

i アプリコンテンツ開発ガイド for Star-2.x
～ i アプリオプション・i アプリ拡張編 ～
第 1.02 版

平成 29 年 5 月 30 日
株 式 会 社 N T T ド コ モ



本製品または文書は著作権法により保護されており、その使用、複製、再頒布および逆コンパイルを制限するライセンスのもとにおいて頒布されます。NTT ドコモ（その他に許諾者がある場合は当該許諾者も含めて）の書面による事前の許可なく、本製品および関連する文書のいかなる部分も、いかなる方法によっても複製することが禁じられます。フォントを含む第三者のソフトウェアは、著作権法により保護されており、その提供者からライセンスを受けているものです。

Sun、Sun Microsystems、Java、J2ME および J2SE 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。サンのロゴマークは、米国 Sun Microsystems, Inc. の登録商標です。

「i モード」、「i アプリ／アイアプリ」、「i-α ppli」ロゴ、「DoJa」、「i メロディ／アイメロディ」、「トルカ」、「ToruCa」ロゴ、「i ウィジェット」、「i アプリオンライン」、「i アプリコール」、「ウィジェットアプリ」は NTT ドコモの商標または登録商標です。

FeliCa は、ソニー株式会社が開発した非接触 IC カードの技術方式です。FeliCa は、ソニー株式会社の登録商標です。本書では、i モード携帯電話に搭載されるモバイル FeliCa IC チップを単に FeliCa チップと呼称しています。

その他、掲載されている会社名、製品名、サービス名は各社の商標または登録商標です。

なお、本書では、コピーライト及び商標・登録商標表記はしておりません。

文書は現状有姿で提供されており、その商品性、特定目的への適合性、第三者の権利の非侵害を含み、明示的または黙示のいかなる条件、表明および保証も行われず、また提供されません。但し、かかる保証の否定が、適用される法律の下で無効である場合は、この限りではありません。

Copyright 2000-2017 NTT DOCOMO, INC. 2-11-1 Nagata-cho, Chiyoda-ku, Tokyo, 100-6150, Japan and Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California, 94303, U.S.A. All rights reserved.

本書は、Star-2.x プロファイルに対応するアプリケーション実行環境（本書中では i アプリ実行環境と呼びます）をターゲットとして記述されています。i アプリの仕様はプロファイルバージョンで識別できます。現在までのプロファイルバージョンの流れと主要な搭載機種は以下のとおりです。

Star-1.0 プロファイル：2008 年秋以降発売の Star プロファイル搭載携帯電話

Star-1.1 プロファイル：2009 年春以降発売の Star プロファイル搭載携帯電話

Star-1.2 プロファイル：2009 年秋以降発売の Star プロファイル搭載携帯電話

Star-1.3 プロファイル：2010 年春以降発売の Star プロファイル搭載携帯電話

Star-1.5 プロファイル：2010 年秋以降（2011 年春以降も含む）発売の Star プロファイル搭載携帯電話

Star-2.0 プロファイル：2011 年秋以降発売の Star プロファイル搭載携帯電話

各機種に搭載されている i アプリ実行環境のプロファイルバージョンは、別途 NTT ドコモよりアナウンスされます。

本書の内容は、以前のプロファイルから存在している機能であっても、以前のプロファイルにのみ対応した携帯電話には適用されませんのでご注意ください。本書で解説するプロファイルには、既存機能に関する未定義の仕様の明確化も含まれています。

目 次

第1章 はじめに	6
第2章 2D グラフィックス描画	10
2.1 ズーム座標系	10
2.2 ラスタオペレーション	11
2.3 その他	12
第3章 3D グラフィックス描画	14
3.1 高レベル 3D グラフィックス描画	14
第4章 サウンド再生	16
4.1 i モーションのオーディオトラックの再生	16
第5章 プラットフォームリソース制御	18
第6章 携帯電話ネイティブの データストレージアクセス	23
第7章 ポインティングデバイス	25
第8章 指紋センサー	27
第9章 アプリケーション連携拡張	29
9.1 映像データ管理機能の呼び出し	29
9.2 トルカ管理機能の呼び出し	30
9.2.1 Toruca クラス	30
9.2.2 トルカデータの新規登録	31
9.2.3 トルカデータの選択読み込み	32
9.2.4 トルカデータの ID 指定読み込み	32
9.2.5 トルカデータの検索取得	33
9.3 住所リンク機能からの i アプリの起動	34
9.4 スケジューラからの i アプリの起動	35
9.5 データ BOX 保存画像からの i アプリの起動	37
第10章 FeliCa チップ	38
10.1 i モード携帯電話に搭載される FeliCa チップの概要	38
10.2 フリーエリアアクセス API	40
10.3 フリーエリア利用のための ADF の設定	42
10.4 外部 R/W からのフリーエリアへのアクセス	43
10.5 フリー領域のセキュリティに関する注意事項	44
10.6 外部 R/W からの指示による携帯電話内の i アプリ起動	46
10.7 FeliCa チップによるアドホック通信	48
10.7.1 FeliCa アドホック通信 API の概要	49
10.7.2 通信リンク制御	50
10.7.3 データ送受信	53
第11章 UART 接続によるシリアル通信	57
第12章 外部メモリ	59
12.1 外部メモリ API の API 構成	59

12.2	外部メモリ API の利用	62
12.2.1	StorageDevice クラスの利用	62
12.2.2	フォルダの操作	64
12.2.3	ファイルの操作と入出力	65
12.3	外部メモリ API に固有の例外	66
12.4	外部メモリを利用するための ADF の設定	67
12.5	外部メモリ上のファイルのメディアプレーヤーでの再生	68
12.6	メーカー実装に起因する外部メモリ利用上の注意事項	69
12.6.1	暗号化機能利用時の注意事項	69
12.6.2	ファイル名に関する注意事項	69
第 13 章	デジタルテレビ制御	71
13.1	DTV 機能の連携起動	71
13.2	テレビ番組の視聴・録画予約の登録	72
第 14 章	動作認識	74
第 15 章	電子コンパス	77
第 16 章	加速度センサー	80
第 17 章	ヘルスケア関連機能	86
17.1	歩数計機能	86
17.2	身体活動量計機能	89
17.3	脈拍計機能	93
17.4	身体情報アクセス機能	95
17.5	フォーム分析機能	95
第 18 章	タッチパネルデバイス	98
第 19 章	シンセサイザ操作機能	101
19.1	音源デバイスへのメッセージの発行	101
19.2	UCS データによる発音	103
第 20 章	i アプリ-Flash 連携機能	105
20.1	i アプリ-Flash 連携概要	105
20.2	Flash プレイヤ	107
20.3	Flash コンテンツの再生	107
20.4	Flash コンテンツでのユーザー操作	109
20.5	Flash コンテンツからの携帯電話リソースの利用	109
20.6	i アプリとのイベントおよびデータのやり取り	110
20.7	Flash Lite 共有オブジェクトのアクセス	112
第 21 章	ユーティリティ API	114
21.1	暗号化機能	114
第 22 章	ジャイロセンサー	116
第 23 章	フルキーボード	121
第 24 章	赤外線リモコン受信機能	123

第 25 章 歩行方位取得機能	126
第 26 章 温湿度計機能	130
第 27 章 音楽プレイヤー機能	133
27.1 音楽プレイヤー機能の API 構成.....	133
27.2 音楽コンテンツ管理データベースへのアクセス.....	135
27.3 音楽プレイヤー制御	139
27.4 音楽プレイヤーイベント.....	141
27.5 Bluetooth 接続された外部機器からの操作イベント	141
第 28 章 カメラファインダ機能	143
第 29 章 音声合成機能	146
第 30 章 MascotCapsule eruptionST API による 3D グラフィックス描画	150
第 31 章 ステレオ表示機能	154
31.1 ステレオ表示機能 API の構成.....	154
31.2 i アプリにおけるステレオ表示機能の概要	155
31.3 ステレオ表示用キャンバス	158
31.4 ステレオ表示用グラフィックスの描画	160
付録 A ミニアプリにおける機能制限	163
A.1 ミニアプリで制限される機能	163
A.2 i アプリ API 利用上の注意点	165
A.2.1 デジタルテレビ制御	165
A.2.2 ヘルスケア関連機能	165

第 1 章

はじめに

i アプリ実行環境は、次の 4 つの API レイヤから構成されています。

- CLDC で規定されている API
- i アプリ基本 API
- i アプリオプション API
- i アプリ拡張 API

本書では、Star プロファイルにおいて i アプリオプション API および i アプリ拡張 API として採用されているものについての概要を解説します。

なお、CLDC で規定されている API については『Java 2 Platform, Micro Edition (J2ME) Connected Limited Device Configuration (CLDC)』（<http://java.sun.com/products/cldc/>）を、i アプリ実行環境の API レイヤ構成および i アプリ基本 API の詳細については『i アプリコンテンツ開発ガイド 詳細編』を参照してください。

i アプリオプション API および i アプリ拡張 API には、以下のような機能カテゴリに属する API が含まれています。

大分類	小分類	解説
2D グラフィックス描画	ズーム座標系	グラフィックスコンテキストに描画を行う際に使用する座標系として、256 倍に拡大された座標系を使用します。
	ラスタオペレーション	描画元のピクセルの色とそれに対応する描画先ピクセルの色を一定のルールで演算し、その結果得られた色でレンダリングを行います
	その他	2D グラフィックス描画に関する、上記分類に属さない単機能的なメソッドを提供します。
3D グラフィックス描画	高レベル 3D グラフィックス描画	3D で表現されたモデルの描画を簡単な API 呼び出しで行うための API を提供します。3D モデルにはポーズ（アクション）を取らせることができます。また、回転や視点移動を行うための演算もサポートしています。 本機能においては、3D モデルデータのデータフォーマットはメーカーにより異なる可能性があります。従って、本機能は i アプリ拡張 API に属します。 なお、本機能をベースとした 3D グラフィックス描画機能が i アプリ基本 API として規定されていますが、そちらは本機能とは互換性はありません。
サウンド再生	i モーションのオーディオトラックの再生	オーディオトラックのみで構成された i モーション形式データを i アプリの動作と並行して再生することができます。
プラットフォームリソース制御	追加キー制御	テンキー、上下左右キー決定キー、ファンクションキー以外の特殊なキーをサポートします。
	アロケート可能な連続メモリ量の参照	i アプリ実行環境のヒープ上で、現在どの程度のサイズの連続したメモリバッファをアロケート可能かを参照します。
	オーディオ出力のサラウンド制御	サラウンド機能をサポートしている携帯電話において、i アプリからのサウンド再生時のサラウンドのオン、オフを制御します。

	ディスプレイスタイル状態の参照	ディスプレイ方向を自由に操作できる機種において、現在のディスプレイスタイルを参照します。
	キーアサイン制御	ディスプレイ方向の自由な操作が可能で、さらにディスプレイ方向に連動してキーアサイン（キー配置）を切替えることができる機種において、キーアサインを固定とするか、ディスプレイ方向に連動させるかを制御します。
	電池パックの満充電までの残り充電時間の参照	充電中の状態において、満充電までの残りの充電時間を参照します。
	電池パックの残り端末利用時間の参照	電池パックが動作中（充電中でない）の状態において、現在のアプリケーションが継続される前提で携帯電話がどのくらい充電しないです利用可能であるかを参照します。
	方位デバイスの基準方向の制御	電子コンパス機能や歩行方位取得機能を利用して携帯電話の向きの方位角を取得する際の、基準となる方向を制御します。
携帯電話ネイティブのデータストレージアクセス	-----	携帯電話ネイティブのデータストレージとは、待ち受け画像など携帯電話のネイティブアプリケーションが使用するデータストレージです。この機能は、このデータストレージに i アプリがアクセスする機能を提供します。 ただし i アプリからアクセス可能な領域は、データストレージの中で i アプリ用に用意された一部分だけであり、任意のデータストレージ領域にアクセスすることはできません。
ポインティングデバイス	-----	ポインティングデバイスをアプリケーションから使用します。ユーザーはポインティングデバイスを使用して、画面上を自由に移動可能なカーソルを操作したり、ジョイスティックのように方向を指示したりすることができます。 ポインティングデバイスは、 Canvas 上でのみ利用することができます。
指紋センサー	指紋認証	指紋センサー機能を搭載している携帯電話において、ネイティブの指紋認証機能呼び出しします。 指紋認証機能を搭載している携帯電話では、認証に使用する指紋データをあらかじめ携帯電話に登録しておくことができます。本機能を利用することで、i アプリは現在その i アプリを操作しているユーザーの指紋と携帯電話に登録されている指紋データの照合を行うことができます。
アプリケーション連携拡張	映像データ管理機能の呼び出し	ネイティブの映像データ保存領域にアクセスし、保存されている映像（i モーション）データを i アプリから利用するための機能を提供します。
	トルカ管理機能の呼び出し	i アプリがネイティブのトルカ管理機能呼び出し、トルカデータの登録や参照、検索を行う機能を提供します（トルカは FOMA 902i シリーズから導入されたサービスです）。
	住所リンク機能からの i アプリの起動	ブラウザで閲覧中の HTML に埋め込まれた住所リンクから i アプリを起動する機能を提供します。i アプリは、起動パラメータとして住所リンクに記述された位置情報（緯度経度や住所などの情報）を受け取ることができます。
	スケジューラからの i アプリの起動	スケジュールデータに埋め込まれたリンクから i アプリを起動する機能を提供します。地図プラットフォームや番組表 i アプリに設定された i アプリを指定して起動することもできます。
	データ BOX 保存画像からの i アプリの起動	データ BOX に保存された画像からメニュー操作により、画像を取り扱う i アプリとしてその携帯電話に登録されている i アプリを起動する機能を提供します。
FeliCa チップ	-----	携帯電話に搭載している FeliCa チップにアクセスするためのインタフェースを提供します。本書では、一般の i アプリからも使用可能なフリー領域、フリーエリアについて解説しています。
UART 接続によるシリアル通信	-----	FOMA 携帯電話の外部接続端子と外部機器を接続ケーブルで接続（UART 接続）し、シリアル通信を行う機能を提供します。
外部メモリ	-----	携帯電話に搭載された外部メモリデバイスを使用して、外部メモリを i アプリのデータストレージとして使用する機能を提供します。
デジタルテレビ制御	-----	携帯電話に搭載されたデジタルテレビ（DTV）機能を利用して、i アプリから DTV 機能を連携起動したり、テレビ番組の視聴・録画予約を登録する機能を提供します。
動作認識	-----	携帯電話のカメラデバイスから得られる画像をリアルタイムに解析し、画像の時系列的な変化を携帯電話の機体の移動量として i アプリに伝える機能を提供します。
電子コンパス	-----	携帯電話の機体の向いている方向角を調べる機能を提供します。

加速度センサー	----	加速度センサーデバイスを用いて、携帯電話の機体にかかる加速度や機体の傾き具合などを調べる機能を提供します。
ユーティリティ API	暗号化機能	パスワードベースの暗号化機能を提供します。
ヘルスケア関連機能	歩数計機能	携帯電話ネイティブの歩数計（万歩計）機能にアクセスし、歩数計機能が保持している歩数データなどの情報を取得する機能を提供します。
	身体活動量計機能	身体活動量とは、日々の生活の中で行われる運動の量を定量化するための考え方です。身体活動量データは、携帯電話ネイティブの身体活動量計機能により蓄積されます。この機能は、携帯電話ネイティブの身体活動量計機能にアクセスし、身体活動量データなどを取得する機能を提供します。
	脈拍計機能	カメラデバイスで撮影された指の画像をリアルタイムに解析して、脈拍データを取得する機能を提供します。
	身体情報アクセス機能	携帯電話ネイティブが保持する身長や体重といった身体情報の設定や参照を行う機能を提供します。
	フォーム分析機能	携帯電話ネイティブのフォーム分析機能にアクセスし、ユーザーの身体の動きに関する情報を取得する機能を提供します。
タッチパネルデバイス	----	タッチパネルデバイスをアプリケーションから使用します。i アプリは、ユーザーのタッチパネルへのタッチ操作をイベントとして受け取ったり、タッチされた位置などの情報を取得することができます。 タッチパネルデバイスは、Canvas 上でのみ利用することができます。
シンセサイザ機能	----	音源デバイスにリアルタイムにメッセージを発行して発音させるシンセサイザ機能を提供します。UCS（User Customized Sound）の利用もサポートしています。
i アプリ-Flash 連携機	----	i アプリから Flash プレイヤを呼び出し、Flash コンテンツ（SWF ファイル）を再生する機能を提供します。
ジャイロセンサー	----	ジャイロセンサーデバイスを用いて、携帯電話の機体の回転運動を調べる機能を提供します。
フルキーボード	----	フルキーボードのキー操作を i アプリへ通知する機能を提供します。
赤外線リモコン受信機能	----	外部から送信された赤外線リモコン信号を携帯電話側で受信する機能を提供します。
歩行方位取得機能	----	携帯電話が備える加速度センサーや電子コンパスを用いて、i アプリ実行中の任意の時点、もしくは i アプリ実行中の特定の期間内に行われたユーザーの歩行について、歩行の向き（方位）などの情報を取得する機能を提供します。
温湿度計機能	----	温湿度センサーデバイスを用いて、現在の温度・湿度情報を取得したり、携帯電話ネイティブの温湿度計機能が保持している温度・湿度情報を取得する機能を提供します。
音楽プレイヤー機能	----	i アプリからデータ BOX に含まれる音楽コンテンツの再生制御、プレイリスト表示ミュージック情報表示、フィルタリングなどを行う機能を提供します。
カメラファインダ機能	----	カメラデバイスから得られる画像をリアルタイムにディスプレイ上に表示する機能を提供します。
音声合成機能	----	String オブジェクトの形で表現されたテキストを音声出力する機能を提供します。
MascotCapsule eruptionST API による 3D グラフィックス描画	----	3D で表現されたモデルの描画を簡単な API 呼び出しで行うための API を提供します。 本機能に対応した機種では、3D グラフィックスエンジンとして Mascotcapsule eruptionST を搭載しています。 また、ステレオ表示に対応する描画もサポートしています。
ステレオ表示機能	----	視差バリアにより立体的に知覚できる描画を行う機能を提供します。

以降、各機能について概要を解説します。各機能で使用するメソッドの詳細については、『i アプリコンテンツ開発ガイド API リファレンス編』または『i アプリコンテンツ開発ガイド オプション API リファレンス編』を参照してください。

なお、これらの API はメソッド単位、フィールド単位でメーカーが実装の有無を決定します。各機種におけるメソッドやフィールドのサポート状況は、別途 NTT ドコモより公開されます。

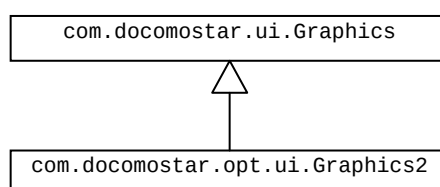
本書で規定される機能には、将来のプロファイルにおいて i アプリ基本 API に移行する可能性のあるものも含まれています。ただし、逆に将来仕様が変更または削除される可能性のあるものも含まれていますのでご注意ください。

第 2 章

2D グラフィックス描画

2D グラフィックス描画機能では、i アプリ基本 API における低レベルグラフィックス描画およびイメージ処理を拡張します。これらの API を使用することで、アプリケーションによる画面の表示に特殊な効果を与えることができます。

2D グラフィックス描画機能を構成するクラス群を以下に示します。



2D グラフィックス描画のいずれかの機能をサポートする携帯電話では、`Canvas.getGraphics()` などによりシステムから取得するグラフィックスコンテキストは `com.docomostar.opt.ui.Graphics2` クラス（拡張グラフィックスコンテキスト）のオブジェクトとなっています。

2.1 ズーム座標系

`com.docomostar.opt.ui.Graphics2` クラスでは描画時に指定する座標位置について、ディスプレイ上のピクセル位置と座標系が 1 対 1 に対応するノーマルモードの他に、座標系が実際のピクセル位置の 256 倍に対応するズームモードをサポートします。

座標系の切り替えは、`Graphics2.setCoordinateMode()` メソッドを使用して行います。このメソッドの引数に `Graphics2.CM_NORMAL` を指定するとノーマルモードに、`Graphics2.CM_ZOOM` を指定するとズームモードになります。

ズームモードでは、実際に描画される位置は描画メソッド呼び出し時に指定された位置の 1/256 のスケールになります。例えばズームモードにおいて

```
drawLine( 2560, 2560, 25600, 25600 );
```

という直線を描画した場合、ディスプレイ上は (10, 10) - (100, 100) の 2 点を結ぶ直線が表示されます。

なお、ズームモードにおいては `Graphics.setOrigin()` で設定されている座標オフセットもズームの対象となります。すなわち、上記直線を描画する際に座標オフセットが

```
setOrigin( 5120, 5120 );
```

と設定されていた場合、ディスプレイ上は $(30, 30) - (120, 120)$ の 2 点を結ぶ直線が表示されます(*1)。

(*1) 詳細には、 $((2560+5120)/256, (2560+5120)/256) - ((25600+5120)/256, (25600+5120)/256)$ を結ぶ直線となります。

2.2 ラスタオペレーション

`Graphics2` クラスでは、描画元のピクセルの色とそれに対応する描画先ピクセルの色を一定のルールで演算し、その結果得られた色でレンダリングを行う機能（ラスタオペレーション）をサポートしています。

レンダリング時にどのような色演算を使用するかは、`Graphics2.setRenderMode()` メソッドで指定します。このメソッドの引数には演算方法、描画元の色への乗数、描画先の色への乗数を指定します。乗数は 1.0 を 256 分割した単位（255 を設定すると 1.0）にて、0～255 の範囲で設定します。以下に、サポートされる演算方法とその内容を解説します。以下の説明において、描画元のピクセル色の RGB 値(S_R, S_G, S_B)の各要素を S_i 、描画先のピクセル色の RGB 値(D_R, D_G, D_B)の各要素を D_i とします。また、`srcRatio` は描画元の色への乗数、`dstRatio` は描画先の色への乗数を示します。

- `Graphics2.OP_ADD`（加算）

描画元のピクセル色の値と描画先のピクセル色の値を加算します。計算式は以下の通りです。

$$D_i = D_i * \text{dstRatio} + S_i * \text{srcRatio}$$

- `Graphics2.OP_REPL`（置き換え）

描画元のピクセル色の値で、描画先のピクセル色を置き換えます。計算式は以下の通りです。

$$D_i = S_i * \text{srcRatio}$$

`srcRatio` が 1 の場合、ラスタオペレーションを使用しないレンダリングと同じ意味になります。

- `Graphics2.OP_SUB`（減算）

描画先のピクセル色の値から描画元のピクセル色の値を減算します。計算式は以下の通りです。

$$D_i = D_i * \text{dstRatio} - S_i * \text{srcRatio}$$

2.3 その他

Graphics クラスおよび Graphics2 クラスでは、前項までに解説した分類以外にも、2D グラフィックス描画に関する単機能的なメソッドをいくつか提供しています。以下、各機能について解説します。

アニメーション GIF 内の特定フレームのイメージ描画 (Graphics2.drawNthImage())

引数で指定されたアニメーション GIF のメディアイメージ内の、特定のフレームのイメージを指定位置に描画します。

数値の描画 (Graphics2.drawNumber())

引数で指定された int 型の数値を、指定された桁数が収まる領域に右詰めで描画します。

グラフィックスコンテキストからのイメージ生成 (Graphics2.getImage())

引数で指定されたグラフィックスコンテキスト上の矩形領域の内容が描画されている、新たなイメージを生成します。

2 色の中間色の取得 (Graphics2.getIntermediateColor())

2 つの色の中間色を求めます。このメソッドの引数には、2 つの色の色情報、および 2 色間の色空間上の距離（2 色間の距離を 255 とする、0～255 までの整数値）を指定します。なお色の情報には、色を表す機種固有の整数値 (Graphics.getColorOfRGB()) メソッドなどで返される値) が使用されます。

アフィン変換付きイメージ描画 (Graphics2.drawImage())

イメージ上の各座標に以下のような 3 行 3 列のアフィン変換行列を適用し、求められた座標群への描画を行ないます。以下の式において、x、y は元のイメージ上の座標を、x'、y' は変換後の座標（描画対象における座標）を、m00～m12 はアプリケーションプログラムから指定可能なアフィン変換行列要素を示します。

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \frac{1}{4096} \begin{bmatrix} m00 & m01 & m02 \\ m10 & m11 & m12 \\ 0 & 0 & 4096 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

この機能を使用することにより、イメージに拡大縮小・回転などの変換を加えて描画を行なうことができます。Graphics2 クラスの drawImage() メソッドでは、アフィン変換行列には高レベル 3D グラフィックス描画機能の com.docomostar.opt.ui.j3d.AffineTrans クラスを使用します。

表示デバイスの垂直同期間隔の取得 (Graphics2.getSyncUnlockInterval())

表示デバイスの垂直同期の間隔を、μ 秒単位で取得します。例えば 1 秒間に垂直同期を 60 回行う表示デバイスでは、このメソッドは 16667 (μ 秒) を返します。このメソッドが返す値は、次項に示す、Graphics2.syncUnlock() メソッドを使用する際の参考情報となります。

表示デバイスの垂直同期タイミングでのダブルバッファリング終了 (**Graphics2.syncUnlock()**)

i アプリの提供するダブルバッファリング機能について、ダブルバッファリングの終了処理 (**Graphics.unlock()** メソッドの処理) を表示デバイスの垂直同期タイミングに合わせて行う機能を提供します。**Graphics.unlock()** メソッドの代わりに **Graphics2.syncUnlock()** メソッドを使用すると、表示デバイスの垂直同期と合致するタイミングでダブルバッファリングの終了処理が行われます。垂直同期のタイミングでダブルバッファリングを終了させることで、ちらつきのより少ない、スムーズな表示を行うことができます。

引数に垂直同期の回数を指定することで、前回のダブルバッファリング終了から何回目の垂直同期を待つかを指定することもできます。1 回の垂直同期に要する時間（垂直同期の間隔）は、前項に記載した **Graphics2.getSyncUnlockInterval()** メソッドを使用して調べることができます。

このメソッドを呼び出した時点からダブルバッファリングの終了処理を行わせるタイミングまでに間隔があいている場合、このメソッドは処理が完了するまでブロックします。このメソッドがブロックしている間は、i アプリ実行環境上でスレッドスイッチは行われず、他のスレッドの処理やイベント通知の処理も行われません。このため、本機能を使用して長時間ウェイトする（引数に指定する垂直同期回数を大きく取る）ようなことは行わないようにしてください。なお、携帯電話システムの実装によっては、引数に指定可能な垂直同期回数に上限を設けている場合もあります。

第 3 章

3D グラフィックス描画

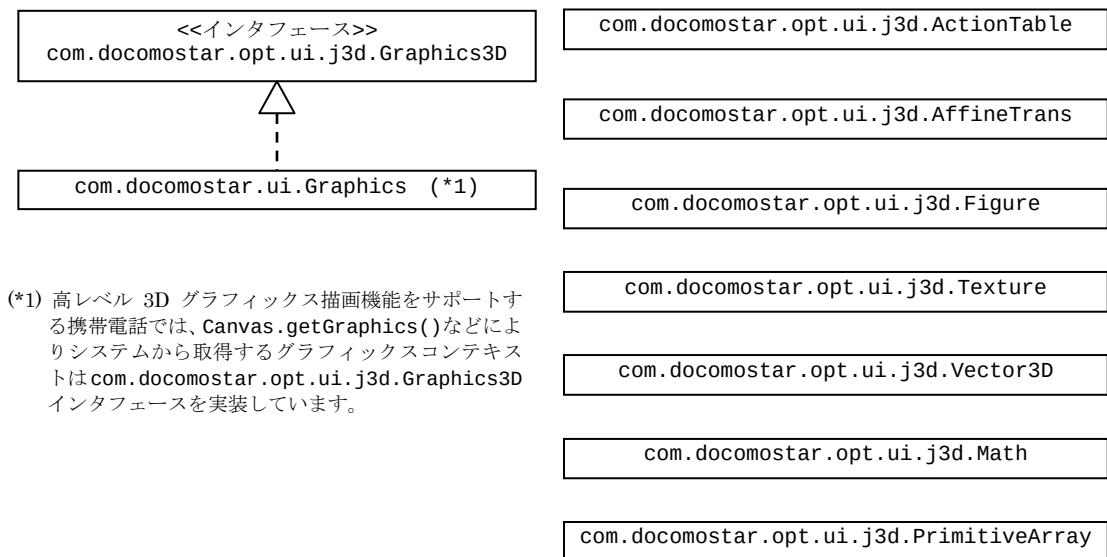
3D グラフィックス描画機能では、外部から取り込んだ 3D モデルデータを操作、表示する機能を提供します。

なお、本章で解説する機能に相当する機能は i アプリ基本 API にも取り入れられています。ただし、i アプリ基本 API の 3D グラフィックス描画機能と i アプリ拡張 API の 3D グラフィックス描画機能 API とは互換性はありません。

3.1 高レベル 3D グラフィックス描画

高レベル 3D グラフィックス描画機能では、比較的簡単な API 呼び出しで 3D モデルの描画を行うための API セットを提供します。

高レベル 3D グラフィックス描画機能を構成するクラス群を以下に示します。



(*1) 高レベル 3D グラフィックス描画機能をサポートする携帯電話では、`Canvas.getGraphics()`などによりシステムから取得するグラフィックスコンテキストは `com.docomostar.opt.ui.j3d.Graphics3D` インタフェースを実装しています。

各クラス、インタフェースの機能概要は以下の通りです。

クラス・インタフェース	機能概要
Graphics3D	高レベル 3D グラフィックス描画機能をサポートするグラフィックスコンテキストが備えるべきメソッドを定義するインタフェースです。
ActionTable	3D モデルのアクション（アニメーションデータ）を保持するクラスです。アニメーションデータは市販の 3D オーサリングツールなどを利用して作成します。 ある特定の瞬間に 3D モデルが取るべきポーズをフレームと呼び、アクションは複数のフレームのシーケンスとして構成されます。 ActionTable には複数のアクションを保持することができます。後述の Figure オブジェクトにフレームを設定することで、3D モデルにポーズを取らせることができます。
AffineTrans	3 次元アフィン変換用の行列を扱うクラスです。3D モデルの回転や視点の移動には、このクラスのオブジェクトを使用します。
Figure	3D モデルのデータを保持するクラスです。3D モデルを描画するには、 Graphics3D.drawFigure() メソッドを使用します。
Texture	テクスチャのデータを保持するクラスです。テクスチャには、3D モデルのポリゴンに設定するモデルマッピングテクスチャと、環境マッピング使用時に環境球に設定する環境マッピングテクスチャの 2 種類があります。本クラスのオブジェクトを生成する際に、どちらの用途のテクスチャであるかを指定します。
Vector3D	3 次元の固定小数点数ベクトルを表すクラスです。内積・外積計算、正規化といったユーティリティメソッドを備えています。
Math	高レベル 3D グラフィックス描画機能で使用する単位系に適合した数値演算ユーティリティです。なお、このクラスで取り扱える数値の精度はメーカーにより異なる場合があります。
PrimitiveArray	プリミティブ（点や線、面などの図形）を描画する際に用いる頂点情報を格納するための、プリミティブ配列クラスです。

