)、許容できる容積および重さを満たすために、バッテリーとの兼ね合いから計算資源については大きく制約されており、PC/WS が有する資源に比し質素なものである。S!アプリでは計算資源を必要とするため、処理速度、記憶域サイズおよびバッテリーの持ち時間に配慮された S!アプリ作りを配慮されたい。即ち、インスタンスを多量に生成することを避け、スレッドを多用することを避ける、といったプロセッサやメモリを節約するプログラミング作法が求められる。

4.9. 待受アプリ

- 待受アプリの場合には、MIDlet-1 に記載されている MIDlet のみが起動する。(MIDlet セレクタには移行しない)

5. データ

5.1. SMAF/Phrase

制限：

・master/slave 設定を用いる場合には、SMAF/Phrase データの EndPoint を全ての Track 同じ位置とする(演奏時間を全 Track 同じにする)。特に Phrase データの途中に EndPoint を置くと、演奏時間にズレが生じやすい為 EndPoint は必ず Phrase データの最後に置くようにする。

5.2. 3D オブジェクト

制限：

- テクスチャ読み込み時のヒープ消費サイズはテクスチャのサイズによらない。テクスチャ一枚きりのメモリ使用量は一律約 80KBytes となる。

5.3. PNG

- 下記の PNG フォーマットについては一色のみ全透過だけ可能である。

実装依存のチャンクデータ

チャンク	説明
tRNS	Transparency