高レベル 3D グラフィックス描画機能で使用するクラス・メソッドでは、整数型を使用して擬似的に小数点以下の値を扱うための独自の単位系を導入しています。導入されている単位系の詳細は API リファレンスを参照してください。

なお、現在のプロファイルでは 3D グラフィックスデータのフォーマットは規定していません。このため使用する 3D グラフィックスデータは本機能を採用するメーカーにより異なる可能性があり、本機能は i アプリ拡張 API に位置付けられます。

メーカーの採用している 3D グラフィックスデータのフォーマットやコンテンツの作成方法は、別途メーカーより公開されます。

第 4 章

サウンド再生

サウンド再生機能では、i アプリ基本 API の **AudioPresenter** の拡張機能を提供します。サウンド再生機能は以下により構成されます。

- `com.docomostar.ui.AudioPresenter` に定義される拡張メソッド
- `com.docomostar.ui.AudioTrackPresenter`

サウンド再生機能で拡張される機能には、以下があります。

- i モーションのオーディオトラックの再生

以下、機能の概略を解説します。

4.1 i モーションのオーディオトラックの再生

FOMA 携帯電話では、映像コンテンツサービスとして i モーションをサポートしています。FOMA 携帯電話には、**AudioTrackPresenter** を使用することで i モーションのオーディオトラックをサウンドとして再生することができるものがあります。**NativeMoviePlayer** による i モーション再生と異なり、**AudioTrackPresenter** による i モーション再生ではサスペンドによって i アプリの動作が中断されることはありません。i アプリの動作と並行してサウンド再生を行うことができるため、BGM のような用途などにも i モーションの音声データを使用することができるようになります。

i アプリが **AudioTrackPresenter** オブジェクトを取得するには、**AudioPresenter** クラスの static メソッドである `getAudioTrackPresenter()` メソッドを呼び出します。

AudioTrackPresenter は **AudioPresenter** クラスを継承したクラスですが、その機能は i モーションのオーディオトラック再生のために特化しており、通常の **AudioPresenter** クラスでは利用可能ないくつかの機能が制限されています。

以下に、**AudioTrackPresenter** クラスと **AudioPresenter** クラスの違いについて解説します。

- **AudioTrackPresenter** クラスでは、メディアデータとして、i モーション形式データから生成した **MediaMovie** オブジェクトを使用します。メディアデータをセットするには、**AudioTrackPresenter** クラスに固有のメソッドである `setSound(MediaMovie)` を使用してください。**AudioPresenter** ク

ラスから継承されている、**MediaSound** を引数に取る **setSound()** メソッドを使用することはできません。

なお、使用する i モーション形式データは、オーディオトラックのみで構成されたものでなければなりません（ビデオトラックやテキストトラックを含むものであってはなりません）。また、プログレッシブダウンロード用に作成されたデータを使用することはできません。

- **AudioTrackPresenter** オブジェクトを複数並列に同時再生することはできません。また、**AudioTrackPresenter** オブジェクトと **AudioPresenter** オブジェクトの同時再生についても、メーカーによってはサポートされないか、もしくは同時再生数に通常より厳しい制限を持つ場合があります。

AudioTrackPresenter オブジェクトと **AudioPresenter** オブジェクトの同時再生をサポートする機種では、**AudioTrackPresenter** の優先度はポート指定のない、最高優先度の **AudioPresenter** と同等となります。

- **AudioTrackPresenter** クラスでは、3D サウンド制御、および同期イベントを使用することはできません。
- **setAttribute()** によって制御できる属性は、**AudioTrackPresenter** クラスでは制限されます。どのメーカーでも制御可能な項目はボリューム (**AudioPresenter.SET_VOLUME**) のみです。

【Star-2.0】

Star-2.0 プロファイル以降、i モーションのオーディオトラック再生機能は i アプリ基本 API となりました。

第 5 章

プラットフォームリソース制御

現在のプロファイルの i アプリ基本 API では、以下のプラットフォームリソースが `com.docomostar.system.PhoneSystem` クラスに定義されています。

- ・ ディスプレイのバックライト
- ・ バイブレータ
- ・ 機体の折りたたみ状態（折りたたみ型およびフリップ型の携帯電話でのみ有効）
- ・ メール・メッセージ受信状態
- ・ 電池パック残量
- ・ 電源の種別
- ・ 電波強度
- ・ マナーモード設定状況
- ・ エリア情報
- ・ 画面・可視不可視状態
- ・ 携帯電話組み込みの効果音

プラットフォームリソース制御の拡張では、`com.docomostar.system.PhoneSystem` クラス、`com.docomostar.opt.system.PhoneSystem2` クラス、`com.docomostar.ui.Display` クラスに対し、以下のプラットフォームリソースを追加定義しています。なお、`PhoneSystem2` クラスは `PhoneSystem` クラスのサブクラスです。

- ・ 追加キー制御（`Display` クラス、`PhoneSystem` クラス）
- ・ アロケート可能な連続メモリ量の参照（`PhoneSystem2` クラス）
- ・ オーディオ出力のサラウンド制御（`PhoneSystem` クラス）
- ・ ディスプレイスタイル状態の参照（`PhoneSystem2` クラス）
- ・ キーアサイン制御（`PhoneSystem2` クラス）
- ・ 電池パックの満充電までの残り充電時間の参照（`PhoneSystem` クラス）
- ・ 電池パックの残り端末利用時間の参照（`PhoneSystem` クラス）
- ・ 方位デバイスの基準方向の制御（`PhoneSystem` クラス）

【Star-1.1】

- ・ Star-1.1 プロファイル以降、プラットフォームリソース制御の拡張として、電池パックの満充電までの残り充電時間の参照と電池パックの残り端末利用時間の参照の 2 つが追加定義されました。

【Star-1.5】

- ・ Star-1.5 プロファイル以降、プラットフォームリソース制御の拡張として、方位デバイスの基準方向の制御が追加定義されました。

以下、各機能の概要について解説します。

<追加キー制御>

i アプリ基本 API では、キーイベントの発生するキーとして数字キー (*と#を含む)、上下左右キー、選択 (決定) キーおよび 4 つのファンクションキーを規定しています。追加キー制御では、これら以外のキーをサポートします。

ページダウンキー、ページアップキーについては、対応する定数として、`com.docomostar.ui.Display` クラスに `KEY_PAGE_DOWN` および `KEY_PAGE_UP` が定義されています。

また、携帯電話にページダウンキー、ページアップキー以外にも備えられているさまざまなオプションのハードウェアキーを取り扱うためのフレームワークが定義されています。

このフレームワークでは、キーをいくつかのキーグループに分類します。各グループには番号が与えられており、個々のキーが何番目のグループに属しているかは、そのキーの値を 5 ビット分右にシフトすることにより知ることができます。現在のプロファイルでは、キーグループ 0 (キー値 0x00~0x1f) およびキーグループ 1 (キー値 0x20~0x3f) の 2 つのキーグループが規定されています。

初期状態ではキーグループ 0 は有効ですが、その他のキーグループは全て無効となっています。無効状態のキーグループに属するキーのキーイベントが i アプリに通知されることはありません。キーグループを有効にするには `PhoneSystem.setAttribute()` メソッドを使用して、`PhoneSystem.DEV_KEYPAD` 属性に対し有効にしたいキーグループのグループ番号を指定します (`PhoneSystem.DEV_KEYPAD` は設定のみ可能な属性です)。

キーグループ 1 を有効化する例：

```
PhoneSystem.setAttribute( PhoneSystem.DEV_KEYPAD, 1 );
```

この操作により、有効化されたキーグループに属するキーについてキーイベントが発生するようになります。また有効化されたキーグループについては、そのキーグループに属するキーのキーパッド状態を知るため、`Canvas.getKeypadState()` メソッドの引数にキーグループ番号 (int 型) を指定して呼び出すことができます。

キーグループの有効化は、i アプリが起動して最初にカレントフレームの設定

(`Display.setCurrent()` メソッドの呼び出し) を行なう前に行なう必要があります。また、一度有効化されたキーグループを、その i アプリが起動している間に再度無効化することはできません。

なお、追加キーは低レベル API において使用可能であり、高レベル API で使用できるかどうかはメーカーにより異なります。また搭載されるキーのハードウェア仕様 (特性) などによっては、イベントまたはキーパッド状態のいずれか一方が取得できない場合もあります。

【Star-1.2】

Star-1.2 プロファイルにて、フルキーボード機能が追加され、フルキーボードのキーに対応した新たなキーグループが追加されました。フルキーボード、およびそのキーグループの詳細は、第 23 章を参照してください。

<アロケート可能な連続メモリ量の参照>

i アプリ実行環境のヒープメモリ上に現時点でどの程度のサイズの連続メモリバッファをアロケートすることができるかを示す、参照のみ可能な属性です。アロケート可能な連続メモリ量を参照するには、属性 `PhoneSystem2.DEV_ALLOCATABLE_JAVA_MEMORY` を使用します。

仮想マシンがヒープコンパクション機能 (ヒープ上に発生したフラグメントの除去機能) をサポートしている機種では、このメモリ量は概ね `java.lang.Runtime.freeMemory()` メソッドが返す値と同等となります。しかしヒープコンパクション機能をサポートしていない機種では、このメモリ量は状況により `Runtime.freeMemory()` メソッドが返す値より大幅に小さくなる場合があります。

`Runtime.freeMemory()` メソッドが返す値が十分に大きい場合でも、アロケート可能な連続メモリ量を超えるサイズのバッファを確保しようとすると `OutOfMemoryError` などのエラーや例外が発生します。

＜オーディオ出力のサラウンド制御＞

オーディオ出力のサラウンド機能のオン、オフを制御するための、参照・設定の両方が可能な属性です。属性 `PhoneSystem.DEV_AUDIO_SURROUND` に対し、`PhoneSystem.ATTR_SURROUND_ON` または `PhoneSystem.ATTR_SURROUND_OFF` を設定することでサラウンド機能を制御します。

なお、サラウンド機能と 3D サウンド制御機能を同時に使用することはできません。また、`AudioPresenter` でサウンドを再生しているときにサラウンド機能のオン、オフを切り替えることはできません。

＜ディスプレイスタイル状態の参照＞

ディスプレイ方向を自由に操作できる機種において、現在のディスプレイスタイルを示す、参照のみ可能な属性です。ディスプレイスタイルを参照するには、属性 `PhoneSystem2.DEV_DISPLAY_STYLE` を使用します。このメソッドが返す値は、`PhoneSystem2` に定義されている以下の属性値です。

- ・ `ATTR_DISPLAY_STYLE_VERTICAL` （ディスプレイを縦型にした、通常の状態）
- ・ `ATTR_DISPLAY_STYLE_HORIZONTAL_RIGHT` （ディスプレイを右側に 90 度回転させた状態）
- ・ `ATTR_DISPLAY_STYLE_HORIZONTAL_LEFT` （ディスプレイを左側に 90 度回転させた状態）
- ・ `ATTR_DISPLAY_STYLE_REVERSE` （ディスプレイを表にして折りたたまれた状態：リバースタイル）

なお、本機能をサポートする機種では、機体のディスプレイスタイルが変化すると `Canvas.processEvent()` メソッドに `com.docomostar.opt.ui.Display2.DISPLAY_STYLE_CHANGED_EVENT` イベントが通知されます。`processEvent()` メソッドの第 2 引数には、その際のディスプレイスタイルに応じて、上記に挙げた属性値が引き渡されます。

ただし、キャンバス上の座標系の原点は、ディスプレイスタイル状態によらず、ディスプレイを縦型にした状態におけるディスプレイ左上隅にあります。このため、ディスプレイを横に倒したスタイルでの i アプリの使用を考える場合、アプリケーションプログラム側で座標系の変換や回転表示などの処理を行う必要があります。

このような場合に必要となる座標系の変換処理をシステム側で行うため、`Canvas2` クラスが i アプリオプション API として `com.docomostar.opt.ui` パッケージに定義されています。`Canvas2` クラスをサポートする機種では、ディスプレイの向きに応じてアプリケーションプログラムが適切にキャンバスの座標系を指示（右もしくは左に 90 度回転）することで、実際の描画命令での煩雑な座標変換や回転表示などの処理を省くことができます。

この機能を利用してキャンバスの座標系を回転させた場合、i アプリの描画領域は、この機能を使わない場合の描画領域の幅と高さを入れ替えた形（ADF に `DrawArea` キーを設定している i アプリでは、キーに指定している幅と高さを入れ替えた形）となります。

＜キーアサイン制御＞

ディスプレイ方向の自由な操作が可能で、さらにディスプレイ方向に連動してキーアサイン（キー配置）を切替えることができる機種において、ディスプレイを縦型にした通常の状態のままキーアサインを固定とするか、ディスプレイ方向に連動してキーアサインを切替えるかを制御します。属性 `PhoneSystem2.DEV_KEY_ASSIGN` に対して、`PhoneSystem2.ATTR_KEY_ASSIGN_DEFAULT` または `ATTR_KEY_ASSIGN_DEPEND_ON_DISPLAY_STYLE` を設定することができます。キーアサインを `ATTR_KEY_ASSIGN_DEPEND_ON_DISPLAY_STYLE` に設定した状態で、ディスプレイスタイルが切り替わると、スタイルの切り替わりに伴いキーアサインも切り替わり、切り替わったキーアサインに基づいてキーイベントが発生するようになります。

キーアサインの設定は、i アプリが起動して最初にカレントフレームの設定（`Display.setCurrent()` メソッドの呼び出し）を行なう前に行なう必要があります。また、一度設定したキーアサインを、その i アプリが起動している間に変更することはできません。

なお、キーアサインの設定内容は高レベル API においても有効です。高レベル API 表示のキーイベントやフォーカス移動、画面スクロール操作は設定されているキーアサインにしたがいます。

<電池パックの満充電までの残り充電時間>

充電中の状態において、満充電までの残りの充電時間を調べることができる、参照のみ可能な属性です。満充電までの残りの充電時間を参照するには、属性 `PhoneSystem.DEV_CHARGING_TIME` を使用します。

満充電までの残りの充電時間は分単位で返されます。充電中でない場合にはこの値は取得できません。

本機能を利用して取得した値は、本機能呼び出した時点の状況を元にした推測値です。従って本機能で得られる値は、あくまで目安として使用するようして下さい。

<電池パックの残り端末利用時間>

電池パックで動作中（充電中でない）の状態において、現在のアプリケーションが継続される前提で携帯電話がどのくらいの時間充電しないで利用可能であるかを調べることができる、参照のみ可能な属性です。携帯電話の利用可能時間を参照するには、属性 `PhoneSystem.DEV_PHONE_REMAINING_TIME` を使用します。

携帯電話の利用可能時間は分単位で返されます。充電中はこの値は取得できません。

本機能を利用して取得した値は、本機能呼び出した時点の状況を元にした推測値です。従って本機能で得られる値は、あくまで目安として使用するようして下さい。

<方位デバイスの基準方向の制御>

電子コンパス機能や歩行方位取得機能において、方位デバイスが計測した携帯電話の向いている方角を i アプリで取得する際の、基準となる方向を制御するための、参照・設定の両方が可能な属性です。方位デバイスは携帯電話本体内の特定の位置に固定されていますが、近年の携帯電話には、横開閉型や 2 軸ヒンジ型など、さまざまなスタイルで持つことを想定したものも多数あります。この機能は、通常の開状態以外の持ち方を意図した i アプリでも、ユーザーにとって自然な向き（視線の方向）を基準に方位が取得できるようにするために設けられました。

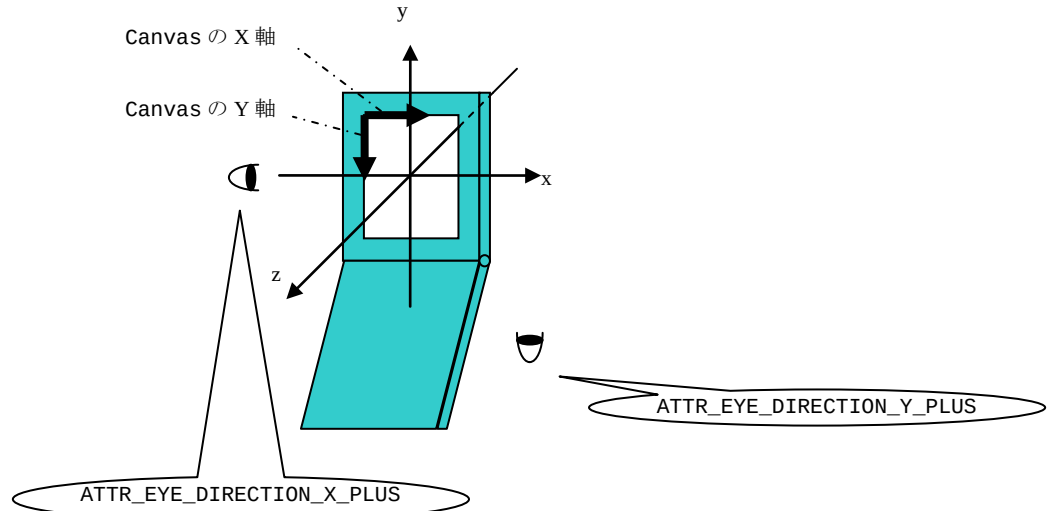
基準とする方向は、`Canvas` 描画面の座標系に沿って定義される携帯電話の軸の方向に対して、ユーザーの視線がどの方向を向いているかで表し、属性 `PhoneSystem.DEV_EYE_DIRECTION` に対して、以下の属性値を設定することで制御します。

- `ATTR_EYE_DIRECTION_Y_PLUS` (y 軸の正の方向)
- `ATTR_EYE_DIRECTION_X_PLUS` (x 軸の正の方向)
- `ATTR_EYE_DIRECTION_Y_MINUS` (y 軸の負の方向)
- `ATTR_EYE_DIRECTION_X_MINUS` (x 軸の負の方向)

Canvas 描画面の座標系、携帯電話の軸の方向およびユーザーの視線の方向の対応は以下のようになります。

携帯電話の x、y、z の各方向：

- x： Canvas の X 軸の正方向に向かう方向
- y： Canvas の Y 軸の負方向に向かう方向
- z： ディスプレイの奥から手前に向かう方向



電子コンパス機能や歩行方位取得機能により取得する方位角は、磁北とこの属性で設定した基準方向が成す角度として表されます。

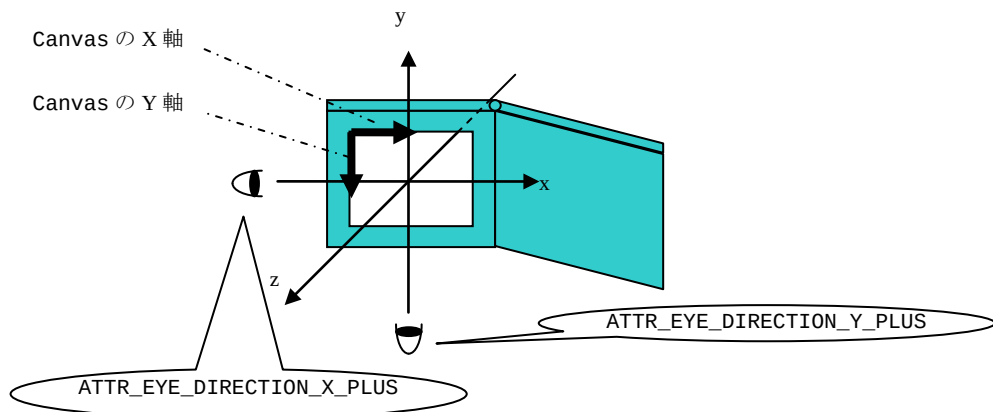
なお、i アプリ起動直後、この属性はストレート型や折りたたみ型の通常開状態の携帯電話をユーザーが普通に持った状態にあたる `PhoneSystem.ATTR_EYE_DIRECTION_Y_PLUS` が設定されています。

注意事項：

- i アプリオプション API である `Canvas2` クラスを利用して座標系を変更した場合、ディスプレイに対する `Canvas` 描画面の座標系の向きが変わるため、本機能の属性値が示す向きも携帯電話の向きに対して以下のように変わります。

携帯電話の x、y、z の各方向：

- x： Canvas の X 軸の正方向に向かう方向
- y： Canvas の Y 軸の負方向に向かう方向
- z： ディスプレイの奥から手前に向かう方向



第 6 章

携帯電話ネイティブの データストレージアクセス

携帯電話ネイティブのデータストレージとは、待ち受け画面に設定する待ち受け画像データや着信メロディ、さまざまな状況に応じて画面に表示される 3D キャラクタデータなど、携帯電話のネイティブアプリケーションが使用するデータを保存する領域を指します。携帯電話ネイティブのデータストレージアクセス機能では、アプリケーションからこれらのデータ領域にアクセスする手段を提供します。現在のプロファイルでは、利用可能なデータストレージはインフォメーションディスプレイ(*1)表示パターン保存領域のみです。

(*1) インフォメーションディスプレイとは、メインディスプレイとは別に携帯電話の背面などに配置される、ユーザーに簡単な情報（現在時刻や、メール着信があったことなど）を提示するためのディスプレイです。

ただし、これらの領域に保存されるデータに対してアプリケーションから制限なくアクセスできることは、ユーザー保護・コンテンツの著作権保護の観点から好ましくありません。

本機能では携帯電話ネイティブのデータストレージの中でも、アプリケーションがアクセスすることを許された特別なエリア（以降、許可エリアと呼びます）にのみ、アプリケーションがアクセスすることを許可しています。許可エリアには、以下のような制限があります。

- ・ ユーザーは、ブラウザなどからのユーザー操作によるデータの保存領域として許可エリアを使用することはできません。ユーザーが使用するエリアとアプリケーションが使用するエリアは明確に分離されており、相互に上書きすることはできないようになっています。
- ・ この機能を使用してアプリケーションが許可エリアにデータを書き込む際、i アプリ実行環境はユーザーに確認を求めるダイアログを表示します。このダイアログでユーザーの許可が得られなかった場合、データの書き込みは行われません。
- ・ 許可エリアに保存されたデータを携帯電話ネイティブのアプリケーションが使用できるようにするには、ユーザーは許可エリア上のデータを使用することを明示的に携帯電話に設定しなければなりません。

<インフォメーションディスプレイ表示パターン保存領域へのアクセス>

インフォメーションディスプレイ表示パターン保存領域に表示パターンを保存するには、`com.docomostar.opt.ui.PhoneResource` クラスの `setInformationDisplay()` メソッドを使用します。許可エリアには表示パターンを複数個保持することができるようになっており、i アプリは何番目の表示パターンを書き換えるかを ID（1 以上の整数値）により指定します。1 つの表示パターンは複数のページから構成させることができ、パターン内でページを順次切り替えて表示させることによりアニメーションのように見せることができます。

このメソッドの引数は以下の通りです。

ターゲット ID (int 型)

ネイティブ保存領域の何番目の表示パターンを書き換えるかを示す ID を、1 以上 11 以下の整数値で指定します。なお、この ID に値 0 を指定すると、点灯パターンをネイティブ保存領域に保存せず、即座にインフォメーションディスプレイ上に表示します。

タイトル (String 型)

表示パターンは、タイトル付きでネイティブ保存領域に保存されます。ユーザーがネイティブメニュー操作でインフォメーションディスプレイの設定を行う際、i アプリが保存した表示パターンはここで指定したタイトルにより識別されます。

有効期限 (long 型)

表示パターンを保存する際は、その表示パターンの有効期限を設定することができます。有効期限を過ぎた表示パターンはインフォメーションディスプレイ上には表示されません。有効期限を設定する場合は、この引数に 1970 年 1 月 1 日午前 0 時からの経過ミリ秒数を指定します。この引数に負の値が設定された場合は、有効期限の指定がないもの（無期限）として取り扱われます。

有効期限は表示パターンをネイティブ保存領域に保存される際に使用される引数であり、ターゲット ID として 0 が指定された場合は無視されます。

繰り返し表示回数 (int 型)

表示パターンを何回繰り返し表示するかを指定します。0 を指定した場合、回数制限がないものとして繰り返し表示を行いません。

表示パターン (int[][]型)

表示パターンを、int 型の二次元配列データとして指定します。配列の第一次元が表示パターンのページ番号に対応し、個々のページ番号に対応するパターンデータを配列の第二次元に設定します。

表示パターンデータの設定イメージ：

```
int[][] patterns = { { 2, 0, 255, ... (略) },           // 1 ページ目のパターンデータ
                     { 0, 0, 255, ... },             // 2 ページ目のパターンデータ
                     { 0, 0, 255, ... },             // 3 ページ目のパターンデータ
                     { 3, 0, 255, ... } };           // 4 ページ目のパターンデータ
```

ページ毎のパターンデータの構成は、実際に携帯電話に搭載される対応デバイスのスペックや機能（パターンをテキストとして指定するかドットパターンとして指定するか、またドットパターンの場合はどのような縦横ドット数で指定するかなど）に強く依存します。このため、パターンデータをどのような構成とするかは各メーカーの設計に委ねられます。実際の機種でパターンデータがどのような構成となっているかは、別途 NTT ドコモより公開されます。

ネイティブ保存領域に保存された表示パターンの情報を取得するには、

PhoneResource.getInformationDisplayData() を使用します。このメソッドに、対象とする表示パターンのターゲット ID (1 以上 11 以下の値) と情報の種別を指定することで、保存されている表示パターンの情報を取得することができます。現在のプロファイルでは、取得可能な情報の種別としてはタイトル (**PhoneResource.INFORMATIONDISPLAY_DATA_TITLE**) のみが規定されています。

第7章 ポインティングデバイス

ポインティングデバイスは、画面上を自由に移動可能なカーソルを操作するための入力デバイスです。ユーザーはポインティングデバイスを操作してカーソルを移動させることで、画面の座標空間上の位置を i アプリに伝えることができます（マウスモード）。また、機種によってはポインティングデバイスのユーザー操作の状態を、向きや傾き量などにより i アプリに通知できるものもあります（ジョイスティックモード）。ポインティングデバイスの機能は `com.docomostar.opt.ui.PointingDevice` クラスで提供されます。なお、本項で解説するポインティングデバイス機能は低レベル API 利用時のみ使用することができます。

ポインティングデバイスは通常は無効化されています。ポインティングデバイスの有効、無効を切り替えるには、`PointingDevice.setEnabled()` メソッドを使用します。

`PointingDevice.isEnabled()` メソッドにより、現在ポインティングデバイスが有効であるか否かを確認することもできます。また、ポインティングデバイスはユーザーの設定により使用が禁止されている場合があります。ユーザー設定によりポインティングデバイスの使用が禁止されている場合は i アプリからもポインティングデバイスを使用することはできませんが、この設定状況は `PointingDevice.isAvailable()` メソッドで確認することができます。

ポインティングデバイスを搭載する機種において、マウスモードとジョイスティックモードのいずれをサポートするか、および初期状態がどちらのモードであるかはメーカーにより異なります。アプリケーションプログラムは、`PointingDevice` クラスの `setMode()` メソッドや `getMode()` メソッドを使用して、モードを設定したり現在のモードの確認を行なうことができます。

以下に、各モードの使用方法について解説します。

<マウスモード (MODE_MOUSE) >

マウスモードでは、アプリケーションプログラムはユーザーがポインティングデバイスを操作したことをポインティングデバイス操作開始イベント (`Display.POINTER_MOVED_EVENT`) の通知により知ることができます。ポインティングデバイスを有効化し、マウスモードに設定した状態でカレントフレームに `Canvas` を設定すると、ユーザーがポインティングデバイスの操作を開始した際にポインティングデバイス操作開始イベントがその `Canvas` の `processEvent()` メソッドに通知されるようになります。このイベントの通知を受けて `PointingDevice.getX()` メソッドおよび `PointingDevice.getY()` メソッドを呼び出すことで、画面上のカーソルの位置を取得することができます。これらのメソッドで取得されるカーソル位置は画面の左上を原点とした絶対座標となります。

マウスモードでは、カーソルの位置を `setPosition()` メソッドで設定したり、カーソルの表示/非表示を `setVisible()` メソッドで切り替えることができます。

注意事項：

- ポインティングデバイスをマウスモードで操作するのと同時にタッチパネルを操作した場合の振る舞いは機種によって異なります。

<ジョイスティックモード (MODE_JOYSTICK) >

ジョイスティックモードでは、アプリケーションプログラムはポインティングデバイスのユーザー操作の状態を、デバイスの押されている向きや傾きの量により知ることができます。マウスモードとは異なり、ユーザー操作の契機をイベントなどで知ることができないため、アプリケーションプログラムはタイマーなどを使用して定期的にデバイスの状態を調べる必要があります。

デバイスの状態を知るためのメソッドとして、**PointingDevice** クラスには以下のメソッドが用意されています。

- **getDirectionX()**

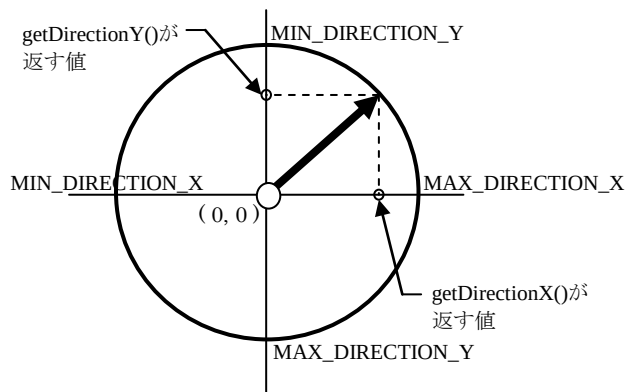
デバイスの現在の操作方向の、X 軸（横方向）の成分を取得します。この値の取り得る範囲は、**PointingDevice** クラスに定義されている定数 **MIN_DIRECTION_X** から **MAX_DIRECTION_X** までとなります。

- **getDirectionY()**

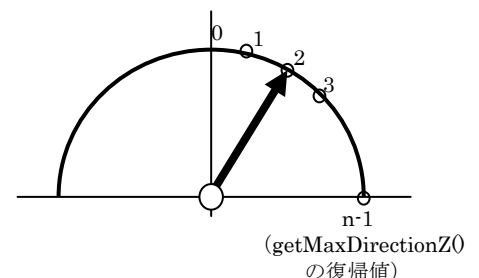
デバイスの現在の操作方向の、Y 軸（縦方向）の成分を取得します。この値の取り得る範囲は、**PointingDevice** クラスに定義されている定数 **MIN_DIRECTION_Y** から **MAX_DIRECTION_Y** までとなります。

- **getDirectionZ()**

デバイスの現在の傾きを取得します。傾きの量を（傾いていない状態含め）**n** 段階に分けて検知できるデバイスでは、この値の取り得る範囲は **0** から **n-1** までとなります。この値の取り得る最大値は **PointingDevice.getMaxDirectionZ()** メソッドによっても取得することができます。



デバイスの向きの表現



デバイスの傾きの表現

注意事項：

- **com.docomostar.opt.ui.Canvas2** クラス (第 5 章参照) で座標系を制御した場合、ポインティングデバイスで扱う座標系に影響を与えるかどうかはメーカーにより異なります。

【Star-1.1】

Star-1.1 プロファイル以降、i アプリからサスペンドメソッドを呼び出してサスペンドした状態でユーザーがポインティングデバイス操作を行うと、i アプリがレジュームすることが規定されました。ポインティングデバイスが無効である状態でユーザーがポインティングデバイス操作を行った場合にレジュームするかどうかは、メーカーにより異なります。

第 8 章

指紋センサー

メーカーによっては、主にセキュリティ認証の用途のために指紋センサーを搭載している場合があります。i アプリオプション API では、この指紋センサーを応用する API として指紋認証機能を規定しています。

指紋認証機能では、携帯電話に搭載された指紋認証デバイスおよび指紋データを使用して、i アプリを使用しているユーザーの本人性確認を行うための機能を提供します。指紋認証機能は、`com.docomostar.opt.device.FingerprintAuthenticator` クラスで提供されます。

指紋認証に対応した携帯電話では、ネイティブの登録機能によりユーザーの指紋データを携帯電話に登録することができます（指紋データを複数登録することもできます）。i アプリの指紋認証機能では、アプリケーションプログラムから指紋認証機能呼び出し、i アプリ利用ユーザーの指紋の入力、および登録済み指紋データとの照合を行います。携帯電話に複数の指紋データが登録されている場合には、その中のどれかに合致しているかを検証することも、特定の指紋データに合致しているかを検証することもできます。

i アプリから指紋認証機能を使用する際には、`FingerprintAuthenticator` クラスの static メソッド `getFingerprintAuthenticator()` を呼び出して `FingerprintAuthenticator` オブジェクトを取得します。このオブジェクトでは以下のメソッドが利用可能です。

- `select()` メソッド

携帯電話に登録されている個々の指紋データには、その携帯電話内で一意となるエントリ ID がシステムにより割り当てられています。登録されている指紋データのうち特定のものと照合を行う場合は、次に述べる `authenticate()` メソッドの引数に登録済み指紋データのエントリ ID を指定します。そのエントリ ID を取得するにはこのメソッドを使用します。

このメソッドを呼び出すと i アプリはサスペンドして、携帯電話ネイティブの指紋データ選択画面が表示されます。ユーザー操作により、登録されている指紋データのうち 1 つが選択されるとレジュームし、アプリケーションプログラムに選択された指紋データに対応するエントリ ID が返されます。

なお、登録済みの全ての指紋データに対して照合を行う場合は、本メソッドを使用する必要はありません。

- `authenticate()` メソッド

指紋認証機能呼び出し、i アプリ利用ユーザーの指紋入力と、入力された指紋と登録済み指紋データとの照合を行います。このメソッドには引数を取らないものと、引数として登録済み指紋データのエントリ ID を指定（1 つまたは複数）するものがあります。入力された指紋が登録済み指紋データ全てのうちいずれかに合致するかどうかを調べるには前者を、特定の指紋データに合致するかどうかを調べるには後者を利用します。

このメソッドを呼び出すと i アプリはサスペンドして、携帯電話ネイティブの指紋入力画面が表示されます。ユーザー操作により指紋の入力および照合が行われるとレジュームし、アプリケーションプログラムに照合結果が返されます。

注意事項：

- 携帯電話に登録済みの指紋データや i アプリ利用ユーザーにより入力された指紋について、形状など指紋そのものを特徴付ける情報を取得することはできません。i アプリからアクセス可能な情報は、登録済み指紋データのエントリ ID、および登録済み指紋データと入力された指紋との照合結果に限定されます。
- 指紋認証では、指紋読み取りデバイスの物理性能や照合アルゴリズムなどの関係により、誤判定が発生する可能性があります。およその目安として、誤判定の発生確率は下記の程度となります。
 - － 他人許容率（他人を本人と誤認識する確率）： 0.01%以下
 - － 本人拒否率（本人を他人と誤認識する確率）： 1%以下

また、誤判定の確率は使用状態やユーザーの年齢などにより変動することがあります。

第 9 章 アプリケーション連携拡張

アプリケーション連携拡張では、i アプリ基本 API のアプリケーション連携機能では規定されていない種類のネイティブアプリケーションと i アプリの間の連携機能を提供します。

現在のプロファイルの i アプリオプション API では、以下の機能が規定されています。

- ・ 映像データ管理機能の呼び出し
- ・ トルカ管理機能の呼び出し
- ・ 住所リンク機能からの i アプリの起動
- ・ スケジューラからの i アプリの起動
- ・ データ BOX 保存画像からの i アプリの起動

注意事項：

- スケジューラからの i アプリの起動およびデータ BOX 保存画像からの i アプリの起動は Star-1.3 プロファイルで新設されました。

9.1 映像データ管理機能の呼び出し

ネイティブの映像データ管理機能のうち、映像データの新規登録は i アプリ基本 API として提供されています。これに対し、ネイティブ保存領域に保存されている映像データへのアクセスは、現在のプロファイルでは i アプリオプション API の位置づけとなっています。本項では、**com.docomostar.system.MovieStore** クラスを使用して、ネイティブ保存領域に保存されている映像データを利用する方法について解説します。

なお、これらの機能を使用する i アプリは、ADF の **AccessUserInfo** キーにてこれらの機能を使用することを宣言（値"yes"を設定）する必要があります。また i アプリは、自分自身が保存したのではなく、かつ再配布が不可と設定されている映像データを取り扱うことはできません。

(A) 映像データの選択読み込み

映像データの選択読み込みとは、アプリケーションプログラムからの映像データ要求（メソッド呼び出し）に対し、その都度画面に表示される映像一覧などからユーザーが選択操作した映像データを与える機能です。映像データの選択読み込みを行うには、**MovieStore** クラスの static メソッド **selectEntry()** を使用します。

映像データの選択読み込みを行うと、携帯電話は以下のような動作を行います。

- (1) アプリケーションプログラムから **MovieStore.selectEntry()** メソッドを呼び出します。このメソッドが呼び出されると i アプリはサスペンドします。

- (2) 映像データの選択を行うためのネイティブのユーザーインターフェース画面が起動します。この画面を使用して、ユーザーは i アプリにアクセスを行わせる映像データを選択します。また、操作をキャンセルして処理を中止することもできます。
- (3) ユーザーが選択する映像データを決定すると、i アプリはレジュームして `selectEntry()` メソッドはアプリケーションプログラムに復帰します。その際このメソッドは、選択された映像データに対応する `MovieStore` オブジェクトを返します。

ユーザーが映像データの選択操作をキャンセルしても例外は発生せず、このメソッドは `null` を返します。

`selectEntry()` メソッドが返した `MovieStore` オブジェクトからは、選択された映像のバイトイメージや `MediaMovie` オブジェクトを取得することができます。また、この `MovieStore` オブジェクトからは映像データのエン트리 ID を取得することもでき、後述する映像データの ID 指定読み込みにおいて、ユーザー操作なしに再度その映像データを読み込むために使用することができます。

(B) 映像データの ID 指定読み込み

映像データのエン트리 ID は、映像がネイティブの保存領域に保存される際にシステムにより割り当てられる一意な整数値であり、その映像データが削除または別のデータにより上書きされるまで変わることはありません。i アプリは映像データの新規登録または選択読み込みで一度有効なエン트리 ID を取得すると、それ以降はユーザー操作を経ることなくその映像データにアクセスすることができます。映像データの ID 指定読み込みを行うには、`MovieStore` クラスの static メソッド `getEntry()` を使用します。

`getEntry()` メソッドの引数に映像データのエン트리 ID を指定して呼び出すと、その映像データに対応する `MovieStore` オブジェクトが返されます。この `MovieStore` オブジェクトからは、映像データの選択読み込み時と同様に映像のバイトイメージや `MediaMovie` オブジェクトを取得することができます。

【Star-2.0】

Star-2.0 プロファイル以降、映像データの選択読み込み機能、映像データの ID 指定読み込み機能は i アプリ基本 API となりました。

9.2 トルカ管理機能の呼び出し

トルカ管理機能の呼び出しでは、i アプリから携帯電話のネイティブのトルカ管理機能呼び出し、トルカデータを新規登録したり、ネイティブに保存されているトルカデータを読み込んだり（選択読み込みおよび ID 指定読み込み）、ネイティブに保存されているトルカデータのうち特定の条件に合致するものを検索取得するための API がオプション API として規定されています。

本項で解説する機能を使用する i アプリは、ADF の `AccessUserInfo` キーにてこの機能を使用することを宣言（値 `"yes"` を設定）する必要があります。

なお本書では、トルカ仕様におけるトルカを「トルカスニップ」、トルカ（詳細）を「トルカデータ本体」と記載しています。

9.2.1 Toruca クラス

i アプリでは、トルカデータは `com.docomostar.system.Toruca` クラスで表現されます。i アプリがトルカデータをネイティブ保存領域に保存するには、コンストラクタを用いて

Toruca オブジェクトを生成し、その **Toruca** オブジェクトに保存したいトルカデータの内容を設定する必要があります。

Toruca オブジェクトにトルカデータの内容を設定するには、以下の2通りの方法があります。

- **Toruca** クラスのコンストラクタに、トルカデータのコンテンツ仕様に従ったコンテンツデータを **byte** 配列の形式で指定します。この場合、**Toruca** オブジェクトの各データ項目は、指定されたコンテンツデータに含まれる各データ項目の値で初期化されます。
- **Toruca** オブジェクトを生成し、データ項目毎に用意されたアクセッサメソッドを使用して各データ項目の設定を行います。

トルカデータのコンテンツデータ項目やコンテンツフォーマット、コンテンツフォーマットバージョンなど、トルカに関するコンテンツ仕様は別途 NTT ドコモより公開されます。

また、**Toruca** クラスに定義された **setProperty()** メソッドを使用して、i アプリから内蔵メモリに新規登録されたトルカデータ（再配布不可と設定されたトルカデータ）をユーザーが SD カードなどの外部メモリに出力する際の、コンテンツ保護レベルを制御することができます。

通常、再配布不可として登録されたトルカデータを外部メモリに移すと、そのトルカデータはデータ保存時と同じ機種、かつ保存時と同じ UIM が挿入されている携帯電話上でなければ復号することはできません。これについて、保存するトルカデータの **Toruca** オブジェクトに対し、**setProperty()** メソッドを

- 第1引数（プロパティ名）： “X-Dcm-Move”
- 第2引数（プロパティ値）： “0001”

を指定して呼び出しておくことで、そのトルカデータについては保存時と同じ UIM が挿入された携帯電話上であれば、機種を問わずに復号できるようになります。

このプロパティ情報は内蔵メモリ上でも保持されており、i アプリが **TorucaStore** クラスを使用して再度内蔵メモリから読み込んだ際にも有効なまま引き継がれます。

9.2.2 トルカデータの新規登録

前項に従って作成した **Toruca** オブジェクトをネイティブ保存領域に新規登録するには、**com.docomostar.system.TorucaStore** クラスの **static** メソッド **addEntry()** を使用します。このメソッドの引数に **Toruca** オブジェクトを指定することで、オブジェクトの設定内容に従ったトルカデータが新規登録されます。

トルカデータの新規登録を行うと、携帯電話は以下のような動作を行います。

- (1) アプリケーションプログラムから **TorucaStore.addEntry()** メソッドを呼び出します。このメソッドが呼び出されると i アプリはサスペンドします。
- (2) トルカデータの登録を行うためのネイティブのユーザーインターフェース画面が起動します。この画面では、ユーザーは i アプリからのトルカデータの新規登録を許可、キャンセルすることができます。また、i アプリがどのようなトルカデータを登録しようとしているかを確認することもできます。
- (3) ユーザーがトルカデータの新規登録を許可するとトルカデータの新規登録が行われ、i アプリはレジュームして **addEntry()** メソッドはアプリケーションプログラムに復帰します。その際このメソッドは、新規登録されたトルカデータにその携帯電話内で 1 対 1 に対応するエン트리 ID を返します。

ユーザーがトルカデータの新規登録をキャンセルしても例外は発生せず、このメソッドは -1 を返します。

保存可能なトルカデータのサイズには上限があります。**TorucaStore** クラスには、**Toruca** オブジェクトとして作成しているトルカデータにあと何バイトのデータが設定できるかを確認するための **getRemainingBytes()** メソッドが用意されています。

9.2.3 トルカデータの選択読み込み

トルカデータの選択読み込みとは、アプリケーションプログラムからのトルカデータ情報要求（メソッド呼び出し）に対し、その都度画面に表示されるトルカデータ一覧からユーザーが選択操作したトルカデータの情報を与える機能です。トルカデータの選択読み込みを行うには、**TorucaStore** クラスの static メソッド **selectEntry()** を使用します。

トルカデータの選択読み込みを行うと、携帯電話は以下のような動作を行います。

- (1) アプリケーションプログラムから **TorucaStore.selectEntry()** メソッドを呼び出します。このメソッドが呼び出されると i アプリはサスペンドします。
- (2) トルカデータの選択を行うためのネイティブのユーザーインターフェース画面が起動します。この画面を使用して、ユーザーは i アプリにアクセスを行わせるトルカデータを選択します。また、操作をキャンセルして処理を中止することもできます。
- (3) ユーザーが選択するトルカデータを決定すると、i アプリはレジュームして **selectEntry()** メソッドはアプリケーションプログラムに復帰します。その際このメソッドは、選択されたトルカデータに対応する **TorucaStore** オブジェクトを返します。

ユーザーがトルカデータの選択操作をキャンセルしても例外は発生せず、このメソッドは **null** を返します。

selectEntry() メソッドが返した **TorucaStore** オブジェクトからは、選択されたトルカデータの内容が格納されている **Toruca** オブジェクトを取得することができます。

また、この **TorucaStore** オブジェクトからはトルカデータのエントリ ID を取得することもでき、後述するトルカデータの ID 指定読み込みにおいて、ユーザー操作なしに再度そのトルカデータを読み込むために使用することができます。

注意事項：

- トルカデータには、テキストデータだけでなく画像コンテンツなどのデータが含まれている場合があります。他者（他の i アプリやネイティブアプリケーション）が保存した再配布不可能なトルカデータや、再配布不可能な画像コンテンツなどを含むトルカデータは本機能で選択することはできません。

9.2.4 トルカデータの ID 指定読み込み

トルカデータのエントリ ID は、トルカデータの新規登録時にシステムにより割り当てられる一意な整数値であり、そのトルカデータが削除されるまで変わることはありません。i アプリは以下によりトルカデータのエントリ ID を取得することができます。

- ・ トルカデータ新規登録 API (**TorucaStore.addEntry()** メソッド) 呼び出しにおける返り値
- ・ トルカデータ選択読み込み API (**TorucaStore.selectEntry()** メソッド) で取得した **TorucaStore** オブジェクトに対する、**getId()** メソッド呼び出しにおける返り値
- ・ トルカデータの検索取得 API (**findByHostAndIpid()** メソッド) で取得した、検索結果のエントリ ID 配列

i アプリはこれらの方法により一度有効なエントリ ID を取得すると、それ以降はユーザー操作を経ることなくそのトルカデータにアクセスすることができます。トルカデータの ID 指定読み込みを行うには、**TorucaStore** クラスの static メソッド **getEntry()** を使用します。

getEntry() メソッドの引数にトルカデータのエントリ ID を指定して呼び出すと、そのエントリ ID に対応付けられたトルカデータを示す **TorucaStore** オブジェクトが返されます。

注意事項：

- トルカデータには、テキストデータだけでなく画像コンテンツなどのデータが含まれている場合があります。他者（他の i アプリやネイティブアプリケーション）が保存した再配布不可能なトルカデータや、再配布不可能な画像コンテンツなどを含むトルカデータを、本機能で読み込もうとすると例外がスローされます。

9.2.5 トルカデータの検索取得

トルカデータの検索取得とは、ネイティブに保存されているトルカデータのうち特定の条件に合致したものの一覧をユーザーに提示して、その中からユーザーが選択操作（複数選択可能）したトルカデータの情報を i アプリに与える機能です。機能としては 9.2.3 項に記載したトルカデータの選択読み込みに似通っていますが、ユーザーに選択させたいトルカデータの範囲に条件をつけられる（コンテンツプロバイダにとって既知の条件で絞り込むことができる）場合には、こちらの機能を使用する方がユーザーの操作性が向上します。

現在のプロファイルでは、トルカデータの検索取得で設定できる条件は、トルカデータ本体の取得先 URL とトルカデータの IP-ID です（これらはいずれもトルカデータを構成するデータ項目であり、詳細仕様は別途 NTT ドコモより公開されます）。指定された取得先 URL と IP-ID の条件に合致したトルカデータの一覧がユーザーに提示され、その中からユーザーが選択したものを i アプリから参照することができます。

トルカデータの検索取得を行うには、TorucaStore クラスの static メソッド `findByHostAndIpid()` を使用します。このメソッドの第 1 引数には検索対象としたいトルカデータの取得先 URL のホスト名（FQDN）を、第 2 引数には検索対象とする IP-ID の値を、それぞれ文字列として指定します。これにより、

- ・ トルカデータに設定された取得先 URL のホストが第 1 引数に指定されたホストと完全に一致している。

例： `findByHostAndIpid()` の第 1 引数にホスト名として `"host.nttdocomo.co.jp"` を指定した場合、トルカデータに設定された取得先 URL として以下のようなものが該当します。

```
http://host.nttdocomo.co.jp/data.trc
```

```
http://host.nttdocomo.co.jp:8080/subfolder/data.trc
```

```
https://host.nttdocomo.co.jp/data.trc
```

- ・ トルカデータに設定された IP-ID の値と第 2 引数に指定された IP-ID の値について、少なくとも 8 バイトが前方一致している（`findByHostAndIpid()` の第 2 引数に指定した IP-ID の先頭 8 バイトと、トルカデータに設定された IP-ID の先頭 8 バイトが一致している）。

の両方を満たしたトルカデータが、検索条件に合致するものとして取り扱われます。特に 2 つ目の条件によって、以下のような点に注意する必要があります。

- ・ `findByHostAndIpid()` の第 2 引数に指定した IP-ID が 8 バイトより短い場合、その検索結果は常に空になります。
- ・ IP-ID 値として 8 バイトより短い文字列が格納されているトルカデータは、`findByHostAndIpid()` メソッドの検索対象にはなりません。

トルカデータの検索取得を行うと、携帯電話は以下のような動作を行います。

- (1) アプリケーションプログラムから `TorucaStore.findByHostAndIpid()` メソッドを呼び出します。このメソッドが呼び出されると i アプリはサスペンドします。
- (2) トルカデータの検索が行われ、条件に合致したトルカデータの中から選択を行うためのネイティブのユーザーインタフェース画面が起動します。この画面を使用して、ユーザーは i アプリにアクセスを行わせるトルカデータを選択（複数選択可能）します。また、操作をキャンセルして処理を中止することもできます。

- (3) ユーザーが選択するトルカデータを決定すると、i アプリはレジュームして `findByHostAndIpid()` メソッドはアプリケーションプログラムに復帰します。その際このメソッドは、選択されたトルカデータに対応するエントリ ID を、`int` 型の配列として返します。

ユーザーがトルカデータの検索取得をキャンセルしたり、条件に一致するトルカデータが 1 つも存在しなかった場合も例外は発生せず、このメソッドは `null` を返します。

`findByHostAndIpid()` メソッドが返すエントリ ID を使用して ID 指定読み込みを行うことで、そのトルカデータに対応する `Toruca` オブジェクトを取得することができます。

注意事項：

- トルカデータには、テキストデータだけでなく画像コンテンツなどのデータが含まれている場合があります。他者（他の i アプリやネイティブアプリケーション）が保存した再配布不可能なトルカデータや、再配布不可能な画像コンテンツなどを含むトルカデータは本機能での検索の対象外となります。
- 本機能は、i アプリからのトルカ管理機能呼び出しをサポートしている機種の中でもオプションです。

9.3 住所リンク機能からの i アプリの起動

住所リンク機能とは、ブラウザ（HTML コンテンツ）やトルカコンテンツなどで位置情報を取り扱うための機能です。住所リンク機能に対応した携帯電話では、HTML などのアンカータグの `href` 属性に `pos` スキーム（`pos` スキームは住所リンク機能のために設けられました）の URL を記述することによってユーザーがそのアンカーから住所リンク機能のためのメニュー（ロケーションサービスメニューと呼ばれることもあります）を呼び出せるようになります。`pos` スキームの URL には緯度や経度、住所などの位置情報を記載できるようになっており、住所リンク機能のメニューを通じて起動された i アプリがそれらの情報を活用することで、多彩な位置情報サービスを提供することができるようになっています。

i アプリは、ADF に `GetLocationParam` キーを記述することで、住所リンク機能から起動されることを意図して作成されたものであることを宣言することができます。

`GetLocationParam` キーの書式は以下の通りです。

`GetLocationParam = yes`

この宣言が行われた i アプリは、ユーザーが住所リンク機能のメニューを開いた際に、そこから起動可能な i アプリとしてリストアップされます。また住所リンク機能から起動された i アプリは、起動元となったアンカータグの `pos` スキーム URL の記述に応じて、以下のようなパラメータを i アプリ実行環境から受け取ることができます。これらの取得には、`StarApplicationManager.getParameter()` メソッドを使用します。

パラメータキー	キー値の意味	キー値の詳細
latitude	緯度	緯度を表すテキスト情報。 <code>pos</code> スキーム URL の <code>lat</code> パラメータ(*1)に記述された値を得ることができます。 例： <code>N12.34.56.789</code> （北緯 12 度 34 分 56.789 秒）
longitude	経度	経度を表すテキスト情報。 <code>pos</code> スキーム URL の <code>lon</code> パラメータ(*1)に記述された値を得ることができます。 例： <code>E12.34.56.789</code> （東経 12 度 34 分 56.789 秒）
datum	測地系	測位時に適用された測地系を示す値。 <code>pos</code> スキーム URL の <code>geo</code> パラメータ(*1)に記述された値に対応し、以下の値を得ることができます。 WGS84 測地系の場合： <code>"0"</code> 東京測地系の場合： <code>"1"</code>
address	住所	住所を表すテキスト情報。 <code>pos</code> スキーム URL の <code>address</code> パラメータ(*1)に記述された値を得ることができます。

title	タイトル	タイトルを表すテキスト情報。pos スキーム URL の title パラメータ(*2)に記述された値を得ることができます。
opurl	オプション情報 URL	オプション情報の取得先 URL を表すテキスト情報。pos スキーム URL の opurl パラメータ(*2)に記述された値を得ることができます。このパラメータは、該当の位置情報を補足する情報の URL（例えば風景画像などの URL）など、任意の用途で使うことができます。

(*1) pos スキームにおいては、address パラメータによる住所指定、もしくは lat/lon/geo の各パラメータによる位置指定のうち、少なくともいずれか一方は必ず指定する必要があります。この規定の範囲内で省略されているパラメータは、値を取得することはできません。

(*2) pos スキームにおいては、これらのパラメータは省略可能です。省略されている場合は、値を取得することはできません。

注意事項：

- アンカータグへの pos スキーム URL の記述方法の詳細は、別途 NTT ドコモより公開されます。
- 起動先 i アプリはフルのみアプリか、フル+ミニアプリでなければなりません。また、起動先 i アプリが、フル+ミニアプリである場合は、フルアプリが起動されます。
- 起動元となったアンカータグの pos スキーム URL に指定されていないデータ項目を `StarApplicationManager.getParameter()` メソッドで取得しようとした場合は、値 null が返ります。
- 住所リンク機能から起動された場合の i アプリの起動種別 (`StarApplication.started()` メソッドの引数に設定される起動種別) は、`StarApplication.LAUNCHED_FROM_LOCATION_INFO` となります。
- 住所リンク機能から起動された i アプリの動作が終了 (`StarApplication.terminate()` により終了) すると、起動元のアプリケーション (ブラウザなど) の状態に復帰します。ただし、i アプリが強制終了された場合など、上記以外の方法で i アプリが終了した場合は、i アプリ終了後にどのような状態に移るかはメーカーにより異なります。
- 住所リンク機能に対応した HTML コンテンツの作成方法の詳細や注意事項は、別途 NTT ドコモより公開されます。
- ADF に `GetLocationParam` キーが設定されている i アプリを更新する際、`GetLocationParam` キーを削除することはできません。i アプリの更新時に `GetLocationParam` キーが削除されている場合、携帯電話は更新処理を中止します。

9.4 スケジューラからの i アプリの起動

携帯電話の中には、ユーザー毎にパーソナライズされた情報を配信してユーザーの行動支援を行う「i コンシェル」と呼ばれるサービスに対応したものがあります。そのような携帯電話では、i コンシェルによりスケジュールデータとして取得した情報を表示するための高機能なスケジューラを搭載しています。このスケジューラを搭載した携帯電話には、取得したスケジュールデータに i アプリ連携起動用のリンクが登録されている場合に、リンクを操作することで指定された i アプリを連携起動できるものがあります。

i アプリは、ADF に `LaunchBySchedule` キーを記述することで、スケジューラから起動されることを意図して作成されたものであることを宣言することができます。

`LaunchBySchedule` キーの書式は以下の通りです。

`LaunchBySchedule = yes`

スケジューラからは、`LaunchBySchedule` キーが記述された i アプリを、その i アプリの ADF 取得元 URL を指定することで起動することができます。

また、「地図プラットフォームに設定された i アプリ」や「番組表 i アプリとして設定された i アプリ」といった指定により、その携帯電話上で地図プラットフォームや番組表 i アプリとして設定されている i アプリを起動することができます。ただしこの場合も、対象となる i アプリ（地図プラットフォームや番組表 i アプリに設定されている i アプリ）には **LaunchBySchedule** キーが記述されている必要があります。

i アプリの連携起動を行うためのスケジュールデータでは、データの中に以下のような行を指定します。

```
X-DCM-LAUNCHIAPP;CHARSET=SHIFT_JIS;ENCODING=BASE64:
TEIOS1NUUklORz2DioOTg06Vto6al/EsCkFERIVSTD1odHRwOi8vYWZhLmNvLmpwL2JiYi5qYW0sTkFNRTk9
VkFMVUUXLE5BTUUyPVZBTfVFMgo=
```

※上記の「BASE64:」以降の文字列は BASE64 エンコードされたデータとなります。上記例では以下のデータが BASE64 エンコードされ、「BASE64:」以降に付与されています。

LINKSTRING=リンク文字列,ADFURL=http://aaa.co.jp/bbb.jam,NAME1=VALUE1,NAME2=VALUE2

LINKSTRING には、スケジューラで表示する際のリンク文字列を指定します。

ADFURL には、連携起動する i アプリの ADF 取得元 URL を指定します。ここに **"MapApp"** を指定することで地図プラットフォームに設定されている i アプリを連携起動することができます。また、**"ProgramGuideApp"** を指定すると番組表 i アプリに設定されている i アプリを連携起動します。

NAME1 以降は、連携起動時に i アプリに渡すパラメータを指定します。NAME1=VALUE1 といったように、**"="** の左側にパラメータの名前を指定し、右側にパラメータの値を指定します（この場合は、NAME1 というパラメータ名で VALUE1 という値がパラメータとして渡されます）。この組み合わせを 1 個として、最大で 16 個のパラメータを設定できます。パラメータ名とパラメータ値（**"="** を除くパラメータ文字列）は、合計で 20480 バイトまで設定できます。パラメータを取得するには、**StarApplicationManager.getParameter()** メソッドを使用します。

なお、i コンシェルでスケジュールデータを配信する際のデータフォーマットの詳細は、別途 NTT ドコモより公開されます。

注意事項：

- 携帯電話ユーザーは、携帯電話の設定によりスケジューラからの i アプリ起動を禁止することができます。またスケジューラから i アプリを起動する前に、i アプリ実行環境はユーザーに i アプリを起動してよいか確認を行います。これらの機構によりユーザーの許可が得られなかった場合、その i アプリの起動は行われません。
- 起動先 i アプリは、フルのみアプリか、フル+ミニアプリでなければなりません。また、起動先 i アプリが、フル+ミニアプリである場合は、フルアプリが起動されます。
- 連携起動しようとしたアプリが存在しない場合、あるいは番組表 i アプリを連携起動しようとして番組表 i アプリが設定されていない場合は、i アプリ実行環境はユーザーにその旨を表示した後、スケジューラ画面に復帰します。また、地図プラットフォームを連携起動しようとしてそれが設定されていない場合は、その旨を表示した後、地図プラットフォームの設定画面に移行します。
- スケジューラから起動された場合の i アプリの起動種別（**StarApplication.started()** メソッドの引数に設定される起動種別）は、**StarApplication.LAUNCHED_FROM_SCHEDULER** となります。
- スケジューラから起動された i アプリの動作が終了（**StarApplication.terminate()** により終了）すると、スケジューラ画面に復帰します。ただし、i アプリが強制終了された場合など、上記以外の方法で i アプリが終了した場合は、i アプリ終了後にどのような状態に移るかはメーカーにより異なります。

9.5 データ BOX 保存画像からの i アプリの起動

データ BOX 内に保存された画像を i アプリで利用するには通常、基本 API である `com.docomostar.system.ImageStore` クラスを利用します。この API では、アプリケーションプログラムが `selectEntry()` メソッドを実行することでユーザーに画像を選択させたり、`selectEntry()` メソッドの結果得られたエントリ ID を用いて画像データを取得することができます。

このような方法とは別に、本項で解説するオプション機能を使用することでデータ BOX 内に保存された画像から画像を取り扱うアプリとしてその携帯電話に登録されている i アプリを起動して、起動された i アプリにその画像を処理させることができます。

データ BOX 内に保存された画像から i アプリを起動するには、ユーザーが事前に画像からの起動用 i アプリとして特定の i アプリを端末に設定する必要があります。この i アプリは、ADF に `LaunchByDataBox` キーを記述することで、データ BOX 保存画像から起動されることを意図した i アプリであること宣言する必要があります。`LaunchByDataBox` キーの書式は以下の通りです。

`LaunchByDataBox = picture`

画像からの起動用 i アプリは、端末で 1 つだけ設定することができます。画像からの起動用 i アプリが設定された状態で別の i アプリに対してこの設定を行うと、元の i アプリに対する設定は解除されます。

画像からの起動用 i アプリに設定された i アプリは、データ BOX のマイピクチャ内で画像を選択した状態でメニュー操作を行うことにより起動します。起動元の画像のエントリー ID は、`StarApplicationManager.getParameter()` の引数に `"entry-id"` を指定することで取得できます。取得したエントリー ID を `ImageStore.getEntry()` の引数に指定することで、起動元の画像データを i アプリ側で取得することができます。

注意事項：

- 画像からの起動用 i アプリを起動可能な画像は、その i アプリから `ImageStore` クラスでアクセスできる画像に限られます。詳細は「i アプリコンテンツ開発ガイド 詳細編」11.10 項を参照してください。
- 起動先 i アプリは、フルのみアプリか、フル+ミニアプリでなければなりません。また、起動先 i アプリが、フル+ミニアプリである場合は、フルアプリが起動されます。
- データ BOX 保存画像から起動された場合の i アプリの起動種別 (`StarApplication.started()` メソッドの引数に設定される起動種別) は、`StarApplication.LAUNCHED_FROM_IMAGE` となります。
- ADF に `LaunchByDataBox` キーが設定されている i アプリを更新する際、`LaunchByDataBox` キーを削除することはできません。i アプリの更新時に `LaunchByDataBox` キーが削除されている場合、携帯電話は更新処理を中止します。

第 10 章

FeliCa チップ

FeliCa 技術方式は非接触 IC カード技術方式の 1 つであり、FeliCa チップを搭載した i アプリ対応携帯電話では i アプリと FeliCa チップの間でデータの送受（データの書き込み・呼び出し）が行えるとともに、そのデータを FeliCa チップと外部機器との間で送受することができます。つまり i アプリと外部機器の間で FeliCa チップのメモリを共有させ、相互にデータのやり取りを行うことができるようになります。

i アプリと外部機器を連携させるためのインタフェースとしては、赤外線ポートを利用した IrOBEX 外部接続機能があります。以下に両者の特長を記載します。

FeliCa チップ：

携帯電話上に搭載された FeliCa チップは、i アプリが動作していない状態であってもチップと外部機器の間でのデータ送受が可能です。携帯電話を取り出した後、i アプリ起動などの余分な操作を行うことなく外部機器にかざすだけ というような、ユーザーにとって手軽で分かりやすいサービスインタフェースを構築することができます。個々人が持ち歩くという携帯電話の特性を利用し、ID カードのような使い方をすることもできます。また、金銭等のデータを扱うことを想定して開発されている FeliCa チップは、セキュリティを保持するための機構も取り入れられています。1 つの i アプリが FeliCa チップ上で取り扱うことのできるデータサイズは数 10～数 100 バイト程度です。

IrOBEX 外部接続：

IrOBEX 外部接続は、i アプリから通信路を開通することで使用可能となります。従ってユーザーは、i アプリを起動させた上で赤外線ポート同士を対向させ、i アプリに通信実行を指示する必要があります。このため一般的には、ユーザーがサービスを受ける際の手順は FeliCa チップと比較すると煩雑になります。1 回の通信で送受信可能なデータのサイズは、画像などのマルチメディアデータを扱える程の容量（数 10K バイト以上）を持ちます。

以降の説明では外部機器のことを外部リーダー・ライターと呼び、外部 R/W と略記します。

10.1 i モード携帯電話に搭載される FeliCa チップの概要

i モード携帯電話に搭載される FeliCa チップのメモリには、共通領域、フリー領域と呼ばれる 2 つの領域がプリセットされています。外部 R/W からは、各々の領域は独立した別のカード（チップ）として見えます（つまり外部 R/W から見て、携帯電話内には 2 枚の論理的なカードが存在することになります）。

< 共通領域 >

共通領域を使用する場合は、アプリケーションに割り当てるメモリ（エリアと呼びます）のサイズやメモリのアクセス手段、メモリの機能を自由に設定できます。共通領域は複数のアプリケーションを 1 枚の論理的なカードに搭載する目的でフェリカネットワークス株式会社によって管理運営されており、共通領域上にエリアの割り当てを受けるにはフェリカネットワークス株式会社への登録申請が必要となります。共通領域の利用はフェリカネットワークス株式会社からエリアの割り当てを受け、i モードサーバーに登録された i アプリにのみ許可されます。許可を受けていない i アプリが共通領域を使用することはできません。

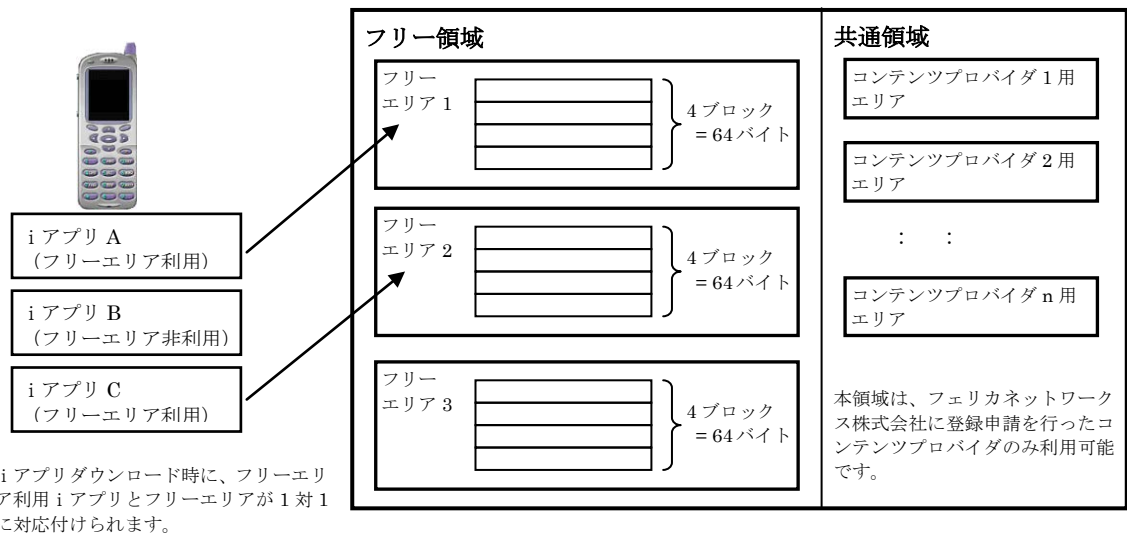
ん（FeliCa チップ上の共通領域へのエリア登録は i アプリからの API 呼び出しにより行われますが、許可を受けていない i アプリはこの API を呼び出すことはできません）。

本書では、共通領域の利用については記載しません。

<フリー領域>

フリー領域とは、1つの i アプリが使用可能なメモリサイズやセキュリティ設計に制限がある一方、i モードサーバーに登録されていない一般の i アプリでも FeliCa チップの機能を一定の範囲で利用できることを目的として設けられた領域です。フリー領域には携帯電話出荷時にあらかじめ定められたサイズのエリア（以降フリーエリアと呼びます）が3つ設けられており、1つの携帯電話内で最大3つまでの i アプリがフリーエリアを使用することができます。1つの i アプリで複数のフリーエリアを使用したり、複数の i アプリで1つのフリーエリアを共有することはできません。またフリーエリアの数は携帯電話内で固定であり、後から追加や削除を行うことはできません。

FeliCa チップでは16バイトを1ブロックとして、ブロック単位でデータの書き込みや読み出しを行います。1つのフリーエリアには、アプリケーションプログラムが利用できるメモリとして4ブロック（64バイト）が割り当てられています。i アプリがどのフリーエリアを使用するかはその i アプリのダウンロード時にシステムにより自動的に決定されます。i アプリに対応付けられたフリーエリアには、後述する API を使用して、割り当てられているメモリ量の範囲で任意のバイトデータを格納することができます。



10.2 フリーエリアアクセス API

FeliCa チップにアクセスするための機能は、`com.docomostar.device.felica` パッケージで提供されます。このパッケージにはフリーエリアアクセスおよび共通領域アクセスのための API が含まれています。

フリーエリアを使用するための API は以下の通りです。これら以外のクラスやメソッドは、共通領域の使用が許可された i アプリでのみ使用することができ、一般の i アプリでは使用することはできません。

クラス名	メソッド名	機能概要
Felica		FeliCa チップを表現するクラス。
	<code>static void open()</code>	FeliCa チップをオープンする。
	<code>static FreeArea getFreeArea()</code>	オープンされた FeliCa チップからフリーエリアオブジェクトを取得する。
	<code>static void close()</code>	FeliCa チップをクローズする。
FreeArea		フリーエリアを表現するクラス。
	<code>byte[] read()</code>	全てのブロックのデータを読み込む。
	<code>byte[] read(int[] index)</code>	指定したブロックのデータを読み込む。
	<code>void write(byte[] data)</code>	全てのブロックにデータを書き込む。
	<code>void write(int[] index, byte[] data)</code>	指定したブロックにデータを書き込む。
	<code>byte[] getIdm()</code>	フリー領域の IDm を取得する。
	<code>void setReset(boolean flag)</code>	フリーエリアアクセス後に、FeliCa チップをリセットするかどうかを設定する。
FelicaException		FeliCa チップ自身やその操作に起因するエラーを通知する例外クラス。
	<code>int getID()</code>	エラー種別を返す。
	<code>int getType()</code>	エラー内容を返す。
PortBusyException		i アプリと外部 R/W の双方から同時にアクセスが発生し、FeliCa チップがビジー状態であることを示す例外クラス。 この例外クラスには、独自のメソッドはありません。

i アプリからフリーエリアを利用する際の、処理の流れは以下の通りです。

- 1) FeliCa チップをオープンするために、static メソッド `Felica.open()` を呼び出します。
- 2) オープンされた FeliCa チップから、static メソッド `Felica.getFreeArea()` メソッドを使用してフリーエリアオブジェクトを取得します。
- 3) 2) で取得したフリーエリアオブジェクトを使用して、ブロックデータの読み込み、書き込みを行います。データの読み込みには `read()` メソッドを、書き込みには `write()` メソッドを使用します。
`read()` メソッド、`write()` メソッドともに、1 回の呼び出しでフリーエリア内の 4 ブロック全てにアクセスすることも、引数のインデックスで指定された一部のブロックのみにアクセスすることもできます。いずれの場合にも FeliCa チップのデータ操作にはトランザクション性があり、1 回のメソッドコール内でのデータブロックアクセスは、全て成功するか全て拒否されるかのいずれかとなります。
- 4) ブロックデータのアクセス終了後、`Felica.close()` メソッドを呼び出して FeliCa チップをクローズします。

以下に、フリーエリアに対して読み込みと書き込みを行う例を示します。

```
try {
    /** 準備処理 */
    Felica.open();                // FeliCa チップをオープンする。
    FreeArea fa = Felica.getFreeArea(); // フリーエリアオブジェクトを取得する。
}
```

```

/** FeliCa チップ上のデータの読み込み（全ブロックの一括読み込み） **/
byte[] readData = fa.read(); // 読み込みを実行する。
... // 読み込んだデータ（4 ブロック=64 バイト）を使用して処理を行う。

/** FeliCa チップへのデータの書き込み（先頭ブロックのみのインデックス指定書き込み） **/
int[] index = new int[1]; // インデックス配列を作成する（1 ブロックアクセスのため要素数 1）。
index[0] = 0; // 先頭ブロックであることを示すインデックス値 0 を設定。
byte[] writeData = {0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, // 書き込むデータを準備する。
                    0x8, 0x9, 0xa, 0xb, 0xc, 0xd, 0xe, 0xf };
fa.write(index, writeData); // 書き込みを実行する。

/** 終了処理 **/
Felica.close(); // FeliCa チップをクローズする。
} catch (FelicaException fe) {
    ... // FeliCa チップアクセスの例外に対する例外処理
} catch (IOException ioe) {
    ... // 入出力例外に対する例外処理
}

```

注意事項：

- 外部 R/W が FeliCa チップにアクセスしているときに **Felica.open()** メソッドを呼び出すと **FelicaException** がスローされます。
- i アプリが FeliCa チップをオープンしている状態であっても、アクセスが同時に行われない限りにおいては外部 R/W からその FeliCa チップにアクセスすることができます。i アプリが FeliCa チップをオープンしている状態で、外部 R/W が FeliCa チップにアクセスしているタイミングで i アプリから FeliCa チップにアクセスしようすると **PortBusyException** が発生します。この例外が発生した場合は、携帯電話を外部 R/W から離す必要がある旨のメッセージをユーザーに提示するなどし、適宜リトライを行うようにしてください。
- フリーエリアを使用する i アプリでは、ADF に **FeliCaCPID** キーと **FeliCaPIN** キーが適切に宣言されていなければなりません。これらのキーの詳細については後述します。
- FeliCa チップは携帯電話に内蔵されており、取り外すことはできません。また、携帯電話の故障などによって FeliCa チップも使用不可能な状態になる可能性があります。重要なデータについては FeliCa チップ上のみに保存するのではなく、サーバーにバックアップを保存するなどの対策を検討するようにしてください。
- フリーエリアに格納されたデータは、通常は ADF の **FeliCaPIN** キーで指定される PIN（FeliCa チップの持つパスワード機能：次項参照）によって保護されています。この PIN は i アプリからフリーエリアのデータを読み書きする際にシステムにより自動的に解除されますが、一度解除された PIN が再び有効となるのは FeliCa チップが次にクローズされるタイミングとなります。この PIN が解除されている区間を極力短くするには、**FreeArea.setReset()** メソッドを利用します。このメソッドを引数に **true** を指定して呼び出すと、i アプリからフリーエリアの読み書きを行うたびに FeliCa チップがリセットされ、解除された PIN がすぐに有効化されるようになります。

10.3 フリーエリア利用のための ADF の設定

フリーエリアを利用する i アプリでは、ダウンロード時にその i アプリと空きフリーエリアを適切に対応付けるために、ADF でフリーエリアを使用する旨の宣言を行う必要があります。また、この宣言で設定される情報は、外部 R/W からの FeliCa チップアクセスにおいて、3 つのフリーエリアの中から目的の（アプリケーション開発者やサービス提供者が意図する）フリーエリアを選択するためのキーとしての役割も持ちます。外部 R/W からのフリーエリア選択の手順などについては、10.4 項を参照してください。

フリーエリア利用のために用意されている ADF のキーは以下の通りです。FeliCa チップ搭載 i モード携帯電話では、これらの宣言を行った i アプリを最大 3 つまでダウンロードすることができます。

FeliCaCPID： フリーエリア利用上の必須キー

FeliCaCPID キーは、携帯電話内に存在するフリーエリア利用 i アプリを一意に識別するための識別子です。FeliCaCPID キーには、32 桁の 16 進数字文字列（16 バイト分のバイト列データを示す値）を設定します。この値は i アプリダウンロード時に、フリーエリア上に設けられた FeliCaCPID 格納データブロック（10.4 項参照）に格納されます。

例： FeliCaCPID = 0123456789abcdef0123456789abcdef

携帯電話にダウンロードしようとしている i アプリの FeliCaCPID キーの設定値が、すでにその携帯電話内にダウンロードされているいずれかの i アプリの FeliCaCPID キーの値と一致している場合、その i アプリはダウンロードすることはできません。

フリーエリアは i モードサーバーに登録されていない一般の i アプリ（トラステッド i アプリではない i アプリ）でも使用することができ、本キーにどのような値を設定するかはコンテンツプロバイダに委ねられます。本キーの値が、他のコンテンツプロバイダが使用している値と衝突しないことの保証は得られません。一方、上述したように 1 台の携帯電話内で同一の FeliCaCPID キーを持つ i アプリを複数共存させることはできません。このため、NTT ドコモが配布する i アプリ開発環境には i アプリの ADF URL などからハッシュ値を生成して、衝突の可能性が低い FeliCaCPID キー値を設定する機能が用意されています。本キーの値の決定には i アプリ開発環境のこの機能を使用することを強く推奨します。

FeliCaPIN： フリーエリア利用上の必須キー

i アプリに対応付けられたフリーエリアに設定する PIN（Personal Identification Number : FeliCa チップの持つパスワード機能）を、100000 以上 8 桁以内の 10 進数字文字列で設定します。

例： FeliCaPIN = 12345678

FeliCa チップでは相互認証や通信の暗号化など、鍵を使用したアクセス手段が提供されていますが、フリーエリアではその性質上鍵を使用することができません。フリーエリアでは、外部 R/W からのアクセスに一定の制限をかけるため、FeliCa チップの PIN 機能を使用しています。フリーエリアを利用する i アプリがダウンロードされる際、その i アプリと対応付けられたフリーエリアの PIN 値として本キーの値が設定されます。

i アプリからの（i アプリ API を使用しての）フリーエリアアクセス時にはシステムにより自動的に PIN の照合、解除が行われますが、外部 R/W からアクセスを行う場合には必要な手順に従って PIN の照合、解除を行う必要があります。また、フリーエリアの PIN は ADF の FeliCaPIN キーでのみ設定可能であり、i アプリから動的に PIN を変更することはできません。

注意事項：

- これらのキーによりフリーエリアを利用するとして携帯電話にダウンロードされた i アプリを、キーを削除してフリーエリアを利用しない i アプリにバージョンアップすることはできません。この場合は i アプリのバージョンアップではなく、いったん i アプリを削除して再度ダウンロードし直す必要があります。なお、FeliCaCPID キーおよび FeliCaPIN キーのいずれも、バージョンアップ時に値を変更することは可能です。

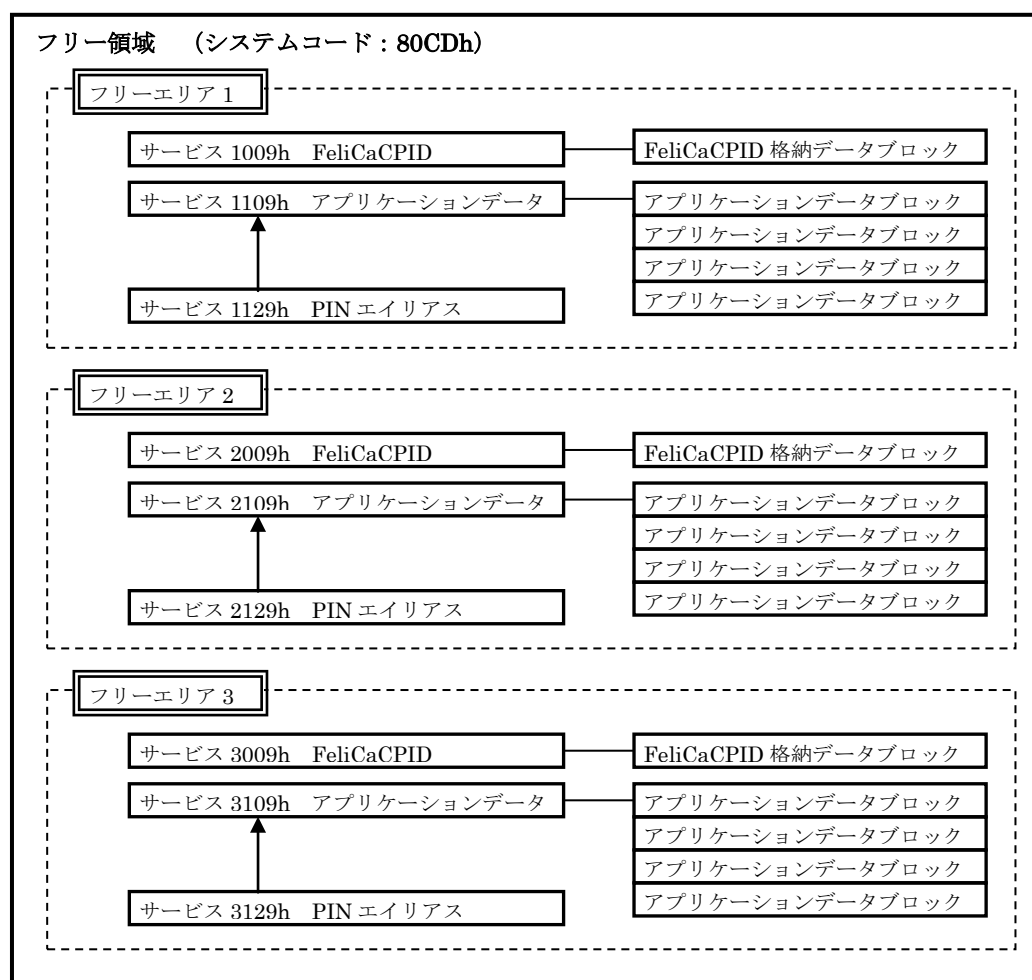
- これらのキーが指定されている i アプリをバージョンアップする際、それまでフリーエリアに格納されていたアプリケーションデータはそのまま引き継がれます。また、これらのキーが指定されている i アプリを削除すると、その i アプリに対応付けられていたフリーエリアの内容は **FeliCaCPID** 格納ブロックや PIN の設定を含め全てクリアされます。

10.4 外部 R/W からのフリーエリアへのアクセス

外部 R/W からフリーエリアにアクセスするためには、フリー領域およびフリーエリアの構造を知る必要があります。以下に、これらの構造の詳細を示します。なお、以下の図中の表記で“h”は 16 進数値を表すものとします（80CDh など）。

なお、下記の説明に現れる FeliCa 技術方式の用語については、FeliCa 技術製品（外部 R/W など）のマニュアル等を参照してください。また、FeliCa チップに関する一般的な情報については、ソニー株式会社の以下のサイトも併せて参照してください。

<http://www.sony.co.jp/Products/felica/>



以下に、各々のフリーエリアを構成するサービスの機能について説明します。

FeliCaCPID (サービスコード 1009h、2009h、3009h)

このサービスがポイントするデータブロック (1 ブロック) には、ADF の **FeliCaCPID** キーで指定した識別子 (32 桁の 16 進数値) が 16 バイトのバイトイメージで格納されています。このサービスは鍵なし・Read/Write 可能なランダムサービスとして定義されており、外部 R/W はこの値が自 i アプリの **FeliCaCPID**

キーの値と一致するフリーエリアを探すことで自 i アプリと対応付けられたフリーエリアを検索することができます。

PIN エイリアス (サービスコード 1129h、2129h、3129h)

アプリケーションデータのサービスを保護する PIN 機能で使用されます。外部 R/W がアプリケーションデータのサービスにアクセスするには、この PIN エイリアスに対して PIN の照合、解除を行う必要があります (i アプリからのアクセスでは PIN の照合、解除はシステムにより自動的に行われます)。i アプリに対応付けられたフリーエリアでは、ADF の FeliCaPIN キーの値 (100000 以上 8 桁以内の 10 進数値) が 4 バイト整数値として PIN 値に設定されています。

アプリケーションデータ (サービスコード 1109h、2109h、3109h)

i アプリと外部 R/W は、このサービスがポイントするデータブロック (4 ブロック) でデータを共有します。このサービスは鍵なし・Read/Write 可能なランダムサービスとして定義されており、外部 R/W は PIN の照合、解除を行った後にこのデータの読み取り、書き込みを行うことができます。

上記構造に従って外部 R/W からフリーエリアアクセスを行う際は下記の通りです。

- 1) フリー領域のシステムをポーリングする。
Polling コマンドを使用して、外部 R/W への FeliCa チップの接近を待ちます。この際、ポーリング対象の FeliCa チップをシステムコード (80CDh) で特定することで、フリー領域を持つ FeliCa チップ (i モード携帯電話に搭載された FeliCa チップ) に限定してポーリングすることができます。
- 2) FeliCaCPID を検索する。
1)でフリー領域を持つ FeliCa チップを検出したら、それに含まれる各フリーエリアの FeliCaCPID のサービス (サービスコード 1009h、2009h、3009h) を読み込み、自 i アプリと同一の FeliCaCPID が格納されているものを探します。FeliCaCPID 格納データブロックには、ADF の FeliCaCPID キーに設定された 32 桁の 16 進数値が 16 バイトのバイトイメージとして格納されています。
自 i アプリと同一の FeliCaCPID が見つからなかった場合、その携帯電話には対応する i アプリはダウンロードされていないものとみなします。
- 3) 目的のフリーエリアに対し、アプリケーションデータの PIN 照合、解除を行う。
2)で目的のフリーエリアが特定できたら、PIN エイリアス (サービスコード 1129h、2129h、3129h のいずれか) に対して PIN の照合、解除を行います。PIN 値には、対応する i アプリの ADF の FeliCaPIN キーに指定した数値を 4 バイト整数値として指定します。
- 4) アプリケーションデータにアクセスする。
3)で PIN の解除に成功すると、アプリケーションデータにアクセス (Read および Write) できるようになります。

注意事項：

- 領域のポーリングを行うと、その結果として外部 R/W は領域の製造 ID ブロック (製造機器コードや製造シリアル番号、チップ型番/バージョンなど、FeliCa 技術方式で定められた各種 ID 情報が格納された特殊なブロック) の内容を取得することができます。製造 ID ブロックの情報には携帯電話の製造ロットにより変化する可能性のあるものも含まれるため、i モード携帯電話に搭載された FeliCa チップのアクセスを処理する外部 R/W では製造 ID ブロックの値に基づいた処理を行う場合には注意が必要です。NTT ドコモからは、i モード携帯電話に搭載された FeliCa チップの製造ブロックの内容に関する情報は公開されません。また同一機種であっても、ある製造ロット以降にこれらの内容が予告なく変更される可能性があります。

10.5 フリー領域のセキュリティに関する注意事項

フリー領域ではその利用のために特別な登録申請を行う必要がない一方、領域やエリアを複数の i アプリ (および外部 R/W) で共有することに起因するセキュリティ上のリスクがあります。本項ではこのリスクについて解説します。

FeliCaCPID 格納データブロック

FeliCaCPID 格納データブロックは鍵なし・Read/Write 可能なサービスです。従って、悪意のある外部 R/W がこのデータブロックの値を書き換えてしまうかもしれません。

携帯電話システムは何番目のフリーエリアにどの i アプリを対応付けているかを内部的に管理しています。i アプリを起動した際に、万一その i アプリに対応付けられているフリーエリアの FeliCaCPID 格納データブロックが書き換えられていた場合には、携帯電話システムは自動的に本来の値（ADF の FeliCaCPID キーに設定された値）に書き戻しを行います。

外部 R/W が FeliCaCPID 格納データブロックの内容を書き換えた以降、ユーザーが対応する i アプリを起動するまでの間は、外部 R/W は目的の FeliCaCPID の検索を行うことはできません。

アプリケーションデータブロック

アプリケーションデータブロックは鍵なし・Read/Write 可能なサービスですが、FeliCaCPID 格納データブロックと異なり PIN によるデータの保護が行われています。ただし、悪意のある外部 R/W からの総当たりなどの方法により PIN が解除された場合には、やはりこのデータブロックの値を読み出したり書き込んだりすることができます。

金銭的価値を持つデータを取り扱う場合には共通領域の利用を検討するようにしてください。また、フリーエリア上で金銭的価値に近い意味を持つデータを取り扱う場合には、ユーザーの i アプリ操作によってデータを（タイムスタンプなどの信頼性を向上させる補助データとともに）FeliCa チップに移し、そのユーザー操作の延長上で外部 R/W にアクセスさせるなどの方法を検討してください。

なお、i アプリに対応付けられていないフリーエリアのアプリケーションデータブロックに外部 R/W が値を書き込んだ場合、後刻そのフリーエリアに対応付けられた i アプリには外部 R/W が書き込んだデータが初期状態として提供されます。すなわち、i アプリダウンロード直後において、フリーエリアのアプリケーションブロックが常に初期状態（0 クリアされた状態）であることは厳密には保証されないことに注意してください。

PIN 値

携帯電話システムは、フリーエリア利用 i アプリをダウンロードする際に、その i アプリに対応付けられたフリーエリアの PIN に ADF の FeliCaPIN キーで指定された値を設定します。また i アプリからのフリーエリアアクセス時、およびフリーエリア利用 i アプリの削除時に、FeliCaPIN キーで指定された値で PIN の解除を行います。従って、外部 R/W によって PIN の値が変更されてしまった場合、そのフリーエリアは i アプリから使用することができなくなり、またその i アプリを削除することもできなくなります。

従って、外部 R/W からフリーエリアの PIN を変更することはしないようにしてください。悪意のある外部 R/W が PIN を変更した場合の対策として、NTT ドコモより各フリーエリアの PIN を初期状態に戻し、i アプリからのアクセスを再度可能とするためのユーザー向けツールが提供される予定です。

なお、FeliCa チップ搭載携帯電話では、FeliCa チップは 1 つの UIM カードとのみ紐付けられて利用可能となります。このため、携帯電話に FeliCa 利用アプリをダウンロードしていたり、プレインストールの FeliCa 利用アプリを利用している状況では、UIM カードを抜いたり差し換えたりすると保護機構が動作して、以下のような動作制限が働きます。

- UIM カードを抜いたり差し換えている間は、他の FeliCa 利用アプリをダウンロードすることができなくなります。
- UIM カードを抜いたり差し換えている間は FeliCa チップ上のエリアやサービスが隠蔽され、外部 R/W からエリアやサービスにアクセスすることができなくなります（外部 R/W からは、エリアやサービスが存在していない場合と同じ見え方となります）。

UIM カードを元に戻すと制限も解除されます。また FeliCa 利用アプリを全て削除すれば、別の UIM カードで FeliCa チップを利用することも可能となります。

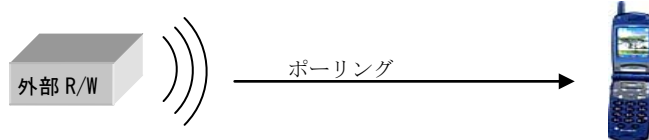
保護機構の動作契機は、UIM カードを抜いたり差し換えた後の最初の電源投入時、もしくは携帯電話の FeliCa ロック機能をオフにするユーザー操作が行われた時となります。

10.6 外部 R/W からの指示による携帯電話内の i アプリ起動

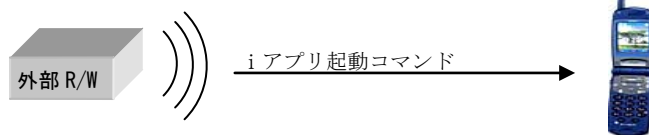
FeliCa チップ搭載 i モード携帯電話の中にはオプション機能として、外部 R/W から携帯電話上の FeliCa チップに所定の信号（コマンド）を送信することにより、携帯電話内にダウンロードされている特定の i アプリを起動する機能を持つものがあります。

本機能のイメージを以下に示します。

- 1) 外部 R/W から携帯電話上の FeliCa チップを捕捉（ポーリング）する。



- 2) 検知した FeliCa チップに対し、外部 R/W から i アプリ起動のためのコマンドを送出する。



- 3) i アプリ起動コマンドの内容に従って、携帯電話内の i アプリ(*1)が起動される。



(*1) 起動対象の i アプリは、FeliCa チップ機能の API を使用する i アプリである必要はありません。FeliCa チップ機能の API を使用しない i アプリであっても、本項に記載する手順に従うことで外部 R/W からの起動を行うことができます。

外部 R/W から携帯電話内の特定機能を起動するための、コマンドシーケンスやデータフォーマットの仕様（「外部リーダー/ライターから携帯端末内の特定機能を起動するデータフォーマット仕様書」以降、データフォーマット仕様と略記します）はフェリカネットワークス株式会社により策定されており、i モード携帯電話における外部 R/W からの i アプリ起動もこのデータフォーマット仕様に基づいています。

データフォーマット仕様、およびこの仕様を利用することのできる外部 R/W 機器等に関する情報は別途フェリカネットワークス株式会社より公開されますが、本項ではデータフォーマット仕様を i モード携帯電話に適用する際の詳細仕様および注意点について記載します。

原仕様であるデータフォーマット仕様についてはフェリカネットワークス株式会社より入手・参照してください（なお、現在は公開終了しています）。以下の解説ではデータフォーマット仕様上の用語等も使用しています。

(1) コマンドシーケンス

外部 R/W から i アプリを起動する際の外部 R/W 側のコマンドシーケンスについては、データフォーマット仕様からの変更点や注意点はありません。i モード携帯電話においても、データフォーマット仕様に記載された通りのコマンド発行シーケンスにて携帯電話内の i アプリを起動することができます。

(2) Push コマンド送信データのフォーマット

外部 R/W から携帯電話内の特定機能を起動するには、FeliCa チップの Push コマンドを使用して、起動したい機能を表現するデータを携帯電話（携帯電話上の FeliCa チップ）に送信します。この送信データのフォーマットについてもデータフォーマット仕様にて規定されていますが、i モード携帯電話における外部 R/W からの i アプリ起動においては以下の内容について注意してください。

—個別部数

データフォーマット仕様では、外部 R/W は携帯電話に対して、1 回のコマンドシーケンスで起動対象機能の情報（個別部）を複数送信することができます。これに対し、外部 R/W からの i アプリ起動では 1 回のコマンドシーケンスで受け付けることのできる個別部数は 1 つだけです。

—個別部ヘッダ

個別部ヘッダの起動制御情報には、アプリケーション（i アプリ）の起動を示す 0x01 のみが指定可能です。起動制御情報として 0x01 以外が指定されている個別部は無効となります。

—個別部パラメータ

外部 R/W からの i アプリの起動では、個別部パラメータのデータ項目にはそれぞれ以下の内容を対応付けて設定します。

アプリケーション URL フィールドおよびアプリケーション URL 長フィールド： 必須

アプリケーション URL には、起動対象とする i アプリの ADF URL を ASCII コード（ISO-8859-1）で指定します。ここで指定する ADF URL と同一ロケーションの ADF を持つ i アプリが携帯電話内にダウンロードされている場合に、その i アプリが起動対象となります。

起動チェックコードフィールドおよび起動チェックコード長フィールド： 必須

起動チェックコードには、起動対象とする i アプリの ADF の AllowPushBy キー（次項(3)参照）に設定した起動コマンドの内容を ASCII コード（ISO-8859-1）で指定します。アプリケーション URL が合致する i アプリが携帯電話内にダウンロードされていても、その i アプリの ADF に設定されている起動コマンドがこの起動チェックコードと異なっている場合には、その i アプリは起動されません。

アプリケーション起動パラメータフィールド： オプション

アプリケーション起動パラメータには、StarApplicationManager.getParameter()メソッドで取得可能なパラメータを必要に応じて設定することができます。パラメータはその名前と値を半角スペース（0x20）で区切って指定します。名前と値のセットをさらに半角スペースで区切って並べることで、複数のパラメータを設定することもできます。パラメータの名前と値には ASCII コード（ISO-8859-1）のみ使用可能です。

以下にパラメータの設定例を記載します（下記において▲は半角スペースを表すものとします）。

例1) name1▲value1

パラメータ名"name1"に対し値"value1"を設定。

例2) name1▲value1▲name2▲value2

パラメータ名"name1"に対し値"value1"を、パラメータ名"name2"に対し値"value2"を設定。

例3) name1▲value1▲name2▲▲name3▲value3

パラメータ名"name1"に対し値"value1"を、パラメータ名"name2"に対し値""（空文字列）を、パラメータ名"name3"に対し値"value3"を設定。

(3) 外部 R/W から i アプリを起動するための ADF の設定

外部 R/W から起動させることを想定した i アプリでは、ADF の AllowPushBy キーを使用して起動コマンドの設定を行う必要があります。外部 R/W から i アプリを起動させる場合、AllowPushBy キーの値には、以下の形式のパラメータを 1 つだけ ASCII コード（ISO-8859-1）で指定します。

icc:<command>

<command>には、i アプリ起動を許可する外部 R/W が起動チェックコードとして送信すべき値を指定します。この値が外部 R/W の送信する起動チェックコードと完全一致した場合、その外部 R/W が i アプリを起動することを許可します。

なお、IrOBEX や 2 次元バーコードによる i アプリ起動では AllowPushBy キーの起動コマンドの値はコンテンツプロバイダ側で任意に決定するのに対し、外部 R/W からの i アプリ起動においては起動コマンド（起動チェックコード）の値はコンテンツプロバイダ間で重複が発生しないようフェリカネットワークス株式会社にて管理されます。起動チェックコードの入手方法などの詳細についてはデータフォーマット仕様を参照してください。

注意事項：

- 携帯電話ユーザーは、携帯電話の設定により外部 R/W からの i アプリ起動を禁止することができます。外部 R/W からの i アプリ起動が禁止されている場合には、i アプリ起動コマンドを受信しても i アプリは起動されません。
- 起動先 i アプリはフルのみアプリか、フル+ミニアプリでなければなりません。また、起動先 i アプリが、フル+ミニアプリである場合、フルアプリが起動されます。
- 携帯電話上で i アプリが起動されている場合、その他、携帯電話上で別の機能が起動していて i アプリを起動することができない場合には、外部 R/W から i アプリの起動が指示されてもシステムはその指示を無視します。
- 外部 R/W から起動された i アプリの動作が終了 (`StarApplication.terminate()`により終了) すると、携帯電話は待ち受け状態に戻ります。ただし、i アプリが強制終了された場合など、上記以外の方法で i アプリが終了した場合は、i アプリ終了後にどのような状態に移るかはメーカーにより異なります。
- Push コマンドで送信可能なデータのサイズは最大 192 バイトであり、本項で解説した送信データは可変長の要素 (ADF URL、起動コマンド、起動パラメータ) を含めて 192 バイトに収める必要があります。
- 外部 R/W が、セキュリティ設定されたサービスにアクセスするための認証を FeliCa チップに対して行っている状態 (FeliCa チップがモード 0 以外の状態) で i アプリを起動しようとする、i アプリが起動できない場合があります。FeliCa チップに対して認証を行った後に i アプリを起動する場合には、一旦外部 R/W の搬送波を止めるなどにより FeliCa チップのモードを 0 とするようにしてください。
- フリーエリアを利用する i アプリを起動する際、および共通領域にエリアを登録する機能を持つ i アプリを起動する際には、i アプリ起動前に携帯電話システムによる FeliCa チップへのアクセスが発生します。外部 R/W から i アプリ起動を行った後に、外部 R/W が携帯電話上の FeliCa チップに対して FeliCa コマンド (ポーリングコマンドを含みます) を発行すると、外部 R/W のチップアクセスと携帯電話システムのチップアクセスが競合して i アプリが起動できない場合がありますので注意してください。携帯電話システムがチップアクセスを行うために要する時間は、機種によって異なりますが、外部 R/W が i アプリを起動するためのコマンドを送出した後おおむね 3 秒程度以内となります。
- 外部 R/W から i アプリを起動する場合、外部 R/W は FeliCa チップの電源保持のために、i アプリ起動のためのコマンドを送信したのち一定時間は搬送波を送出し続ける必要があります。確保すべき最小の搬送波送出時間は機種により異なりますが、どの機種でも問題なくアプリを起動できるようにするためには概ね 3 秒程度以上を確保する必要があります (前項の制約により、この間は FeliCa コマンドを送信しないようにしなければなりません)。

しかし、搬送波によって電源が保持されている FeliCa チップに対しては、i アプリは FeliCa 関連 API を使用してアクセスすることができないという制約があります。また一部の機種では、外部 R/W からの i アプリ起動では搬送波が停止するまで i アプリの起動が保留されるものがあります。

このような場合の問題を避けるため、外部 R/W は、i アプリ起動のためのコマンドを送信したのち 3 秒程度は搬送波送出を継続し、その後数秒間搬送波を停止してから次の処理 (次に処理すべき FeliCa チップのポーリングなど) を行うようにしてください。また i アプリからの FeliCa 関連 API の利用は、搬送波が停止している区間、もしくはユーザーが外部 R/W に携帯電話をかざし終えた後に行うようにしてください。

10.7 FeliCa チップによるアドホック通信

携帯電話で採用されているモバイル FeliCa IC チップ (モバイル FeliCa OS Version 2.0 搭載の FeliCa チップ) には、FeliCa チップを搭載した機器間で近傍無線通信を行うためのアドホック通信機能が搭載されています。この機能を利用することで、FeliCa チップを搭載した 2 つの機器上のアプリケーションが、FeliCa 技術方式の無線通信路を介した双方向のデータ通信を行うことができるようになります。現在のプロファイルでは、FeliCa チップのアドホック通信機能を応用して、2 つの携帯電話上で動作している i アプリ間で OBEX (オブジェクト交換) 通信を行うための API (FeliCa アドホック通信 API) が用意されています。

本項では、FeliCa アドホック通信 API の利用方法について解説します。なお、OBEX 通信の基本的な概念などは、赤外線通信上での OBEX 通信（IrOBEX 通信）と同様です。これらについては、「i アプリコンテンツ開発ガイド 詳細編」10.1 項も併せて参考としてください。

なお、FeliCa アドホック通信 API は FeliCa チップ搭載機種の中でもオプションの機能です。FeliCa チップを搭載している機種であっても、本機能をサポートしていないものもありますので注意してください。

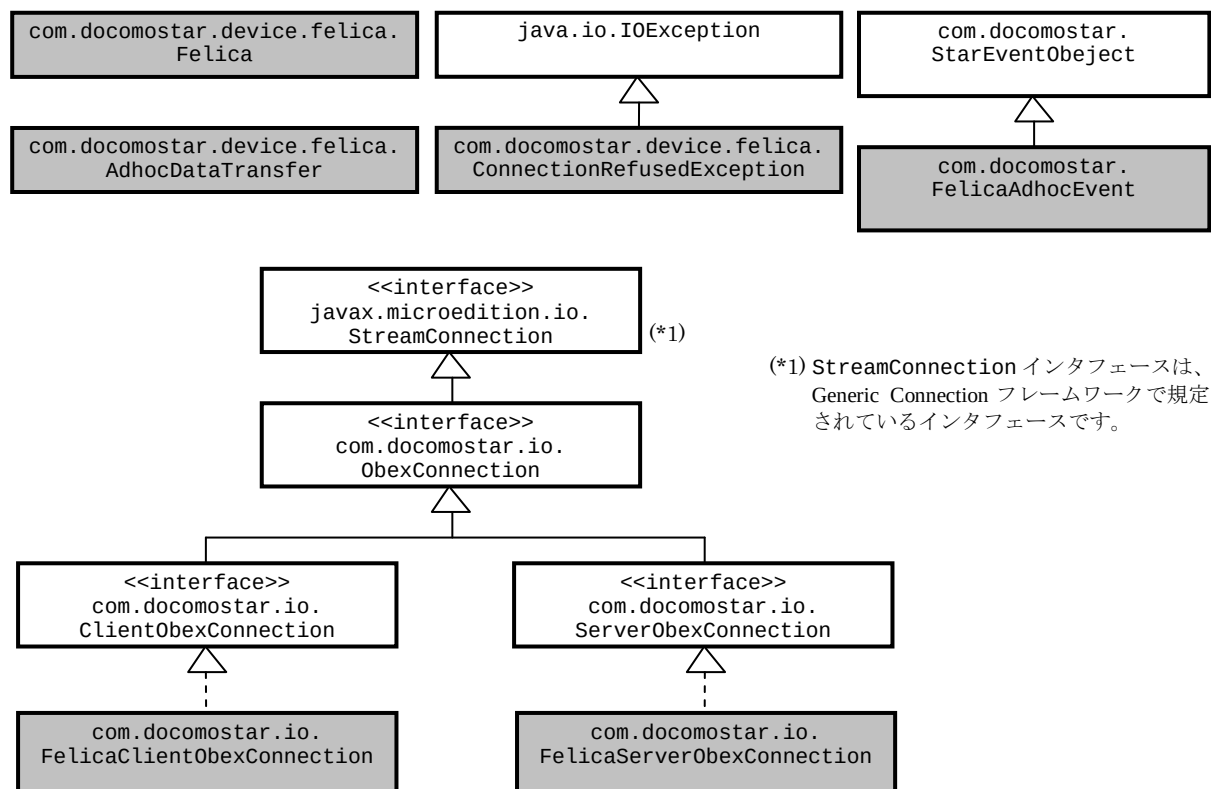
注意事項：

- FeliCa アドホック通信 API の機能は、フリーエリアや共通領域へのアクセス機能と同時に使用することはできません。i アプリが FeliCa アドホック通信 API を使用している間は、フリーエリアや共通領域へのアクセスは行わないようにしてください。
- FeliCa アドホック通信での通信速度は、メーカーの実装にもよりますが概ね数 10kbps 程度となります。
- FeliCa アドホック通信は、FeliCa チップ間の無線通信が安定して行える状況の中で行われなければなりません。原則として、FeliCa アドホック通信は携帯電話に記載された FeliCa のプラットフォームマーク同士を密着させた状態で行う必要があります。

10.7.1 FeliCa アドホック通信 API の概要

FeliCa アドホック通信 API は、FeliCa アドホック通信のための通信リンクを制御するための API と、確立された通信リンク上で実際にデータを送受信するための API に大別されます。データ送受信の API は Generic Connection フレームワーク上に構築されています。

以下に、FeliCa アドホック通信 API のクラス構成を示します。関係するクラス、インタフェースのうち、FeliCa アドホック通信に直接関係するもの（本項での説明対象）にはハッチングを施しています。



FeliCa アドホック通信に関する各クラスの機能概要は以下の通りです。FeliCa アドホック通信 API では、クライアント側 i アプリとサーバー側 i アプリで使用すべきクラスが異なります。

com.docomostar.device.felica パッケージ：

クラス	機能概要
Felica	FeliCa チップを表現するクラスです。FeliCa アドホック通信のクライアント側 i アプリでは、FeliCa アドホック通信を行う際に本クラスを使用して FeliCa チップのデバイスをオープンする必要があります。
AdhocDataTransfer	FeliCa アドホック通信のクライアント側 i アプリあるいはサーバー側 i アプリにて、通信リンクの制御などを行うためのクラスです。
ConnectionRefusedException	FeliCa アドホック通信のクライアント側 i アプリがサーバー側 i アプリに通信リンク確立の要求を出した際に、その要求が拒否されたことを示す例外クラスです。
FelicaAdhocEvent	FeliCa アドホック通信のサーバー側 i アプリが、クライアント側 i アプリからの通信リンク確立要求を受けたことを表すイベントクラスです。サーバー側 i アプリでは、クライアント側からの通信リンク確立要求を、Star アプリケーションイベントの機構を用いて検知します。サーバー側 i アプリでは、この FelicaAdhocEvent オブジェクトから AdhocDataTransfer オブジェクトを取得し、クライアント側 i アプリに通信リンク確立要求に対する応答を返します。

com.docomostar.io パッケージ：

クラス	機能概要
FelicaClientObexConnection	確立された通信リンク上で、FeliCa アドホック通信のクライアント側 i アプリが、OBEX 通信によるデータ送受信を行うためのクラスです。このクラスのオブジェクトは、Generic Connection フレームワークの Connector クラスを使用して取得します。
FelicaServerObexConnection	確立された通信リンク上で、FeliCa アドホック通信のサーバー側 i アプリが、OBEX 通信によるデータ送受信を行うためのクラスです。このクラスのオブジェクトは、Generic Connection フレームワークの Connector クラスを使用して取得します。

クライアント側 i アプリ、サーバー側 i アプリともに、それぞれの手順に従って通信リンクの確立を行った後に、データ送受信の API を使用してデータの送受を行います。以下に、通信リンク制御とデータ送受信のそれぞれについての処理の流れを示します。

10.7.2 通信リンク制御

FeliCa アドホック通信では、クライアント側 i アプリが行った通信リンクの確立要求の通知を受けたサーバー側 i アプリが、その要求に対する受け入れ処理を行うことでクライアント・サーバー間の通信リンクが確立します。サーバー側 i アプリは、携帯電話上で起動された状態であっても構いませんし、起動されていない状態であっても構いません。サーバー側 i アプリが携帯電話上で起動していない状態で通信リンクの確立要求が行われた場合、該当するサーバー側 i アプリが自動的に起動された上で、その i アプリに確立要求の通知が配送されます。

クライアント側 i アプリにおける通信リンク確立要求

クライアント側 i アプリで通信リンクを確立するには、以下の手順で処理を行います。

- (1) FeliCa チップのオープン (**Felica.open()**)
- (2) **AdhocDataTransfer** オブジェクトの取得 (**Felica.getAdhocDataTransfer()**)
- (3) 通信リンクの確立 (**AdhocDataTransfer.setup()**)

`AdhocDataTransfer.setup()`メソッドでは、以下の情報を引数に指定します。

- ・ サーバー側 i アプリの ADF の URL

携帯電話は i アプリをダウンロードした際に、その i アプリに対応する ADF の URL を記憶しています。クライアント側 i アプリは通信先となるサーバー側 i アプリの ADF の URL を引数として指定します。対向先となる携帯電話上ではこの情報を利用して実際に通信を行わせる i アプリを特定します。

指定されたサーバー側 i アプリが携帯電話上で実行されている状態であれば、その実行中の i アプリにリンク開設要求があった旨のイベント通知が行われます。また指定されたサーバー側 i アプリが起動されていなかった場合には、その i アプリが自動的に起動されます。

- ・ サーバー側 i アプリの ADF に設定された、`AllowPushBy` キーの起動コマンド

FeliCa アドホック通信のサーバー側 i アプリでは、ADF の `AllowPushBy` キーを使用して、その i アプリに対応する起動コマンドが適切に宣言されている必要があります。クライアント側 i アプリはリンクを確立する際に、サーバー側 i アプリの起動コマンドを引数として指定します。

サーバー側 i アプリにおける ADF (`AllowPushBy` キー) の設定については次項を参照してください。

- ・ リンク開設要求の通知時にサーバー側 i アプリに引き渡すパラメータ（利用はアプリケーション側の任意）

FeliCa アドホック通信のための通信リンク確立要求を行う際、クライアント側 i アプリは任意の文字列パラメータをサーバー側 i アプリに渡すことができます。

パラメータの受け渡しには `java.util.Hashtable` を使用します。`Hashtable` に設定するキーと値は、空白を除く ASCII 図形文字で構成される文字列 (`String` オブジェクト) である必要があります。またキーと値のペアは 16 組以内で、キー長と値の長さの総合計（全てのペアの合計）は 255 文字以内としてください。

`AdhocDataTransfer.setup()`メソッドを呼び出した後で通信先の携帯電話と FeliCa チップをかざし合い、サーバー側 i アプリが通信リンク開設要求を受け入れると実際に携帯電話間で通信リンクが確立され、`AdhocDataTransfer.setup()`メソッドは正常に復帰します。通信先の携帯電話上でサーバー側 i アプリを起動できない状態（他のアプリケーションを使用している場合や、`WidgetView` が表示されている場合など）やサーバー側 i アプリがリンク確立要求を受け入れなかった場合（次節参照）には、`setup()`メソッドは正常復帰せず `ConnectionRefusedException` がスローされます。`ConnectionRefusedException` は `getStatus()`メソッドを備えており、リンク確立要求が拒否された大まかな理由を調べることができます。

通信リンクが確立された状態では、クライアント側 i アプリは `AdhocDataTransfer.getIdm()`メソッドを使用してサーバー側 i アプリ側の FeliCa チップの IDm を知ることができます。また、FeliCa アドホック通信が完了して通信リンクが不要になった際には、`AdhocDataTransfer.terminateAdhoc()`メソッドを使用して通信リンクを切断します。

サーバー側 i アプリにおける通信リンク確立要求の受け入れ処理

FeliCa アドホック通信の通信リンク確立要求は、Star アプリケーションイベントの機構によりサーバー側 i アプリに通知されます。FeliCa アドホック通信のサーバー側 i アプリが、クライアント側 i アプリからの通信リンク確立要求を受信するには、事前に `StarApplication.addEventListener()`メソッドに `StarEventObject.STAR_FELICA_ADHOC_REQUEST_RECEIVED` を指定してイベントリスナオブジェクトを登録しておく必要があります。i アプリ実行環境は登録されたイベントリスナオブジェクトの `updateStarApplication()`メソッドを通じて、クライアント側 i アプリから通信リンク確立要求を受け取ったことをサーバー側 i アプリに伝えます。このとき、`updateStarApplication()`メソッドの引数には `FelicaAdhocEvent` オブジェクトが設定されています。

該当するサーバー側 i アプリ（クライアント側 i アプリが `AdhocDataTransfer.setup()`メソッドで指定した ADF URL に該当する i アプリ）が携帯電話上で起動されていなかった場合、i アプリ実行環境はその i アプリを起動します。なお、通信リンク確立要求を受け取ったことにより起動された i アプリの起動種別

（`StarApplication.started()`メソッドの引数に設定される起動種別）には `StarApplication.LAUNCHED_FROM_FELICA_ADHOC` が設定されています。この起動種別を伴って i アプリが起動した場合、イベントリスナオブジェクトの `updateStarApplication()`メソッドは呼び出されません。この場合、通信リンク確立要求のイベントオブジェクトは、

`StarEventObject.STAR_FELICA_ADHOC_REQUEST_RECEIVED` を指定して
`StarApplicationManager.getPushedEvent()` メソッドを呼び出すことで取得することができます。

このようにして取得した `FelicaAdhocEvent` オブジェクトには `getReceivedParameter()` メソッドが定義されており、このメソッドを呼び出すことで得られる `Hashtable` オブジェクトには、クライアント側 i アプリが `AdhocDataTransfer.setup()` メソッドの引数の `Hashtable` オブジェクトに設定したキーと値のテキストが設定されています。

また、`FelicaAdhocEvent` オブジェクトには `getAdhocDataTransfer()` メソッドが定義されており、このメソッドを使用して `AdhocDataTransfer` オブジェクトを取得することができます。サーバー側 i アプリは、このメソッドを呼び出して取得した `AdhocDataTransfer` オブジェクトの `reply()` メソッドを使用して、クライアント側 i アプリに要求を受け入れるか否かを通知します。アプリケーションプログラムが `AdhocDataTransfer.REPLY_ACCEPT` を指定して `reply()` メソッドを呼び出した場合にクライアント側 i アプリとサーバー側 i アプリの間で通信リンクが確立します。アプリケーションプログラムが `AdhocDataTransfer.REPLY_REJECT` を指定して `reply()` メソッドを呼び出した場合は、通信リンク開設要求は拒否されたものとして取り扱われます。

なお、サーバー側 i アプリの ADF には、`AllowPushBy` キーを使用して、i アプリの起動コマンドを宣言する必要があります。`AllowPushBy` キーの書式は以下の通りです。

`AllowPushBy = icc:<起動コマンド>`

起動コマンドには、1 文字以上 251 文字以下の ASCII 文字列を指定することができます。`AllowPushBy` キーで設定した起動コマンドと、クライアント側 i アプリが `AdhocDataTransfer.setup()` メソッドの引数に指定する起動コマンドが一致していない場合には、FeliCa アドホック通信を行うことはできません。

上記の手続きによって通信リンクが確立すると、次項に記載する方法によってデータ送受信を行うことができます。

注意事項：

- 1 つの i アプリから同時に確立可能な通信リンクは 1 つだけです。
- サーバー側 i アプリは、フルのみアプリか、フル+ミニアプリでなければなりません。また、サーバー側 i アプリがフル+ミニアプリである場合、通信リンク確立要求を受けてその i アプリが起動する場合は、フルアプリが起動します。
- クライアント側 i アプリから指定されたサーバー側 i アプリがフル+ミニアプリである場合において、そのミニアプリが実行中の状態では、通信リンクを確立することはできません。
- ユーザーが携帯電話をセルフモードにしている状態では、通信リンクを確立することはできません。
- 通信リンクを保つためには、携帯電話の FeliCa チップをかざし続けている必要があります。通信中に FeliCa チップを遠ざけると、通信リンクは切断されます。
- i アプリが通信リンクの確立を行っている間は、i アプリ実行環境は通信を実行中である旨のメッセージを表示します。また通信リンクの確立が行えなかった場合や、タイムアウト、ユーザー操作などにより通信が中断された場合には、i アプリ実行環境はその旨をユーザーに伝えるメッセージを表示します。これらのメッセージを非表示とすることはできません。
- `FelicaAdhocEvent.getAdhocDataTransfer()` メソッドにより取得した `AdhocDataTransfer` オブジェクトに対して、`reply()` メソッド以外のメソッドを呼び出すことはできません。また、`Felica.getAdhocDataTransfer()` メソッドにより取得した `AdhocDataTransfer` オブジェクトに対して、`reply()` メソッドを呼び出すことはできません。
- 携帯電話ユーザーは、携帯電話の設定により FeliCa アドホック通信を契機としたサーバー側 i アプリの起動を禁止することができます（典型的な実装では、10.6 項に記載した外部 R/W からの i アプリ起動を禁止する設定と共用の設定になります）。サーバー側 i アプリの起動が禁止されている場合には、FeliCa アドホック通信のリンク確立要求を受信してもサーバー側 i アプリは起動されません。

10.7.3 データ送受信

i アプリでは、FeliCa アドホック通信を利用して OBEX 通信を行うことができます。この OBEX 通信機能は Generic Connection フレームワーク上に構築されており、Connector クラスから以下のコネクションオブジェクトを取得して使用します。

- **FelicaClientObexConnection** : クライアント側 i アプリで利用
- **FelicaServerObexConnection** : サーバー側 i アプリで利用

クライアント側 i アプリにおけるデータ送受信処理

クライアント側 i アプリは、URL 文字列“obex:/felicaclient”を指定して **Connector.open()** メソッドを呼び出すことでコネクションオブジェクト (**FelicaClientObexConnection** オブジェクト) を取得します。**FelicaClientObexConnection** クラスは **ClientObexConnection** インタフェースを実装しています。このコネクションオブジェクトの使用方法は、原則的に赤外線ポートを介した外部接続 (IrOBEX) で使用する **IrClientObexConnection** と同様です。

アプリケーションプログラムは **FelicaClientObexConnection** を使用する際に、まず **connect()** メソッドを呼び出して通信路を開通します。いったん開設された通信路上では、複数のオペレーションを連続して処理することができます。

クライアント側 i アプリでは、オペレーションの処理は以下の手順に従って行います。

1. **setOperation()** メソッドを使用してオペレーション種別を設定します。
2. **ObexConnection** で定義されているアクセスメソッドを使用して、OBEX リクエストヘッダの設定を行います。アクセスメソッドを使用して設定することのできる OBEX リクエストヘッダには、Name ヘッダ、Type ヘッダ、Time ヘッダがあります。
3. PUT オペレーションの場合、送信するオブジェクトボディの内容を書き込みます。この操作は、コネクションオブジェクトの **openOutputStream()** または **openDataOutputStream()** メソッドで取り出した出力ストリームを使用して行います。
4. **sendRequest()** メソッドを使用してオペレーション処理要求をサーバー側 i アプリに送信します。このメソッドはサーバー側 i アプリから応答が返されるまでブロックします。このメソッド呼び出しから復帰すると、必要に応じてサーバー側 i アプリから返却された OBEX レスポンスコードや OBEX レスポンスヘッダの内容を参照することができます。クライアント側 i アプリが参照することのできる OBEX レスポンスヘッダには、Name ヘッダ、Type ヘッダ、Time ヘッダがあります。また、OBEX レスポンスコードは **ObexConnection** で定義されています。**ObexConnection** で定義されている OBEX レスポンスコードは、OBEX 規格におけるレスポンスコードの規定に準拠しています。各コードの意味などの詳細については OBEX 規格を参照してください。サーバー側 i アプリがクライアント側 i アプリの要求を正常に処理できた場合、一般的にはクライアント側 i アプリは、OBEX レスポンスコードとして i アプリ実行環境 (**FelicaClientObexConnection.getResponseCode()** メソッド) から **ObexConnection.SUCCESS** を受け取ります。
5. GET オペレーションの場合、返却されたオブジェクトボディの内容を読み出します。この操作は、コネクションオブジェクトの **openInputStream()** または **openDataInputStream()** メソッドで取り出した入力ストリームを使用して行います。オブジェクトボディの長さは、コネクションオブジェクトの **getContentLength()** メソッドを使用して取得することができます。

必要なオペレーションの処理が全て完了したら、クライアント側 i アプリは以下の手順で FeliCa アドホック通信を終了します。

1. **FelicaClientObexConnection.close()** メソッドにより通信路を切断する。
2. **AdhocDataTransfer.terminateAdhoc()** メソッドにより通信リンクを切断する。
3. **Felica.close()** メソッドにより FeliCa チップをクローズする。もしくは FeliCa アドホック通信のために FeliCa チップが送出している搬送波を、**Felica.turnOffRFPower()** メソッドにより停止する。

クライアント側 i アプリの処理は、全体として上記のシーケンスに合致するようにしてください。例えばオペレーション処理要求をサーバー側 i アプリに送信していない状態で OBEX レスポンスヘッダを取得しようとすると例外が発生します。

注意事項：

- サーバー側 i アプリ（通信リンク確立においてサーバーとして振舞った側の i アプリ）が上記の処理を行うことはできません。すなわち、サーバー側 i アプリは URL 文字列“obex:/felicaclient”を指定して `Connector.open()` メソッドを呼び出すことはできません。
- FeliCa アドホック通信においては、`Connector` クラスから直接 `openInputStream()`、`openDataInputStream()`、`openOutputStream()`、`openDataOutputStream()` の各メソッドにより入出力ストリームを取り出すことはできません。アプリケーションプログラムは `Connector.open()` メソッドで `FelicaClientObexConnection` オブジェクトを取得し、そのオブジェクトから入出力ストリームを取り出す必要があります。
- 1 度のオペレーションで、サーバー側 i アプリにオブジェクトを送信するとともにサーバー側 i アプリからオブジェクトを受信することはできません。クライアント側 i アプリは、PUT オペレーションではオブジェクトの送信、GET オペレーションではオブジェクトの受信のみ行うことができます。
- クライアント側 i アプリにおけるコネクションオブジェクトのオープンモードは、READ モードが GET オペレーションに、WRITE モードが PUT オペレーションに対応します。1 回の接続で GET、PUT の両オペレーションを使用する場合は、READ_WRITE モードを使用するようにしてください。
- Name ヘッダ、Type ヘッダの値にはそれぞれ最大 63 文字まで指定可能です。また、Type ヘッダの値には ASCII 文字のみ使用することができます。
- 1 つの i アプリから同時に開設可能な通信路は 1 つだけです。
- あるスレッドが `connect()` メソッドや `sendRequest()` メソッドを呼び出してサーバー側 i アプリのアクションを待っている時、他のスレッドが `FelicaClientObexConnection` オブジェクトの `close()` メソッドを呼び出すと通信は切断されます。また、タイムアウト発生やユーザーの中断操作といった要因により、通信が切断されることがあります。クライアント側 i アプリでは、オペレーションをサーバー側 i アプリに送信してから 30 秒以内にレスポンスを受け取れない場合はタイムアウトにより通信を切断します。
- GET、PUT の両オペレーションとも、1 回につき 100K バイトまでのオブジェクトを取り扱うことができます。この制限を越えたサイズのオブジェクトを送受信することはできません。
- アプリケーションプログラムが設定したオペレーション種別や OBEX ヘッダ、オブジェクトデータの内容は、1 回のオペレーションでのみ有効です。1 回の接続で連続してオペレーションを実行する場合、その都度これらを設定しなおすようにしてください。
- サーバー側 i アプリから返却されたレスポンスコードが異常を示す値（0x20 番台以外のコード）であった場合、i アプリ実行環境は例外をスローします。

サーバー側 i アプリにおけるデータ送受信処理

サーバー側 i アプリは、URL 文字列“obex:/felicaserver”を指定して `Connector.open()` メソッドを呼び出すことでコネクションオブジェクト（`FelicaServerObexConnection` オブジェクト）を取得します。`FelicaServerObexConnection` クラスは `ServerObexConnection` インタフェースを実装しています。このコネクションオブジェクトの使用方法は、原則的に赤外線ポートを介した外部接続（IrOBEX）で使用する `ServerObexConnection` と同様です。

アプリケーションプログラムは `FelicaServerObexConnection` を使用する際に、まず `accept()` メソッドを呼び出してクライアント側 i アプリからの通信路開設要求（クライアント側 i アプリの `connect()` メソッド実行）を待ちます。いったん開設された通信路上では、複数のオペレーションを連続して処理することができます。

サーバー側 i アプリでは、オペレーションの処理は以下の手順に従って行います。

1. `receiveRequest()` メソッドを使用して要求を受け付けます。このメソッドは、クライアント側 i アプリからの要求を受け付けるまでブロックします。

2. `getOperation()`メソッドを使用して要求のオペレーション種別を取得します。オペレーション種別として `ObexConnection.DISCONNECT` を受け取った場合は、クライアント側 i アプリが通信路の開放を通知したことを示しています。この要求を受け取った場合、サーバー側 i アプリもコネクションオブジェクトをクローズするようにしてください。
3. `ObexConnection` で定義されているアクセスメソッドを使用して、OBEX リクエストヘッダの内容を取得します。アクセスメソッドを使用して取得することのできる OBEX リクエストヘッダには Name ヘッダ、Type ヘッダ、Time ヘッダがあります。
4. PUT オペレーションの場合、クライアント側 i アプリから送信されたオブジェクトボディの内容を読み出します。この操作は、コネクションオブジェクトの `openInputStream()` または `openDataInputStream()` メソッドで取り出した入力ストリームを使用して行います。オブジェクトボディの長さは、コネクションオブジェクトの `getLength()` メソッドを使用して取得することができます。
5. `ObexConnection` で定義されているアクセスメソッドを使用して、OBEX レスポンスヘッダの内容を設定します。アクセスメソッドを使用して設定することのできる OBEX レスポンスヘッダには Name ヘッダ、Type ヘッダ、Time ヘッダがあります。
6. GET オペレーションの場合、クライアント側 i アプリが要求しているオブジェクトのオブジェクトボディを書き込みます。この操作は、コネクションオブジェクトの `openOutputStream()` または `openDataOutputStream()` メソッドで取り出した出力ストリームを使用して行います。
7. `sendResponse()` メソッドを使用して、クライアント側 i アプリにレスポンスを返します。このメソッドの引数には、`ObexConnection` で定義されている OBEX レスポンスコードを指定します。なお、`ObexConnection` で定義されている OBEX レスポンスコードは、OBEX 規格におけるレスポンスコードの規定に準拠しています。各コードの意味などの詳細については OBEX 規格を参照してください。サーバー側 i アプリがクライアント側 i アプリの要求を正常に処理できた場合、一般的にはサーバー側アプリケーションは、OBEX レスポンスコードとして i アプリ実行環境 (`FelicaServerObexConnection.sendResponse()` メソッド) に `ObexConnection.SUCCESS` を返します。

クライアント側 i アプリが `FelicaClientObexConnection.close()` メソッドを使用して通信路を切断する際、サーバー側 i アプリには DISCONNECT オペレーション (`ObexConnection.DISCONNECT`) が送信されます。サーバー側 i アプリは DISCONNECT オペレーションを受け取ったら、`FelicaServerObexConnection.close()` メソッドを使用して通信を終了してください。

サーバー側 i アプリの処理は、全体として上記のシーケンスに合致するようにしてください。例えばオペレーション処理要求を受け付けていない状態で OBEX リクエストヘッダを取得しようとすると例外が発生します。

注意事項：

- クライアント側 i アプリ（通信リンク確立においてクライアントとして振舞った側の i アプリ）が上記の処理を行うことはできません。すなわち、クライアント側 i アプリは URL 文字列 `"obex://felicaserver"` を指定して `Connector.open()` メソッドを呼び出すことはできません。
- FeliCa アドホック通信においては、`Connector` クラスから直接 `openInputStream()`、`openDataInputStream()`、`openOutputStream()`、`openDataOutputStream()` の各メソッドにより入出力ストリームを取り出すことはできません。アプリケーションプログラムは `Connector.open()` メソッドで `FelicaServerObexConnection` オブジェクトを取得し、そのオブジェクトから入出力ストリームを取り出す必要があります。
- クライアント側 i アプリからの PUT オペレーションに対して、サーバー側 i アプリがオブジェクトを送信することはできません。サーバー側 i アプリは GET オペレーションに対してのみオブジェクトを送信することができます。
- サーバー側 i アプリにおけるコネクションオブジェクトのオープンモードは、READ モードが PUT オペレーションに、WRITE モードが GET オペレーションに対応します。1 回の接続で GET、PUT の両オペレーションを受け付ける可能性がある場合は READ_WRITE モードを使用するようにしてください。
- Name ヘッダ、Type ヘッダの値にはそれぞれ最大 63 文字まで指定可能です。また、Type ヘッダの値には ASCII 文字のみ使用することができます。
- あるスレッドが `accept()` メソッドや `receiveRequest()` メソッドを呼び出してクライアント側 i アプリのアクションを待っている時、他のスレッドが `FelicaServerObexConnection` オブジェクトの

`close()`メソッドを呼び出すと通信は切断されます。また、タイムアウト発生やユーザーの中断操作といった要因により、通信が切断されることがあります。サーバー側 i アプリでは、オペレーション受信待ちとなってから 60 秒以内にオペレーションを受け取れない場合はタイムアウトにより通信を切断します。

- GET、PUT の両オペレーションとも、1 回につき 100K バイトまでのオブジェクトを送受信することができます。この制限を越えたサイズのオブジェクトを送受信することはできません。
- アプリケーションプログラムが設定した OBEX ヘッダやオブジェクトデータの内容は、1 回のオペレーションでのみ有効です。1 回の接続で連続してオペレーションを実行する場合、その都度これらを設定しなおすようにしてください。

第 11 章

UART 接続によるシリアル通信

本章では、FOMA 携帯電話の外部接続端子と外部機器を接続ケーブルで接続 (UART 接続) し、シリアル通信を行う方法について解説します。

UART 接続によるシリアル通信の機能は、Generic Connection フレームワーク上で提供されます。UART 接続でシリアル通信を行うには、接続先として以下の形式の URL を指定して `Connector.open()` メソッドを呼び出します。

```
comm:/0 [ ; <パラメータ名>=<パラメータ値>, ... ]
```

オプションとして指定可能なパラメータには以下のようなものがあります。

パラメータ名	意味およびパラメータ値
<code>baudrate</code>	ボーレートを <code>baudrate=<値></code> の形式で指定します。共通的にサポートされる値には、 9600 、 38400 、 115200 (いずれも単位は bps) の 3 つがあります。指定がない場合、 9600 が指定されたものとして動作します。
<code>databitlen</code>	データビット長を <code>databitlen=<値></code> の形式で指定します。共通的にサポートされる値は 8 のみであり、指定がない場合も 8 が指定されたものとして動作します。
<code>parity</code>	パリティの設定方法を <code>parity=<識別子></code> の形式で指定します。識別子には以下のものが規定されていますが、共通的にサポートされるものは NO (パリティなし) のみであり、指定がない場合も NO が指定されたものとして動作します。 NO : パリティなしを示します。 ODD : 奇数パリティを使用することを示します。 EVEN : 偶数パリティを使用することを示します。 MARK : 常にパリティビットとして 1 を付加することを示します。 SPACE : 常にパリティビットとして 0 を付加することを示します。
<code>stopbitlen</code>	ストップビット長を <code>stopbitlen=<値></code> の形式で指定します。値としては以下のものが規定されていますが、共通的にサポートされるものは 1 のみであり、指定がない場合も 1 が指定されたものとして動作します。 1 : ストップビット長が 1 であることを示します。 1.5 : ストップビット長が 1.5 であることを示します。 2 : ストップビット長が 2 であることを示します。
<code>flowctrl</code>	フローコントロールの指定を <code>flowctrl=<識別子></code> の形式で指定します。値としては以下のものが規定されていますが、共通的にサポートされるものは OFF のみであり、指定がない場合も OFF が指定されたものとして動作します。 ON : フロー制御 (XON/XOFF、RTR、RTC) を行なうことを示します。 OFF : フロー制御 (XON/XOFF、RTR、RTC) を行なわないことを示します。

以下に、UART 接続でシリアル通信を行う際の URL の記述例を示します。

```
comm:/0;baudrate=38400
```

`Connector.open()` メソッドに上記のような URL を与えると、このメソッドは UART 接続上にシリアル通信路を開設し、`StreamConnection` インタフェースを実装したコネクションオ

ブジェクトを返します。アプリケーションプログラムは、このコネクションオブジェクトから取り出した入出力ストリームを使用してシリアル通信を行います。

Connector.open() メソッドにオープンモードとして **Connector.READ** または **Connector.WRITE** を指定した場合は、片方向のシリアル通信を行うことを意味します。双方向のシリアル通信を行う場合には、オープンモードとして **Connector.READ_WRITE** を指定するようにしてください。**Connector.READ_WRITE** を指定した場合は、コネクションオブジェクトから入力ストリームと出力ストリームの両方を各々1 つずつまで取得することができます。

注意事項：

- 1 つの i アプリから同時に開設可能な通信路は 1 つだけです。
- UART 接続を使用している状態で、i アプリがサスペンドすると、UART 接続は中断されます。その場合は、ストリームやコネクションオブジェクトをクローズした上で再度 UART 接続を行うようにしてください。
- UART 接続の機能と FeliCa チップの機能は、1 つのアプリにおいて排他的にのみ使用可能です。ADF やトラステッド i アプリのパーミッションで FeliCa チップの機能を使用することが宣言されている i アプリや、アドホック通信用に FeliCa チップをオープンした状態の i アプリは、UART 機能を使用することはできません。
- UART 接続のシリアル通信で 1 度に送受可能なデータサイズの最大値（ネイティブのシリアル通信部の送受信バッファサイズなど）は、メーカーにより異なります。
- 本機能を利用して携帯電話（携帯電話上の i アプリ）と外部機器を接続する場合、NTT ドコモが販売している FOMA USB 接続ケーブルではなく、特殊な仕様を搭載した接続ケーブルが必要となる場合があります。この接続ケーブルの開発を行うための技術仕様については、別途 NTT ドコモより公開される予定です。

第 12 章

外部メモリ

携帯電話の中には、内蔵カメラで撮影した画像などを保存する用途で外部メモリインタフェースを備えているものがあります。本章では、外部メモリを i アプリのデータストレージとして使用するための API について解説します。外部メモリを使用することにより、i アプリで ScratchPad の容量制限を超える大容量データを取り扱うことができるようになります。また、着脱可能であることを利用して、携帯電話個体間でデータを流通させるための手段として利用することもできます。

本 API で取り扱う外部メモリは著作権保護機能付きの外部メモリを前提としており、i アプリ開発者の指定によって、保存データに対するアクセス権のコントロールを行うことができます。この機能を使用することで、あるユーザーの携帯電話上で保存されたデータを他のユーザー（異なる UIM が挿入された携帯電話）が使用するといったことを制限したり、特定のいくつかの i アプリに外部メモリ上のデータを共有させるような制御を行うことができます。

注意事項：

- i アプリと並列に起動された他のネイティブアプリケーション（音楽プレーヤーなど）が外部メモリを排他的に占有利用している状況では、本章の機能を i アプリから利用できない場合があります。

12.1 外部メモリ API の API 構成

この項では外部メモリ API のクラス、インタフェースの構成について解説します。外部メモリのメディアにはいくつかの規格が存在しますが、i アプリの外部メモリ API ではメディアの種別を意識することなく使用できるように、メディアの種別に左右されない汎用的な API として設計されています。ただしメディア種別に固有の能力を細かく制御する必要がある場合には、メディア種別ごとに拡張されたクラスを使用してそのような制御も可能となるような構成としています。

外部メモリ API は、以下のパッケージに属するクラス、インタフェースから構成されます。

com.docomostar.device パッケージ

外部メモリインタフェース（外部メモリのスロットデバイス）に関する機能は、com.docomostar.device パッケージで提供されます。

com.docomostar.fs パッケージ

com.docomostar.fs パッケージでは、ファイルシステムに関する機能をメディアの種別によらない形で取り扱うためのクラスを規定しています。

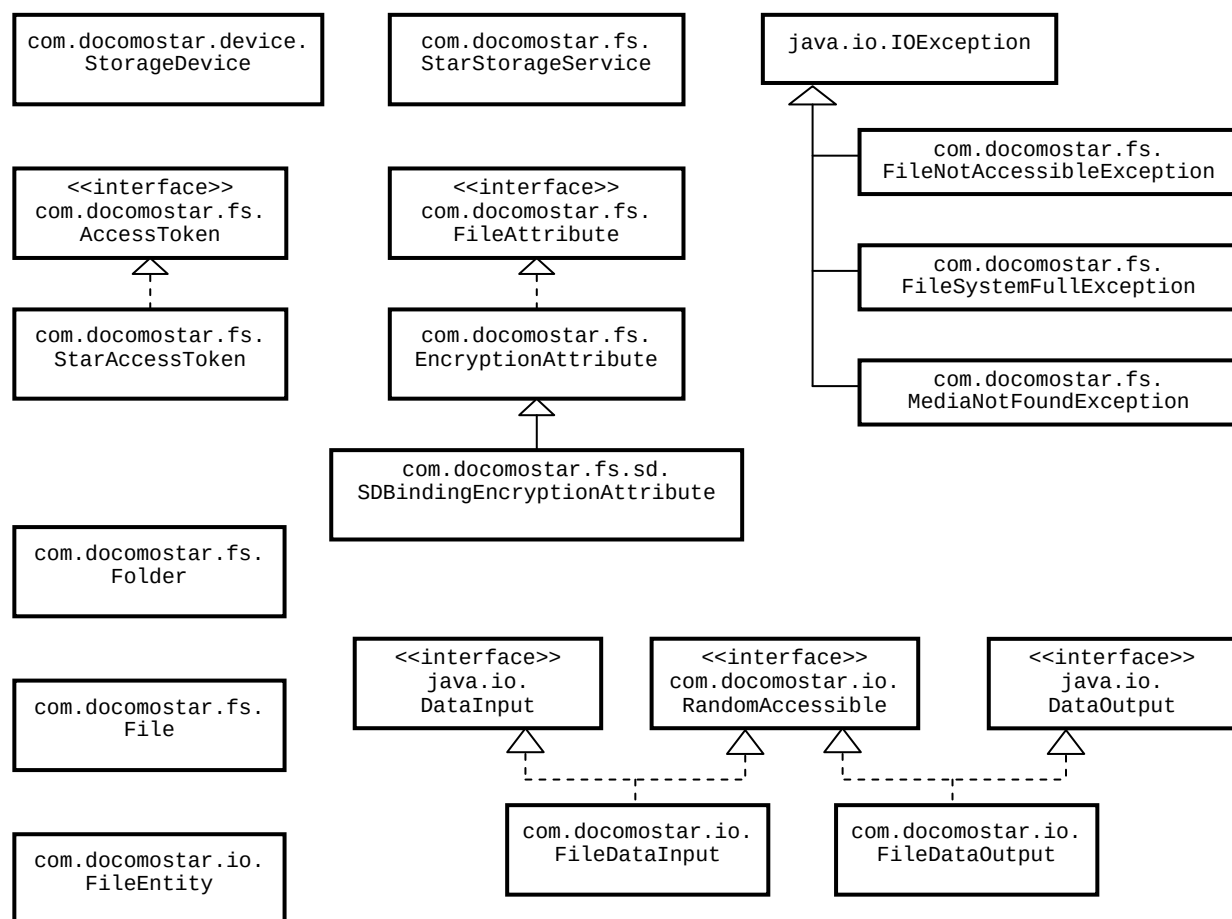
com.docomostar.fs 配下サブパッケージ

ファイルシステムに関する機能について、メディアの種別に固有の能力をメディアに依存した形で取り扱うためのクラスを規定しています。このパッケージはメディアの種別ごとに存在し、現在のところ SD メモリカード（miniSD メモリカードを含む）に対応する com.docomostar.fs.sd パッケージが規定されています。

com.docomostar.io パッケージ

com.docomostar.io パッケージでは、ファイルシステム上に配置されたファイル実体に対して、実際にデータの入出力操作を行う際に必要な機能を規定しています。

以下に、外部メモリ API のクラス・インタフェース構成を図示します。



各クラス、インタフェースの機能概要は以下の通りです。

com.docomostar.device パッケージ

クラス	機能概要
StorageDevice	外部メモリのスロットデバイスを表現するクラスです。デバイスや、デバイスに挿入されている外部メモリメディアに関する情報を取得したり、データアクセスの基点となるフォルダの取得を行う機能を提供しています。

com.docomostar.fs パッケージ

クラス、インタフェース	機能概要
Folder	フォルダを表現するクラスです。外部メモリ API では、i アプリが指定するアクセス権限の組み合わせ毎に異なるフォルダが使用されます。StorageDevice クラスのフォルダ取得メソッド (getFolder() メソッド) にどのようなアクセス権を指定するかにより、得られるフォルダが決定付けられます。なお、現在のプロファイルの外部メモリ API では、フォルダ配下にさらにサブフォルダを生成、使用することはできません。
AccessToken	アクセス権を表すクラスが実装するインタフェースです。アクセス権を表すクラスはシステムにより提供されますが、それらはこのインタフェースを実装しています。

StarAccessToken	i アプリから、外部メモリのメディア種別に依存しない形でアクセス権を表現するクラスです。アクセス権を表現するため、上記の AccessToken インタフェースを実装しています。i アプリは、このクラスのオブジェクトを StarStorageService クラスのファクトリメソッドを使用して取得し、フォルダ取得メソッドの引数に指定します。
StarStorageService	外部メモリのメディア種別に依存しない、汎用のストレージデバイスサービスを定義するクラスです。現在のプロファイルでは、このクラスに定義されたメソッドは StarAccessToken オブジェクトのファクトリメソッドのみです。
File	ファイルを表現するクラスです。i アプリは Folder オブジェクトのメソッドを使用して、フォルダ内に新規にファイルを生成したり、フォルダに存在しているファイルを取得することができます。いずれの場合も、ファイルは File クラスのオブジェクトとして表されます。実際にファイル内のデータにアクセスするには、このクラスのオブジェクトから com.docomostar.io.FileEntity クラスのオブジェクトを取り出して使用します。
FileAttribute	ファイル属性を表すクラスが実装するインタフェースです。ファイルを生成する際には、オプションとしてファイル属性を指定することができます。ファイル属性はその種別ごとに対応するクラスがシステムにより提供されますが、それらのクラスは本インタフェースを実装しています。現在のプロファイルでは、ファイル属性として暗号化パラメータが規定されています。
EncryptionAttribute	ファイル属性の1つで、暗号化パラメータの属性を表すクラスです。このクラスは、外部メモリのメディア種別に依存しない形での暗号化パラメータ制御を実現するため、暗号化のオン、オフの制御のみ行うことができます（暗号化をオンにした際のデフォルトの暗号化パラメータはメーカーにより異なります）。メディアに依存した暗号化能力を詳細に制御するには、メディア種別ごとに規定されたクラスを使用します。
FileNotAccessibleException	ファイルが使用中であったり、存在しないファイルの読み書きやすでに存在している名前のファイルの作成を指示したなどの理由で、ファイルにアクセスできなかったことを示す例外です。
FileSystemFullException	ファイルシステムやフォルダに空き容量がないために出力に失敗したことを示す例外です。
MediaNotFoundException	使用中のメディアが抜かれたり（挿し直した場合も含む）、利用不能になった場合に発生する例外です。

com.docomostar.fs.sd パッケージ

クラス	機能概要
SDBindingEncryptionAttribute	SD メモリカード向けに、暗号化パラメータの属性を指定するためのクラスです。このクラスは com.docomostar.fs.EncryptionAttribute クラスを継承しています。本クラスを使用することで、暗号化のオン、オフの制御の他に暗号化ブロックサイズを指定することができます。

com.docomostar.io パッケージ

クラス、インタフェース	機能概要
FileEntity	ファイルの実体を表現するクラスです。ファイル内のデータにアクセスするには、まず File オブジェクトの open() メソッドを呼び出して本クラスのオブジェクトを取得します。このクラスのオブジェクトから、ファイル内のデータにアクセスするためのストリームやファイルデータ入出力オブジェクトを取得することができます。
RandomAccessible	ランダムアクセス可能であることを示すインタフェースです。データ中のアクセス位置を制御するためのメソッドが定義されています。
FileDataInput	ファイルデータ入力クラスです。このクラスは RandomAccessible インタフェースと java.io.DataInput インタフェースを実装しており、ファイル中の任意の位置のデータを参照することが可能です。
FileDataOutput	ファイルデータ出力クラスです。このクラスは RandomAccessible インタフェースと java.io.DataOutput インタフェースを実装しており、ファイル中の任意の位置にデータを書き込むことが可能です。

次に、これらのクラス、インタフェースを使用して外部メモリへのアクセスを行う方法について解説します。

12.2 外部メモリ API の利用

前項で説明したクラス、インタフェースを使用して実際に外部メモリとの間でデータ入出力を行うための、典型的な流れは以下のようになります。

- 1) アプリケーションプログラムはまず **StorageDevice** オブジェクトを取得する。次に **StarStorageService** クラスのファクトリメソッドで目的に適合したアクセス権オブジェクト (**StarAccessToken** オブジェクト) を生成し、**StorageDevice** オブジェクトのフォルダ取得メソッドを呼び出すことでアクセス権に合致したフォルダ (**Folder** オブジェクト) を取得する。
- 2) **Folder** オブジェクトのメソッドを使用して、そのフォルダにファイルを作成 (ファイル属性を指定することも可能) したり、フォルダ内に存在するファイルを取得する。いずれの場合も、ファイルは **File** オブジェクトとして i アプリに返される。
- 3) **File** オブジェクトのメソッドにオープンモードを指定してファイルをオープンし、そのファイルの実体 (**FileEntity** オブジェクト) を取得する。**FileEntity** オブジェクトからはファイル内のデータにアクセスするためのストリームやファイルデータ入出力オブジェクトを取得することができるので、それらを使用してデータアクセスを行う。ファイルへのアクセスが終わったら、**FileEntity** オブジェクトをクローズする。

以下に、それぞれのフェーズについての詳細を記載します。

12.2.1 StorageDevice クラスの利用

StorageDevice クラスのオブジェクトは、static メソッド **getInstance()** に対し、引数にデバイス名を指定することで取得します。現在のプロファイルでは、デバイス名は **"/ext0"** のみが規定されています。

StorageDevice オブジェクトの **getFolder()** メソッドに、目的のアクセス権に適合したアクセス権オブジェクト (**AccessToken** インタフェース実装オブジェクト) を指定することで、そのアクセス権に合致したフォルダの **Folder** オブジェクトを取得することができます。現在のプロファイルでは、アクセス権を表すクラスとして **StarAccessToken** クラスが規定されています。**StarAccessToken** クラスは、メディア種別に依存しない形でアクセス権を表現するクラスです。このクラスのオブジェクトは、**StarStorageService** クラスの static メソッド **getAccessToken()** にアクセス識別子と共有識別子の 2 つのパラメータを指定して取得します。

アクセス識別子：

アクセス識別子とは、外部メモリに保存されたデータが、保存された際の機器構成とは異なる機器の上でも利用可能とするかどうかを示す識別子です。アクセス識別子は、**StarAccessToken** クラスに以下の定数として定義されています。

- **ACCESS_UIM**： データ保存時と同じ UIM が挿入された携帯電話の場合のみアクセス可能とします。
- **ACCESS_PLATFORM**： データ保存時と同じ機種の場合のみアクセス可能とします。

アプリケーションプログラムは、上記を任意の組み合わせの論理和として使用することができます。また、値 **0** を指定した場合は上記のようなアクセス制御を行わないことを意味します。

注意事項：

- **StarAccessToken** クラスには、アクセス識別子を表す定数として **ACCESS_SERIES** が定義されています。しかし、現在のプロファイルでは **ACCESS_SERIES** はサポートされません。現在のプロファイルでは、**ACCESS_SERIES** を指定しても無視されます。

共有識別子：

共有識別子とは、外部メモリに保存されたデータへのアクセスを、それを保存した i アプリ以外の i アプリにも許容するかどうかを示す識別子です。共有識別子は **StarStorageService** クラスに以下の定数として定義されています。

- **SHARE_APPLICATION**: データを保存した i アプリのみが、そのデータを使用することができます。
- **SHARE_CONTENTS_PROVIDER**: 同じコンテンツプロバイダが提供する複数の i アプリで、そのデータを共有することができます。この場合、ADF にコンテンツプロバイダの識別情報を設定する必要があります（後述）。

アプリケーションプログラムは、共有識別子として上記のいずれかの値を使用します。

注意事項:

- 共有識別子に **SHARE_CONTENTS_PROVIDER** を使用する i アプリでは、ADF にデータ共有のための設定を行なう必要があります。具体的な設定内容については 12.4 項を参照してください。

以下に、**StorageDevice** クラスのオブジェクトの取得から目的とするフォルダを取得するまでのコード例を示します。

例: **StorageDevice** クラスの利用

```
try {
    // StorageDevice オブジェクトを取得する
    StorageDevice sd = StorageDevice.getInstance("/ext0");
    // データ保存時と同一 UIM、同一機種の携帯電話でアクセス可能となるアクセス識別子を用意する
    int access = StarAccessToken.ACCESS_UIM | StarAccessToken.ACCESS_PLATFORM;
    // データを保存した i アプリのみそのデータを使用可能となる共有識別子を用意する
    int share = StarStorageService.SHARE_APPLICATION;
    // アクセス識別子、共有識別子に合致したアクセス権オブジェクトを取得する
    StarAccessToken sat = StarStorageService.getAccessToken( access, share );
    // アクセス権オブジェクトを指定して、対応するフォルダの Folder オブジェクトを取得する
    Folder folder = sd.getFolder( sat );
    ... // 以降、フォルダ内に新規にファイルを作成したり、フォルダ内に
        // 存在しているファイルへのデータ入出力処理を行う
} catch (IOException ioe) {
    ... // StorageDevice.getFolder()は IOException をスローする可能性がある
}
```

StorageDevice クラスにはフォルダを取得する以外に、以下のような機能を持つメソッドが用意されています。

外部メモリのスロットデバイスに関する情報の取得:

- getDeviceName()**: デバイス名 (**getInstance()** に指定した名前) を取得します。
- getPrintName()**: デバイスの表示名を取得します。表示名とは、ユーザーに理解可能なデバイスの一般名称（例えば **"miniSD"** など）のことです。
- isRemovable()**: メディアが取り出し可能かどうかを調べます。一般的には、外部メモリメディアは取り出し可能です。
- isAccessible()**: スロット内のメディアにアクセス可能かどうかを調べます。

スロットデバイスに挿入されたメディアに関する情報の取得:

- isWritable()**: メディアが書き込み可能かどうかを調べます。
- isReadable()**: メディアが読み込み可能かどうかを調べます。
- getMediaId()**: メディアの個体ごとにユニークな識別子を取得します。
- getCapability()**: メディアの能力を調べます。調べることができるメディアの能力には、ハードウェア能力 (**CATEGORY_HARDWARE**)、暗号化能力 (**CATEGORY_ENCRYPTION**)、ファイルシステム能力 (**CATEGORY_FILESYSTEM**) があります。いずれかの種別を指定してこのメソッドを呼び出すと、対応するメディアの能力 (**CAPABILITY** 定数) を取得することができます。

12.2.2 フォルダの操作

StorageDevice.getFolder() メソッドでフォルダオブジェクトを取得すると、そのフォルダに新規にファイルを作成したり、既存のファイルにアクセスすることができます。いずれの場合にも、ファイルは **File** オブジェクトとして表現されます。

ファイルの新規作成：

ファイルを新規に作成するには、**Folder.createFile()** メソッドを使用します。このメソッドの引数にファイル名（パス名は含みません）を指定して呼び出すと、フォルダ配下にその名前でファイルが作成され、そのファイルに対応する **File** オブジェクトが返ります。

実装によっては、ファイルの新規作成時にファイル属性（**FileAttribute** インタフェース実装オブジェクト）を指定できるものがあります。ファイル属性は個々の種別ごとにクラスが用意されており、現在のプロファイルでは、ファイル属性を表現するクラスとして以下の 2 つが規定されています。

- ・ 暗号化のオン、オフを指定する **EncryptionAttribute** クラス
- ・ SD メモリカード向けに、詳細な暗号化パラメータを指定する **SDBindingEncryptionAttribute** クラス（**EncryptionAttribute** のサブクラス）

そのフォルダでファイル属性がサポートされているかどうかは、**Folder.isFileAttributeSupported()** メソッドを使用することで調べることができます。このメソッドの引数に調べたいファイル属性クラスの **Class** オブジェクトを指定すると、そのファイル属性がサポートされているかどうか **boolean** 値として返ります。

既存ファイルへのアクセス：

既存ファイルにアクセスする方法には、以下の 2 つがあります。

- ・ 特定のファイルの名前を **Folder.getFile()** メソッドの引数に指定し、そのファイルに対応する **File** オブジェクトを取得する方法
- ・ **Folder.GetFiles()** メソッドにより、そのフォルダ配下の全てのファイルの **File** オブジェクトを配列として取得する方法

Folder クラスや **File** クラスはフォルダやファイルの、ファイルシステム上のエントリに相当するものです。上記手順により取得した **File** オブジェクトを使用してファイル内のデータにアクセスするには、**File** オブジェクトからさらに **FileEntity** オブジェクト（ファイル実体を表現するオブジェクト）を取得して使用します（詳細は次項で解説します）。

なお、現在のプロファイルではサブフォルダの作成や利用はサポートされません。

Folder クラスには、ファイルに関する上記の機能以外に、以下のようなフォルダ情報取得メソッドが用意されています。

getAccessToken()	： このフォルダへのアクセス権 (StorageDevice.getFolder() に指定したアクセス権と同等の内容) を取得します。
getFreeSize()	： このフォルダの空き容量を取得します。フォルダ毎に容量を割り当てる機能を持たないファイルシステム上では、どのフォルダでも同じ値（ファイルシステム全体の空き容量）が返ります。
getPath()	： このフォルダのパス名を取得します。パスの区切り文字は"/"です。
getStorageDevice()	： このフォルダの取得元となったデバイスの StorageDevice オブジェクトを取得します。

注意事項：

- ファイルを新規に作成した後、データを 1 バイトも書き込まないままそのファイルを保存することはできません。ファイルを新規に作成した後に、データを 1 バイトも書き込まずにそのファイルへのアクセスを終了した場合、そのファイルは削除されます。

- **Folder.GetFiles()**メソッドが返す **File** オブジェクトの配列上の順序は、各メーカーの OS やファイルシステムの実装に強く依存します。これらの順序はファイル名称順や書き込んだ順など、特定の順序に従って並んでいる保証はありません。特定の順序に基づいてファイル进行处理する必要がある場合は **Folder.GetFiles()**メソッドではなく、**Folder.getFile()**メソッドに個々のファイル名を指定してアクセスするようにしてください。

12.2.3 ファイルの操作と入出力

Folder オブジェクトから取り出した **File** オブジェクトはファイルシステム上のファイルエントリに相当する機能を持つオブジェクトであり、ファイルのパス名やサイズ、更新日時などのファイル管理情報を取り扱います。**File** クラスには、ファイル管理情報に関するメソッドとして以下のようなものが用意されています。

delete()	: このファイルを削除します。
getAccessToken()	: このファイルへのアクセス権 (StorageDevice.getFolder() に指定したアクセス権と同等の内容) を取得します。
getFolder()	: このファイルの取得元となったフォルダの Folder オブジェクトを取得します。
getLastModified()	: このファイルの更新日時を取得します。
getLength()	: このファイルのサイズを取得します。
getPath()	: このファイルのパス名を取得します。パスの区切り文字は"/"です。

また、ファイル内のデータにアクセスするには **File.open()**メソッドを呼び出して **FileEntity** オブジェクトを取得する必要があります。**File** オブジェクトがファイルシステム上のファイルエントリを表現するのに対し、**FileEntity** オブジェクトはファイル実体を表現します。

File.open()メソッドの引数にオープンモード (**MODE_READ_WRITE**、**MODE_READ_ONLY**、**MODE_WRITE_ONLY**) を指定して呼び出すと、そのオープンモードでファイルがオープンされて **FileEntity** オブジェクトが返ります。

FileEntity オブジェクトを使用してファイル内のデータにアクセスするには、以下の 2 種類の方法があります。

ストリームを使用したファイルアクセス :

FileEntity.openInputStream()や **FileEntity.openOutputStream()**により、ファイルに対応付けられた入出力ストリームを取得して使用します。ファイル内のデータに、先頭からシーケンシャルにアクセスする機能を提供します。Java 言語標準ライブラリの **java.io** パッケージにも取り入れられている一般的なデータアクセス方法であり、ストリームを引数に取る他の API (例えば、メディアデータの生成など) とも親和性が高い方法です。

データ入出力オブジェクトを使用したファイルアクセス :

FileEntity.openDataInput()や **FileEntity.openDataOutput()**により、ファイルに対応付けられたデータ入出力オブジェクトを取得して使用します。ファイル内の任意の位置にあるプリミティブ型データや文字列にアクセスする機能を提供します。

この方法では、データの入出力に i アプリ独自のクラスである **FileDataInput** クラスと **FileDataOutput** クラスを使用します。これらのクラスは、プリミティブ型データ等を読み書きするためのインタフェースである **DataInput** や **DataOutput** に加え、ランダムアクセスを可能とするための **RandomAccessible** インタフェース (i アプリ独自インタフェース) を実装しています。データ入出力オブジェクトを使用してのファイルアクセスでは、**RandomAccessible** インタフェースで規定されている **setPosition()**メソッドや **setPositionRelative()**メソッドによってファイル内の位置を設定した後に、リードメソッドやライトメソッドを使用してデータの入出力を行います。

ストリームを使用したファイルアクセスとデータ入出力オブジェクトを使用したファイルアクセスを、同一ファイルに対して同時に使用することはできません。これらのファイルアクセス方法は排他的にのみ使用可能です。

FileEntity クラスには、ファイルアクセスで使用する内部バッファのサイズを設定、確認するためのメソッド (**setBufferSize()**、**getBufferSize()**) が規定されています。ただし、システムの実装上の制限により、バッファサイズの設定を行ってもそのサイズとは異なるサイズのバッファが使用される場合があります。

ファイルのアクセスが終了したら、**FileEntity.close()**メソッドを呼び出します。このメソッドを呼び出すことで、一般的なファイルクローズ処理が行われます。

なお、外部メモリへのアクセスは、デバイス自身のアクセス性能や暗号化処理のオーバーヘッドなどの要因により、**ScratchPad** アクセスと比較して一般に低速です。また、ファイルのオープン時にもオーバーヘッドが生じるため、アクセスするファイル数が多いほど処理に時間を要します。リアルタイムな処理が必要な箇所では外部メモリへのアクセスは極力抑え、時間を要しても問題がない箇所で一括して外部メモリからデータを読み込むようなアプリケーション処理構成とすることを推奨します。

注意事項：

- 一般的に、外部メモリの暗号化機能を使用する場合は、暗号化機能を使用しない場合と比較してファイルアクセス性能は低下します。i アプリはファイルの生成時に **EncryptionAttribute** を使用することで暗号化機能のオン、オフを指定することができますが、SD カードを使用する場合には **EncryptionAttribute** クラスの継承クラスである **SDBindingEncryptionAttribute** クラスを使用することで、暗号化の際のブロックサイズも指定することができます（デフォルトのブロックサイズは 64 バイト）。実際にファイルに書き込む内容より暗号化ブロックサイズの方が大きい場合は、余分な暗号処理が実行されオーバーヘッドがさらに大きくなりますので、その場合は暗号化ブロックサイズを適切な値に設定するようにしてください（暗号化ブロックサイズとして設定可能な値については API リファレンスを参照してください）。

12.3 外部メモリ API に固有の例外

外部メモリ API では、取り外し可能な記憶メディアを取り扱う上で発生するエラーを表現するために、以下の 3 つの例外を追加で規定しています。これらの例外は、いずれも **IOException** のサブクラスです。

FileNotAccessibleException

処理対象のファイルにアクセスできないことを示す例外です。存在しないファイルにアクセスしようとした場合やすでに存在する名前のファイルを新規作成しようとした場合、排他制御などによりアクセスが拒否された場合などにこの例外が発生します。例外が発生した大まかな理由を、**getStatus()**メソッドにより知ることができます。

FileSystemFullException

ファイルシステムやフォルダに空き容量がないため出力に失敗したことを示す例外です。

MediaNotFoundException

使用中のメディアが抜かれたり、利用不能になったことを示す例外です。外部メモリを使用中にメディアを抜くと、例えメディアを挿入しなおしたとしても、その後のアクセスではこの例外が発生します。その場合は、**StorageDevice** オブジェクトから再度 **Folder** オブジェクトを取得し直す必要があります。

12.4 外部メモリを利用するための ADF の設定

外部メモリを使用する i アプリは、ADF の **UseStorage** キーにより、自身が外部メモリを使用することを宣言する必要があります。また、複数の i アプリに外部メモリ上のデータを共有させる場合には、ADF の **CPBindingAuth** キーおよび **PackageHashCode** キーにてデータ共有のための識別情報を設定する必要があります。

本項では、外部メモリを利用するために必要となる ADF の設定内容について解説します。なお、以下の設定項目のうち **CPBindingAuth** キーと **PackageHashCode** キーについては値の計算など設定手順が煩雑であるため、NTT ドコモの提供する i アプリ開発環境にはこれらのキーの値を設定するためのツールが含まれます。

UseStorage キー（外部メモリを使用する i アプリ全てで必須）

外部メモリを使用する全ての i アプリは、ADF の **UseStorage** キーにより、自身が外部メモリを使用する i アプリであることを宣言する必要があります。値に **"ext"** が指定されている場合、その i アプリが外部メモリを使用することを示します。

i アプリからの外部メモリ利用をサポートする機種であっても、外部メモリメディアが挿入されていないなどの理由により、必ずしも外部メモリが使用できるとは限りません。このキーが設定された i アプリのダウンロード時には、その i アプリが外部メモリを使用するものであることをユーザーに知らせるメッセージが表示されます。ただし、外部メモリが使用できない状態であっても、本キーが宣言された i アプリをダウンロードしたり起動することは可能です。

CPBindingAuth キー（他の i アプリと外部メモリのデータを共有する非トラステッド i アプリでは必須）

アクセス権オブジェクト (**StarAccessToken** オブジェクト) に対し、共有識別子として **SHARE_APPLICATION** を指定した場合には、外部メモリのデータはそれを保存した i アプリ（ADF の URL が同一の i アプリ）のみがアクセスすることができます。これに対し、共有識別子として **SHARE_CONTENTS_PROVIDER** を指定した場合には、同一コンテンツプロバイダが提供する複数の i アプリの間で、外部メモリのデータを共有することができます。

外部メモリのデータを複数の i アプリで共有する場合、すなわち共有識別子として

SHARE_CONTENTS_PROVIDER を指定する場合には、ADF の **CPBindingAuth** キーにコンテンツプロバイダを特定するための識別情報を設定する必要があります。**CPBindingAuth** キーには、ADF の以下のキーのパラメータ値を連結したものにデジタル署名を施し（PKCS#7 SignedData 形式）、その内容を Base64 エンコードしたものを設定します。

- **AppName** キー
- **PackageURL** キー
- **AppSize** キー
- **AppClass** キー
- **LastModified** キー
- **PackageHashCode** キー（後述）

LastModified キーや **AppSize** キーなどの内容が含まれるため、本キーは i アプリをビルドする度に再設定を行う必要があります（i アプリをビルドした後に、ツールを利用して設定します）。

本キーの設定を行う際には、デジタル署名を行うための秘密鍵が必要となります。この秘密鍵とペアになる証明書は、携帯電話上で検証可能なものでなければなりません。携帯電話上で検証可能な証明書は、「i アプリコンテンツ開発ガイド 詳細編」5.3 項に記載されたルート認証局に対応したもののみです。

なお、外部メモリを使用する i アプリがトラステッド i アプリの場合には、コンテンツプロバイダの識別にはトラステッド i アプリの ID が使用されます。従ってトラステッド i アプリでは、共有識別子に **SHARE_CONTENTS_PROVIDER** を指定する場合でも本キーを設定する必要はありません。

また、トラステッド i アプリと一般 i アプリで、外部メモリ上のデータを共有することはできません。

注意事項：

- このキーの設定（デジタル署名処理）に使用する証明書は、ルート証明書からコンテンツプロバイダの証明書までの証明書チェーンが 4 段以内でなければなりません。すなわち、ルート証明書とコンテンツ

プロバイダの証明書の間に中間 CA 証明書などがチェーンされている場合は、それらは 2 つ以下である必要があります。

PackageHashCode キー（他の i アプリと外部メモリのデータを共有する非トラステッド i アプリでは必須）

共有識別子として `SHARE_CONTENTS_PROVIDER` を指定する i アプリにおいて、`CPBindingAuth` キーとともに設定するキーです。このキーには、その i アプリの Jar ファイルの内容のハッシュ値（SHA-1）を Base64 エンコードしたものを設定します。設定すべき値が Jar ファイルの内容に依存するため、本キーは i アプリをビルドする度に再設定を行う必要があります（i アプリをビルドした後に、ツールを利用して設定します）。

CPName キー（他の i アプリと外部メモリのデータを共有する i アプリでは必須）

共有識別子として `SHARE_CONTENTS_PROVIDER` を指定する i アプリにおいて、そのデータを利用するコンテンツプロバイダの名称を設定するキーです。このキーの値は、携帯電話ネイティブの外部メモリ管理ツールにおいて、`SHARE_CONTENTS_PROVIDER` の指定を伴って作成されたフォルダの情報を表示する際に利用されます（該当フォルダの詳細情報表示などにおいて、そのフォルダの作成者情報として表示されます）。

このキーの値には、16 バイト以内の SJIS テキストを指定します。`SHARE_CONTENTS_PROVIDER` の指定によって複数の i アプリで外部メモリのデータを共有する場合、そのデータを共有する全ての i アプリについて、このキーを同じ値で設定するようにしてください。

注意事項：

- このキーは `SHARE_CONTENTS_PROVIDER` を指定する i アプリにおける必須キーとなります。このキーが設定されていない i アプリは `SHARE_CONTENTS_PROVIDER` を使用することができないため注意してください。

なお、i アプリの ADF に対して以下のような変更を行った場合には、変更前の ADF でダウンロードされた i アプリはバージョンアップを行うことができなくなりますので注意してください。

- ・ `UseStorage` キーを削除した場合
- ・ `CPBindingAuth` キーの設定（デジタル署名処理）で使用する証明書を変更した場合

12.5 外部メモリ上のファイルのメディアプレーヤーでの再生

i アプリでは、外部メモリを応用した機能として、i アプリが外部メモリ上に保存したファイル（i モーションファイル）を直接ネイティブのメディアプレーヤーに引渡して再生するための API が定義されています。この機能に対応した携帯電話で外部メモリ上のファイルをメディアプレーヤーで再生するには、`com.docomostar.opt.device.MediaPlayer` クラスを使用します。この機能を使用して i モーションファイルを再生する場合には i モーションを `MediaMovie` オブジェクトの形式にする必要がないため、i アプリ実行環境のヒープメモリを圧迫することなく i アプリから大きな i モーションファイルを再生することができます。

このクラスは、以下の手順で使用します。

- 1) `static` メソッド `MediaPlayer.getMediaPlayer()` を使用して、このクラスのオブジェクトを取得します。
- 2) 再生したいファイルに対応する `File` オブジェクトを作成します。
- 3) `MediaPlayer.play()` メソッドを、2) で作成した `File` オブジェクトを引数に指定して呼び出します。このメソッドを呼び出すとネイティブのメディアプレーヤーが起動され、i アプリはサスペンドします。

`play()` メソッドには、指定したファイルの再生開始位置を指定することもできます。

- 4) ネイティブのメディアプレーヤーが終了すると i アプリはレジューム復帰し、`play()` メソッドは戻り値としてメディアプレーヤーの終了ステータス (`MediaPlayer` クラスの `STATUS` 定数) を返します。

i アプリから起動されたネイティブのメディアプレーヤーには停止、巻き戻し、早送りなどの操作を行うボタンがありますが、それらのいずれを操作した場合もメディアプレーヤーでの再生は中断され、i アプリはレジューム復帰します。その際、どのボタンにより終了したかなどを示す終了ステータスが `play()` メソッドから返されます。

`play()` メソッドから復帰した後、`getLastStoppedPosition()` メソッドを使用して、前回の再生におけるメディアの停止位置を調べることができます。

なお、`MediaPlayer` クラスは外部メモリ API を搭載している機種の中でもオプションの機能です。外部メモリ API を搭載していても、`MediaPlayer` クラスをサポートしていない機種もありますので注意してください。

12.6 メーカー実装に起因する外部メモリ利用上の注意事項

12.6.1 暗号化機能利用時の注意事項

外部メモリの実装によっては、暗号化機能利用時に以下のような制限を持つものがありますので注意してください。

- 暗号化機能を有効にした場合、ファイル書き込みについて 1 ファイルあたりの最大サイズに制限が発生する場合があります (500K バイト程度)。暗号化機能を有効にする場合は、ファイルサイズは最大でも 500K バイト以内となるように設計してください。それより大きな容量のデータを扱う必要がある場合は、複数のファイルに分割することを検討してください。
- 暗号化機能を有効にしたファイル書き込みを行っている間 (ファイルオープンからクローズまでの間) は、他のファイルに対して暗号化機能を有効にした読み書きが行えない場合があります。暗号化機能を有効にする場合は、ファイルの書き込みは他の外部メモリアクセス処理とは排他的に行うようにしてください。

また音楽プレーヤーなどのネイティブアプリケーションが外部メモリ上の暗号化コンテンツを使用している間は同時に実行されている i アプリ側では暗号化機能を有効にしたファイルの読み書きが行えなかったり、逆に i アプリ側で暗号化機能を有効にしたファイルをオープンしている状況ではネイティブアプリケーションが外部メモリ上の暗号化コンテンツを利用できない場合があります。暗号化機能を有効にしたファイルについては、オープンされている期間ができるだけ短くなるよう考慮してください。

- 暗号化機能を有効にしたファイルに対して追記や部分的な変更を行う場合、実装によっては追記部分や変更部分だけでなく、ファイル全体に対して再度暗号化処理を行う必要がある場合があります。この場合、ファイル書き込みのパフォーマンスは、追記部分や変更部分のみのサイズではなく、ファイル全体のサイズに比例して影響を受けます。
- 一部の機種では、`EncryptionAttribute` クラスを使用した場合に、システム内部におけるデフォルト暗号化パラメータの関係で暗号化機能が働かない場合があります。どの機種でも暗号化機能が有効となるようにするためには、暗号化の指定は `SDBindingEncryptionAttribute` クラスを使用して行うようにしてください。
- SD メモリカードで採用されている暗号化仕様では、ファイルサイズを n バイトとした際に、ファイル末尾の $n\%8$ バイト (%: 剰余) 分のデータは暗号化されません。ファイルサイズが 8 バイトに満たない場合を含め、ファイル全体を完全に暗号化する必要がある場合は、ファイルサイズが 8 の倍数となるようにパディングを行うようにしてください。

12.6.2 ファイル名に関する注意事項

外部メモリで使用されるファイルシステムは、メーカー実装によってファイル名の太文字・小文字の違いを区別する場合と区別しない場合があります。いずれの場合にも正しく動作する i アプリを開発するには、ファイル名の指定を以下のような指針に基づいて行ってください。

- 同じファイルにアクセスする場合、ファイル名にはいつでも大文字・小文字の指定も含め完全に同じ文字列を指定するようにしてください。新規にファイルを作成する時と後刻そのファイルを再度オープンする時とで大文字・小文字の指定が異なるファイル名を使用すると、再度のオープン時にファイルが存在しない旨を示す例外が発生する場合があります。
- 異なる複数のファイルを取り扱う場合、それらには大文字・小文字の差を同一視した場合にも異なる名称となるファイル名をつけるようにしてください。

ファイル名を指定する際には、常に全て大文字で指定する、もしくは全て小文字で指定するといったガイドラインを設けることを推奨します。

なお、ファイル名の大文字・小文字の違いを区別しないファイルシステムにおいて、内部のファイル名管理がどのように行われるか (`File.getPath()` メソッドの返すファイルパス名が大文字もしくは小文字に統一されるかどうかなど) はメーカーにより異なります。

第 13 章

デジタルテレビ制御

i アプリでは、デジタルテレビ（以降、DTV と略記します）機能を搭載した携帯電話向けに、オプション API として i アプリから DTV 機能を制御するための機能が定義されています。この機能に対応した携帯電話では、i アプリから携帯電話ネイティブの DTV 機能に対し、API を通じて以下のような制御を行うことができます。

- DTV 機能の連携起動
- テレビ番組の視聴・録画予約の登録

なお、テレビ番組の録画予約の登録機能は、DTV 機能搭載機の中でもオプションの機能です。DTV 機能に対応している機種でも、録画予約登録に対応していないものもありますので注意してください。

13.1 DTV 機能の連携起動

本機能に対応した携帯電話では、`com.docomostar.system.Launcher.launch()` メソッドを使用することによって、i アプリから DTV 機能を連携起動することができます。

DTV 機能を連携起動するには、`launch()` メソッドの第 1 引数（起動対象となるアプリケーションの種別）に `Launcher.LAUNCH_DTV` を指定します。また第 2 引数（起動引数を示す String 配列）には、以下の内容を表す数字文字列を指定します。

- 0 番目の要素： 物理チャンネル番号を示す数字文字列。13 以上 62 以下の値を設定することができます。
- 1 番目の要素： サービス識別番号を示す数字文字列。0 以上 65535 以下の値を設定することができます。
- 2 番目の要素： 系列識別番号を示す数字文字列。0 以上 11 以下の値を設定することができます。

注意事項：

- `Launcher.launch()` メソッドで DTV 機能を連携起動する場合は、このメソッドの第 2 引数そのもの、もしくは第 2 引数の各要素には `null` を設定することができます。その場合は、物理チャンネル番号やサービス識別番号、系列識別番号が指定されなかったものとして DTV 機能が起動します。
- DTV 機能を連携起動する i アプリは、ADF の UseDTV キーで DTV 機能を使用することを宣言（値“`launch`”を設定）していなければなりません。そうでない i アプリが DTV 機能を連携起動しようとすると例外が発生します。ADF における UseDTV キーの設定例を以下に示します。

```
UseDTV = launch
```

- `Launcher.launch()`メソッドでDTV機能を連携起動する際に、iアプリが終了するかサスペンドするかはメーカーにより異なります。iアプリがサスペンドする機種では、連携起動したDTV機能が終了するとiアプリはレジューム復帰します。

13.2 テレビ番組の視聴・録画予約の登録

テレビ番組の視聴・録画予約を登録するには、`com.docomostar.system.DTVSchedule`クラスのstaticメソッド`addEntry()`、および予約のためのパラメータのコンテナクラスである`com.docomostar.system.DTVScheduleParam`クラスを使用します。

テレビ番組の視聴・録画予約を登録するには、`DTVScheduleParam`クラスのオブジェクトに以下の情報を設定した上で、予約タイプ（視聴予約もしくは録画予約を示す、`DTVSchedule`クラスのTYPE定数）とともに`addEntry()`メソッドに渡します。

- ・ 物理チャンネル番号 (13～62)、サービス識別番号 (0～65535)、系列識別場号 (0～11) の値 (*1)
- ・ チャンネル名と番組名を表すテキスト (いずれも省略可能)
- ・ 視聴・録画の時間帯 (開始時刻、終了時刻、繰り返しの有無やサイクルを指定する日時指定タイプ、期間の集合) を表す `ScheduleDate` オブジェクト (`ScheduleDate` の詳細は「iアプリコンテンツ開発ガイド 詳細編」11.9 項を参照してください。)

(*1) 物理チャンネル番号、サービス識別番号、系列識別番号はそれぞれ省略することができますが、これらのうち最低1つは省略せずに設定する必要があります。これらの項目には、値を指定しないことを表す定数として、`FREQUENCY_NONE`、`SERVICE_ID_NONE`、`AFFILIATION_ID_NONE` の3つが `DTVSchedule` クラスに用意されています。

テレビ番組の視聴・録画予約を登録すると、携帯電話は以下のような動作を行います。

- (1) アプリケーションプログラムから `DTVSchedule.addEntry()`メソッドを呼び出します。このメソッドが呼び出されるとiアプリはサスペンドします。
- (2) 視聴・録画予約の登録を行うためのネイティブのユーザーインターフェース画面が起動します。この画面では、ユーザーはiアプリからの視聴・録画予約の新規登録を許可、キャンセルするだけでなく、設定内容を変更することもできます。
- (3) ユーザーが視聴・録画予約の新規登録を許可するとそのときに設定されていた内容で視聴・録画予約の新規登録が行われ、iアプリはレジュームして `addEntry()`メソッドはアプリケーションプログラムに復帰します。ユーザーが視聴・録画予約の新規登録をキャンセルしても、特に例外は発生せずアプリケーションプログラムに復帰します。

`addEntry()`メソッドは、視聴・録画予約の新規登録に成功すると `true` を返します。登録がキャンセルされたり、登録件数の上限を超えたなどの要因で登録が行われなかった場合は `false` を返します。

注意事項：

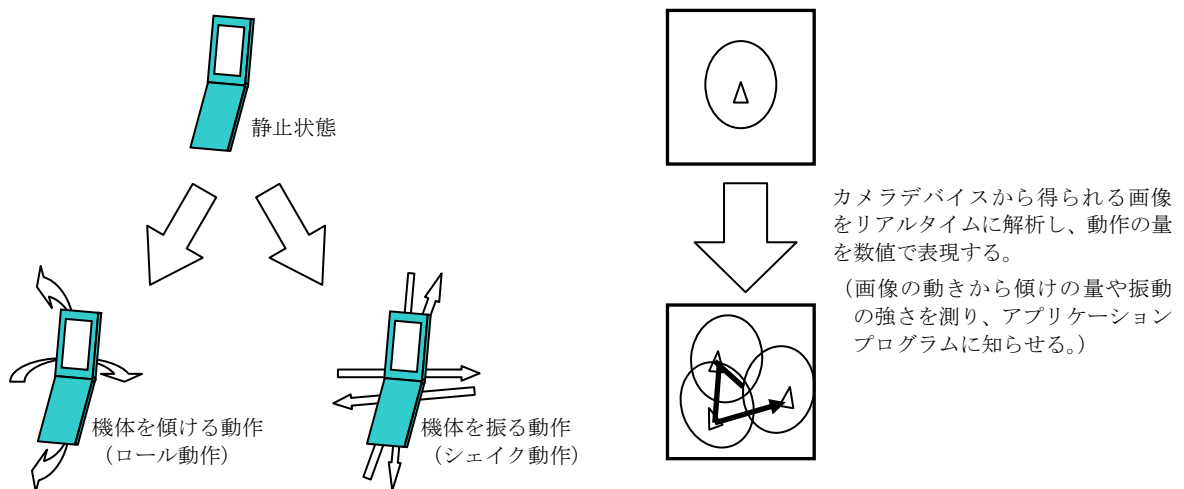
- 視聴・録画予約の登録において、期間の集合の利用可否、およびどのような日時指定タイプが使用可能かはメーカーにより異なります。どの機種でも共通的に指定可能な日時指定タイプは「1回限り」です。また、特定の機種でどのような日時指定タイプが使用できるかは `DTVSchedule.getSupportedTypes()`メソッドを使用して調べることができます。
- 視聴・録画予約の登録において、どのような日時指定タイプで期間の集合が使用可能かはメーカーにより異なります。特定の機種でどのような日時指定タイプで期間の集合が使用できるかは `DTVSchedule.getMultiRepeatableTypes()`メソッドを使用して調べることができます。
- テレビ番組の視聴・録画予約を行うiアプリは、ADFの `AccessUserInfo` キーにてこの機能を使用することを宣言（値“yes”を設定）する必要があります。

- 端末ネイティブの時計が設定されていない状態では、この機能を使用することはできません。
- 繰り返しのない1回限り（ScheduleDate.ONETIME）の予約を行う場合、視聴・録画の開始時刻、終了時刻は、端末ネイティブの時計に対して将来の時刻である必要があります。そうでない時刻を指定した場合は視聴・録画予約の登録はできません。

第 14 章

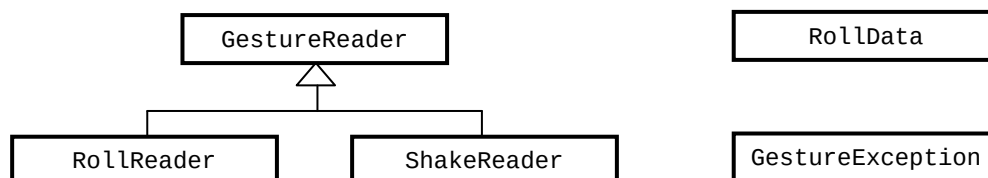
動作認識

動作認識機能は、携帯電話のカメラデバイスから得られる画像をリアルタイムに解析し、画像の時系列的な変化を携帯電話の機体の移動量として i アプリに提供する機能です。動作認識機能を使用することで、携帯電話の機体そのものをアナログ入力デバイスに見立て、ユーザーの行動（機体を傾けたり振ったりする動き）を数値として得ることができるようになります。



動作認識機能は `com.docomostar.device.gesturereader` パッケージで提供されます。動作認識機能のクラス構成を以下に示します。

`com.docomostar.device.gesturereader` パッケージ



クラス	概要
GestureReader	動作認識機能の基底クラスです。アプリケーションプログラムはこのクラスを直接使用するのではなく、このクラスを継承した、動作種別ごとに用意されている動作認識機能クラスを使用します。現在のプロファイルでは、ロール動作（傾け： RollReader クラス）とシェイク動作（振動： ShakeReader クラス）の2種類の動作の認識がサポートされています。
RollReader	ロール動作認識機能です。ロール動作認識機能では、以前（基準時点）のカメラ画像と現在のカメラ画像の比較結果から携帯電話のロール量を知ることができます。認識した結果は、次項に示す RollData オブジェクトの形で返されます。
RollData	ロール動作の認識結果を表すクラスです。このクラスのメソッドを使用して、携帯電話のロール量（x 方向、y 方向それぞれの移動量）を取得できます。
ShakeReader	シェイク動作認識機能です。シェイク動作認識機能では、定められた範囲の値で携帯電話の機体を振る動作の強さを表現し、その値を数値として知ることができます。
GestureException	動作認識機能に固有の例外を表現する例外クラスです。この例外クラスは getStatus() メソッドを備えており、例外の大きな原因を知ることができます。

以下に、ロール動作認識およびシェイク動作認識の使用方法について解説します。

ロール動作認識

ロール動作認識を行うための **RollReader** オブジェクトを取得するには、**RollReader** クラスの static メソッド **getRollReader()** を使用します。**getRollReader()** メソッドの引数には、動作認識に使用するカメラデバイスのカメラ ID (**Camera.getCamera()** メソッドでも使用する、複数のカメラデバイスの中から目的のものを選ぶための ID) を指定します。取得した **RollReader** オブジェクトに対して **start()** メソッドを呼び出すことで動作認識が利用可能な状態となります。また **stop()** メソッドを呼び出すことで動作認識機能が停止状態となります。

動作認識が利用可能な状態で **RollReader.getRollData()** メソッドを呼び出すと、その時点での認識結果を保持する **RollData** オブジェクトを取得できます。**RollData** オブジェクトは **getRollData()** メソッドが呼び出される度に新たに生成されるのではなく、**getRollData()** メソッドの初回呼び出し時に生成されたオブジェクトが以降の **getRollData()** メソッド呼び出し時に再利用される点に注意してください。

RollData オブジェクトからは、以下の2つの観点からのロール量（x 方向、y 方向それぞれの移動量）を知ることができます。

- 基準時点と **getRollData()** メソッド呼び出し時点の間でのロール量 (**RollData.getAccumulatedMotion()** メソッド)
- getRollData()** メソッド呼び出し時点における、最新の連続2フレーム間でのロール量 (**RollData.getImmediateMotion()** メソッド) (*1)

(*1) カメラ画像の認識がどの程度のフレームレートで行われるかは各機種のパフォーマンス、および処理実行時の携帯電話システムの負荷状況により異なります。他に負荷のかかっていない状況でのフレームレートの目安は、概ね 10～10 数 fps 程度となります。

基準時点は、当初は **RollReader.start()** メソッドを実行した時点に設定されています。その後 **RollReader.getRollData()** メソッドもしくは **RollReader.recenter()** メソッドを呼び出す度に、新たにその時点が基準時点として設定されます（基準時点におけるカメラ画像のことを基準画像と呼びます）。

RollData オブジェクトにおけるロール量の表し方（値の範囲など）については、API リファレンスを参照してください。

シェイク動作認識

シェイク動作認識を行うための **ShakeReader** オブジェクトを取得するには、**ShakeReader** クラスの static メソッド **getShakeReader()** を使用します。**getShakeReader()** メソッドの引数にも、上述の **RollReader.getRollReader()** メソッドと同じく、動作認識に使用するカメラデバイスのカメラ ID

を指定します。また、動作認識の開始、停止の制御も、RollReader クラスと同様に `start()` メソッドと `stop()` メソッドを使用して行います。

動作認識が利用可能な状態で `getShakeData()` メソッドを呼び出すと、その時点での認識結果（振動の強さ）を数値として取得できます。振動の強さの表し方（値の範囲など）については、API リファレンスを参照してください。

注意事項：

- 複数のカメラデバイスを搭載している機種では、全てのカメラが動作認識に使用できるとは限りません。どのカメラデバイスが動作認識に使用できるかはメーカーにより異なります。一般的には、インカメラ（ディスプレイ側に配置されたカメラ）が動作認識をサポートしている場合には、アウトカメラ（背面などに配置されたカメラ）を使用するよりもインカメラを使用する方が、手にさえぎられずスムーズに動作認識を行うことができます。
ただしこの場合は、動作認識処理は主にインカメラに映し出されたユーザーの顔の画像を基準として行われるため、ユーザーの顔がカメラの画角から外れるような大きな動作が行われた場合には正しく動作認識が行えない場合があります。
なおロール動作認識においては、インカメラ使用時とアウトカメラ使用時では、機体を傾けた際に得られる移動量の向きが **x 方向**、**y 方向** とも逆方向になります。これは機体を傾けた際に、インカメラの捉える画像の動きとアウトカメラの捉える画像の動きが逆方向になるためです。インカメラを使用する場合とアウトカメラを使用する場合では、機体を傾けた際の移動量の向きの解釈を変える必要がある点に注意してください。（これは機体の傾け動作でのみ発生する特性であり、機体の平行移動動作では向きの逆転は生じません。）
- 複数の動作認識オブジェクトを同時に利用可能な状態（`start()` メソッドが呼び出された状態）にすることはできません。
- 動作認識オブジェクトの `start()` メソッド、`stop()` メソッドを呼び出すことで、カメラデバイスと動作認識エンジンの起動や停止が行われます。これらのメソッドは起動処理もしくは停止処理が完了するまでブロックしますが、特に起動処理ではカメラデバイスの起動に時間を要するため、完了までに一定の時間（数百～数千 msec 程度）がかかることに注意してください。
- 動作認識オブジェクトが利用可能な状態でサスペンド状態に遷移した場合は、動作認識は自動的に停止します。自動的に再開は行われないため、必要に応じてアプリケーションプログラム側で再開処理を行うようにしてください。
- ロール動作認識においては、カメラで捉えた画像の動きがあまりにも速い、もしくはカメラで捉えた画像が平坦（例えば一面白の壁など）で動きを検出できないなどの理由で、認識結果の品質を保てない場合があります。このため、RollData クラスでは認識結果の品質を調べるための `getQuality()` メソッドが用意されています。このメソッドを使用することで、認識結果の品質の大まかな目安を知ることができます。
- ロール動作認識では、RollReader.getRollData() メソッドでロール量を取得するたびに、新たにその時点が次回 getRollData() メソッドが使用される際の基準時点となります（getRollData() メソッドが呼び出されるたびに基準時点が移動していきます）。このため、常にある特定の時点（例えば、RollReader.start() メソッドを呼び出してロール認識を開始した時点など）を基準としたロール量を求めたい場合には、RollData.getAccumulatedMotion() メソッドで得られるロール量を積算していく必要があります。
ただしロール動作認識で得られるデータには若干の誤差が含まれるため、このような利用方法では誤差も積算されることとなります。ロール量を積算していくような使い方をした場合、この誤差の蓄積により、得られる結果と実際の機体の傾き状態が徐々にずれて行く場合があります。
- カメラファインダ機能と動作認識機能を同時に起動することはできません。

第 15 章

電子コンパス

携帯電話では、位置情報に関連する機能として電子コンパスを搭載している場合があります。このような機種のために、i アプリでは電子コンパスを扱うための `com.docomostar.device.location.Compass` クラスが定義されています。

電子コンパスを使用するためには `Compass` クラスの static メソッド `getCompass()` を呼び出して `Compass` オブジェクトを取得します。`Compass` オブジェクトに対して `setEnabled()` メソッドを呼び出して電子コンパス機能を有効化した後に、`getAzimuth()` メソッドを呼び出すことで携帯電話本体の向いている方角（概ね端末ディスプレイの上部が向いている方角）を調べることができます。

方角の表現には、`com.docomostar.device.location.Degree` クラスが使用されます。`getAzimuth()` メソッドは端末の向いている方角を、磁北からの時計回りの角度を表す `Degree` オブジェクトとして返します。`Degree` クラスは、角度を以下のいずれかの単位で表します。

- DEGREE 単位

45.67° のように、角度（単位は°）を 10 進小数点数で表現する単位です。`DEGREE` 単位で角度を扱う際は、`double` 型による表現を利用し、その値がそのまま単位が°である角度を表します。`DEGREE` 単位で表される角度を取得するには `getDegreeValue()` メソッドを使用します。

- DMS 単位

12° 34' 56.00" のように、度、分、秒を表す 3 つの値の要素から構成される単位です（秒は小数点以下の精度を持ちます）。`Degree` クラスには、これら各要素に対応する値取得用のメソッド（`getDegreePart()`、`getMinutePart()`、`getSecondPart()`）が用意されています。

【Star-1.3】

Star-1.3 プロファイル以降、電子コンパスを扱う機能の中でもオプションとなる機能を扱うため、`Compass2` クラスと `Compass2EventListener` インタフェースが i アプリオプション API として `com.docomostar.opt.device.location` パッケージに定義されました。

Star-1.3 プロファイルでは、電子コンパス機能から後述する特定のイベントを検知し、そのイベントをアプリケーションプログラムに通知するための機構が用意されました。

電子コンパスからのイベント検知機能を使用するには、アプリケーションプログラムで以下のような手順を踏みます。

- 1) `Compass2EventListener` インタフェースを実装したリスナークラスのオブジェクトを用意し、`Compass2.setEventListener()` メソッドでシステムに登録します。

- 2) 検知するイベントの種別を引数に指定して **Compass2.startEventDetection()** メソッドを呼び出し、電子コンパス機能からのイベント検知を開始します。複数の種別のイベントを検知したい場合は、**startEventDetection()** メソッドの引数にそれらイベント種別の論理和を指定します。指定したイベントが発生すると、リスナオブジェクトの **actionPerformed()** メソッドが呼び出されることでアプリケーションプログラムにイベントが通知されます。**actionPerformed()** メソッドは、イベント発生源になった **Compass2** オブジェクト、発生したイベントの種別、および各イベントに固有のイベント値を受け取ります。
- 3) イベント検知を停止し機能の利用を終了するには、**Compass2.stopEventDetection()** メソッドを呼び出します。このメソッドを呼び出すと、開始したイベント検知はすべて停止します。

Star-1.3 プロファイルでは、キャリブレーションが必要となったことを示すイベント

(**Compass2.EVENT_CALIBRATION_REQUIRED**) とキャリブレーションが完了したことを示すイベント (**Compass2.EVENT_CALIBRATION_COMPLETED**) が定義されています。

キャリブレーションとは、電子コンパスが磁北位置を正しく認識できなくなった場合にそれを補正することを言います。電子コンパスは、家電製品等の磁気源近くにある場合や電子コンパスを起動した直後である場合などに、磁北位置を正しく認識できず方角を正確に調べることができないことがあります。そのような場合にキャリブレーションを行うことで、正しい方角を調べることができるようになります。

キャリブレーションをサポートする機種では、基本的にシステム側で自動的にキャリブレーションを行います。強磁界下や地下街など地磁気を検出できない環境下ではキャリブレーションを行うことはできません。

このような場合、i アプリ実行環境は **EVENT_CALIBRATION_REQUIRED** イベントを通知します。このイベントの通知を受けたら、ユーザーに磁気を帯びたものから機体を遠ざけるか、地磁気を検出できるよう場所を変えることなどを促してください。（場合によっては、8 の字を描くように機体を回すなどの操作も必要となります。）障害が取り除かれキャリブレーションが行われたら、i アプリ実行環境は **EVENT_CALIBRATION_COMPLETED** イベントを通知します。

なおどのイベントを検知可能かは実装により異なり、**Compass2.getAvailableEvent()** により調べることが出来ます。

【Star-1.5】

Star-1.3 プロファイルまでは、必要なタイミングでアプリケーションプログラムが

Compass.getAzimuth() メソッドを呼び出し、携帯電話の向いている方角を計測を行うワンショット計測のみがサポートされています。

Star-1.5 プロファイルでは、ワンショット計測に加え、アプリケーションプログラムが指示する間隔でシステムが自動的に計測を行ってその結果を蓄積していき、アプリケーションプログラムが必要とするタイミングでその結果を取り出す周期計測を行う **API** が追加されました。

周期計測を開始するには、**Compass.setEnabled()** メソッドで電子コンパス機能を有効化した後に、周期間隔を指定して **Compass.start()** メソッドを呼び出します。周期計測ではアプリケーションプログラムとは非同期に、指定した周期間隔で携帯電話の向いている方角の計測が行われ、その結果がシステム内部のバッファに蓄積されていきます。周期計測を開始した後に、任意のタイミングで、**Compass.getData()**、もしくは **Compass.peekLatestData()** メソッドを呼び出すと計測結果を取得することができます。**getData()** メソッドは蓄積されているすべての計測結果を返しますが、**peekLatestData()** メソッドは蓄積されている計測結果のうち最新の 1 件のみを返します。開始された周期計測は **Compass.stop()** メソッドを呼び出すと停止します。

なお、ワンショット計測、周期計測で計測する方角は、事前に **PhoneSystem** クラスの **setAttribute()** メソッドにより **DEV_EYE_DIRECTION** 属性の設定を行うことにより、設定した方位デバイスの基準方向（第 5 章参照）が反映された方角を取得することができます。

getData()、**peekLatestData()** メソッドは計測結果を DEGREE 単位で表した値（double 型）で返します。DMS 単位で表された角度を取得する場合は、**com.docomostar.device.location.Degree** クラスを利用して単位の変換を行ってください。

周期計測の計測結果が何件蓄積されているかは `Compass.getDataSize()` メソッドを使用して調べることができます。

また、周期計測に関して電子コンパス機能がサポートしている能力・機能を知るために以下のメソッドが用意されています。

- **`getIntervalResolution()`**
周期計測における、実行可能な最短の周期間隔を調べるメソッドです。周期計測ではアプリケーションプログラムが周期間隔を指定しますが、指定された値はシステム内部でこのメソッドの返す値の倍数に丸められます。
- **`getMaxDataSize()`**
周期計測において、システム内部のバッファに何件分の計測結果を保持できるかを返します。

注意事項：

- 電子コンパス機能は位置情報 API のカテゴリに属する機能です。位置情報 API のうち電子コンパス以外の機能は、位置情報 API の利用が許可されているトラステッド i アプリしか使用することはできません。本書では、電子コンパス以外の位置情報 API については解説しません。
- **`setEnabled()`** メソッドを呼び出すことで、**Compass** オブジェクトを保持したまま電子コンパスの有効、無効を切り替えることができます。一般に電子コンパスが有効な状態では消費電力が大きくなるため、機能を使用しない間は電子コンパス機能は無効にしておくようにすることを推奨します。
- **Compass2** クラスをサポートしている場合、**`Compass.getCompass()`** は常に **Compass2** オブジェクトを返します。
- 電子コンパスからのイベント検知機能を開始している状態でサスペンド状態に遷移した場合は、イベント検知は自動的に停止します。自動的に再開は行われないため、必要に応じてアプリケーションプログラム側で再開処理を行うようにしてください。
- 周期計測によってシステム内部に蓄積された計測結果は、以下の契機でクリアされます。
 - **`Compass.getData()`** メソッドによって、周期計測の結果が取り出されたとき
 - 周期計測を停止した後、周期計測を開始するか、ワンショット計測 (**`Compass.getAzimuth()`** メソッド) を行ったとき
 - **`Compass.disposeData()`** メソッドによって明示的に計測結果のクリアが指示されるか、もしくは計測結果を蓄積したまま i アプリを終了したとき

周期計測を停止した後も、これらの契機までは計測結果はシステム内部に蓄積されたままとなっており、**`Compass.getData()`** メソッドで結果を取り出すことができます。
- 周期計測を開始している状態でサスペンド状態に遷移した場合や **`setEnabled()`** メソッドにより電子コンパスを無効化した場合は、周期計測は自動的に停止します。自動的に再開は行われないため、必要に応じてアプリケーションプログラム側で再開処理を行うようにしてください。その際、前項の規定により、周期計測が停止する前に蓄積された計測結果は周期計測を再開する前に取り出す必要があることに注意してください。
- 周期計測を行っている状態では、ワンショット計測を行うことはできません。
- 周期計測を行っている状態で、**`com.docomostar.opt.ui.Canvas2`** クラス (第 5 章参照) を利用して座標系を変更した場合、蓄積された計測結果の一貫性が保証されなくなり、期待する計測結果が得られないことがあります。期待する計測結果を得るためには、一旦周期計測を停止させ蓄積された計測結果を取得してから、座標系を変更し、周期計測を再開してください。
- 周期計測と歩行方位取得機能の歩行検知計測機能を同時に利用することはできません。

第 16 章

加速度センサー

加速度センサー機能は、文字通り携帯電話の機体にかかる加速度を数値として取得する機能です。加速度の変化を調べることで、機体の動きの変化（機体が移動や静止を行う際の加速度の変化）を知ることができます。また、地球の重力（地面方向への $1g = \text{約 } 9.8\text{m/s}^2$ の加速度）が機体にかかる向きを調べることで、地面（水平面）に対する機体の傾きを知ることができます。

i アプリの加速度センサー機能は、携帯電話の機体にかかる加速度を数値として取得する加速度センサーとしての基本機能に加え、以下のようなオプション機能から構成されています。

- ・ 機体のロール角、ピッチ角（水平面に対する、機体の左右方向および上下方向の角度）の取得
- ・ メインスクリーンの向きの取得
- ・ 加速度の変化に関するイベント検知

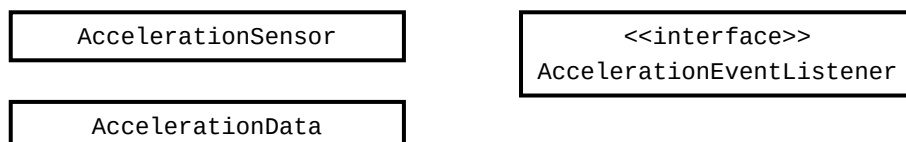
加速度センサー機能をサポートする機種においても、上記オプション機能をサポートするかどうかはメーカーの設計に委ねられています。

なお加速度センサーAPI では、加速度を x、y、z の 3 方向の成分として取り扱えるよう API を定義しています。これは 3 軸の加速度センサーデバイスを前提とした仕様ですが、実際に携帯電話に搭載される加速度センサーデバイスは 2 軸以下のものであることもあり、その場合は取得できる方向成分に制限が生じる可能性があります。

このように、加速度センサー機能はメーカーによって使用可能な機能・能力に差が生じる可能性があるため、加速度センサーAPI ではその機種における加速度センサー機能の機能・能力のサポート範囲を調べるための API が用意されています。

加速度センサー機能は、`com.docomostar.device.location` パッケージで提供されます。加速度センサー機能のクラス構成は以下の通りです。

`com.docomostar.device.location` パッケージ



クラス・インタフェース	概 要
AccelerationSensor	加速度センサー機能を提供するクラスです。アプリケーションプログラムは、このクラスを使用して加速度の計測を行います。加速度の計測には、アプリケーションプログラムが必要とする任意のタイミングでの計測（ワンショット計測）と、アプリケーションプログラムが指示する間隔で繰り返し計測を行う周期計測の2種類があります。 計測結果は、次項に記載する AccelerationData クラスのオブジェクトとして表現されます。
AccelerationData	加速度センサーの計測結果を表すクラスです。このオブジェクトから、軸ごとの加速度データや機体の傾き、メインスクリーンの向きといった計測データ値を取り出すことができます。 ワンショット計測では1つの AccelerationData オブジェクトに1件分の計測結果が格納されますが、周期計測では1つの AccelerationData オブジェクトに複数件の計測結果が格納される場合があります。
AccelerationEventListener	加速度の変化に関するイベント検知において、イベント通知を受けるリスナオブジェクトが実装すべきインタフェースです。

加速度センサー機能を利用するには、まず **AccelerationSensor** クラスの static メソッド **getAccelerationSensor()** を呼び出して **AccelerationSensor** オブジェクトを取得します。機種によっては複数の加速度センサーデバイスを搭載しているものがあり、そのような機種では個々の加速度センサーデバイスに0から始まる整数がデバイスIDとして固定的に割り当てられています。デバイスIDに対応した加速度センサー機能を利用するには、オプションの static メソッドである **getAccelerationSensor(int)** の引数にこのデバイスIDを指定して **AccelerationSensor** オブジェクトを取得します。以下に、加速度センサーAPIの利用方法について解説します。

(1) 加速度センサー機能の機能・能力のサポート範囲

AccelerationSensor クラスには、加速度センサー機能がサポートしている機能・能力を知るための以下のようなメソッドが用意されています。

メソッド	概 要
getAvailableData()	加速度センサー機能で、どのデータを計測できるかを調べるメソッドです。x,y,z 各軸の加速度、機体のピッチ角、ロール角、メインスクリーンの向きといったデータ項目のうち、どの項目をサポートしているかを調べることができます。
getAvailableEvent()	加速度センサー機能で、どのような加速度変化イベントをサポートしているかを調べるメソッドです。現在のプロファイルで定義されているイベントには、メインスクリーンの向きの変化のイベントとダブルタップイベントがあります。 Star-1.3 プロファイル以降、加速度センサー機能へのアクセス可否状態の変更イベントが定義されました。
getIntervalResolution()	周期計測における、実行可能な最短の周期間隔を調べるメソッドです。周期計測ではアプリケーションプログラムが周期間隔を指定しますが、指定された値はシステム内部でこのメソッドの返す値の倍数に丸められます。

<code>getMaxDataSize()</code>	周期計測において、システム内部に蓄積可能な計測データの最大件数を調べるメソッドです。周期計測ではアプリケーションプログラムとは非同期に、定期的に加速度の計測が行われ、その結果がシステム内部のバッファに蓄積されていきます。このメソッドは、システム内部に何件分の計測データを保持できるかを返します。 なお、このバッファに保持可能な最大件数を超えて周期計測が続けられた場合は、バッファ内の古いデータから順次上書きされていきます。
<code>getMaxDataValue()</code>	各計測データ項目について、取り得る値の最大値を調べるメソッドです。
<code>getMinDataValue()</code>	各計測データ項目について、取り得る値の最小値を調べるメソッドです。
<code>isAccessible()</code>	この <code>AccelerationSensor</code> オブジェクトに対応付けられている加速度センサーにアクセスが可能であるかを調べるメソッドです。複数の加速度センサーを搭載した機種では、特定の条件下でアクセスできない加速度センサーが存在する場合がありますため、このメソッドにより現在アクセスが可能かどうかを調べることができます。 加速度センサーにアクセスできる場合は <code>true</code>、アクセスできない場合は <code>false</code> を返します。 このメソッドは、Star-1.3 プロファイルから追加されました。

`getAvailableData()`メソッドや `getAvailableEvent()`メソッドで返されるデータ項目種別やイベント種別は、`AccelerationSensor` クラスの定数として定義されています。

(2) ワンショット計測と周期計測

ワンショット計測とは、必要なタイミングでアプリケーションプログラムがシステムに計測指示を出し、計測を行う方法です。また周期計測とは、アプリケーションプログラムが指示する間隔でシステムが自動的に繰り返し計測を行ってその結果を蓄積していき、アプリケーションプログラムが必要とするタイミングでその結果を取り出す方法です。

ワンショット計測は、アプリケーションプログラムが任意のタイミングで `AccelerationSensor.getCurrentData()`メソッドを呼び出すことにより行われます。

また周期計測を行うには、`AccelerationSensor.start()`メソッド（引数に周期間隔を指定します）により周期計測を開始した上で、任意のタイミングで `AccelerationSensor.getData()`メソッド、もしくは `AccelerationSensor.peekLatestData()`メソッドを呼び出します。`getData()`メソッドは蓄積された全ての計測結果を返しますが、`peekLatestData()`メソッドは蓄積されている計測結果のうち最新の1件のみを返します。開始された周期計測は、`AccelerationSensor.stop()`メソッドを呼び出すと停止します。

`getCurrentData()`、`getData()`、`peekLatestData()`の各メソッドは、`AccelerationData` オブジェクトの形で測定結果を返します。システムはあらかじめ `getCurrentData()`、`getData()`、`peekLatestData()`の各メソッドごとに専用の `AccelerationData` オブジェクトを割り当てて保持しており、個々のメソッドはそれぞれ常に同じオブジェクトを返すようになっています。`AccelerationData` オブジェクトの利用方法については、次項を参照してください。

周期計測の計測結果が何件蓄積されているかは、`AccelerationSensor.getDataSize()`メソッドを使用して調べることができます。

(3) 測定結果の利用（`AccelerationData` クラス）

加速度センサー機能の計測結果は、`AccelerationData` クラスで表現されます。このクラスには、加速度センサー機能で計測可能な各データ項目ごとに値を取り出すためのメソッドが用意されています。

各軸毎の加速度：`getAccelerationX()` / `getAccelerationY()` / `getAccelerationZ()`
単位 mg（ミリジー = 1/1000 g = 0.0098m/s²）

機体のピッチ角：`getPitch()`
単位 度

機体のロール角 : `getRoll()`

単位 度

メインスクリーンの向き : `getScreenOrientation()`

単位 度 (ただし返却値は 0、90、180、270 のいずれかに丸められます)

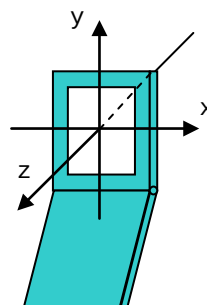
加速度の x, y, z の各方向はメインスクリーンを基準に定められています。 x 方向はメインスクリーン上を左から右に向かう方向、 y 方向はメインスクリーン上を下から上に向かう方向、 z 方向はメインスクリーンの奥 (背面側) から手前に向かう方向になります。

x, y, z の各方向 :

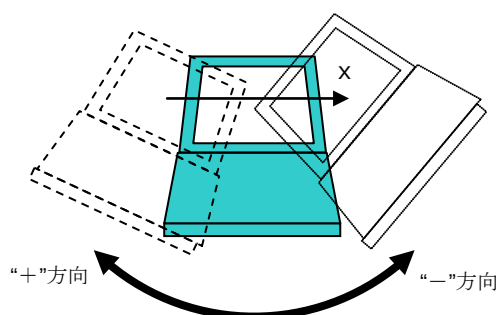
x : メインスクリーン上を左から右に向かう方向

y : メインスクリーン上を下から上に向かう方向

z : メインスクリーンの奥から手前に向かう方向



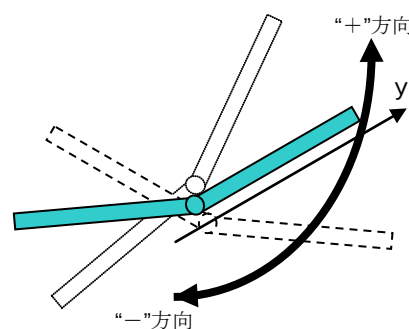
ロール角は、上記の x 方向が地面に対してどれだけ傾いているかを表します。またピッチ角は、上記の y 方向が地面に対してどれだけ傾いているかを表します。採用される加速度センサーの能力により、-180 度以上 180 度未満の範囲を計測できる機種と -90 度以上 90 度以下の範囲を計測できる機種があります。



ロール角 :

地面に対する x 方向の傾き

x 方向と地面が平行な状態を基準 (0 度) とする



ピッチ角 :

地面に対する y 方向の傾き

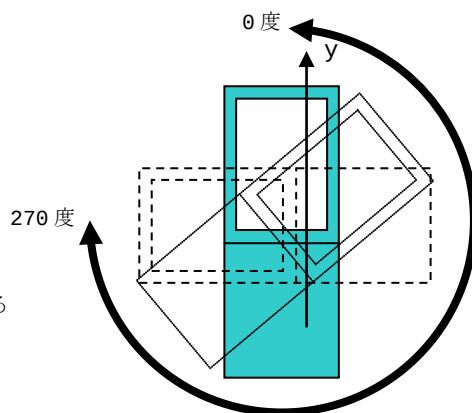
y 方向と地面が平行な状態を基準 (0 度) とする

また `getScreenOrientation()` メソッドで取得できるメインスクリーンの向きは、 z 方向を軸に y 方向がどの程度傾いているかを表します。

メインスクリーンの向き :

z 方向を軸とした、 y 方向の傾き

y 方向が上空を向いた状態を基準 (0 度) とする



周期計測の結果を `AccelerationSensor.getData()` メソッドで取得する場合には、蓄積された件数分の計測結果が 1 つの `AccelerationData` オブジェクトの中に格納されます。このため、データ項目値を取得する各メソッドは、結果を配列の形で返すようになっています。`getData()` メソッド用に割り当てら

れた **AccelerationData** オブジェクトは、周期計測でシステムが蓄積可能な件数を格納するのに十分な長さの配列をあらかじめ保持しています。実際に蓄積されている件数が少ない状況では配列の先頭要素から順に有効なデータが設定されますが、その有効な要素数は **size()** メソッドによって調べることができます。

(4) 加速度の変化に関するイベント検知

AccelerationSensor クラスには、加速度センサーから得られる情報をイベントとして検知し、そのイベントをアプリケーションプログラムに通知するための機構が用意されています。現在のプロファイルでは、イベントの種類としてはメインスクリーンの向きの変化のイベント

(**AccelerationSensor.EVENT_SCREEN_ORIENTATION**) およびダブルタップイベント

(**AccelerationSensor.EVENT_DOUBLE_TAP**)、加速度センサーのアクセス可否状態の変更イベント

(**AccelerationSensor.EVENT_ACCESSIBILITY**) が定義されています。

この機能を使用するには、アプリケーションプログラムは以下のような手順を踏みます。

- 1) **AccelerationEventListener** インタフェースを実装したリスナオブジェクトを用意し、**AccelerationSensor.setEventListener()** メソッドでシステムに登録します。
- 2) 検知するイベントの種類を引数に指定して **AccelerationSensor.startEventDetection()** メソッドを呼び出し、加速度イベントの検知を開始します。複数の種類のイベントを検知したい場合は、**startEventDetection()** メソッドの引数にそれらイベント種類の論理和を指定します。指定したイベントが発生すると、リスナオブジェクトの **actionPerformed()** メソッドが呼び出されることでアプリケーションプログラムにイベントが通知されます。**actionPerformed()** メソッドは、イベント発生源になった **AccelerationSensor** オブジェクト、発生したイベントの種類、および各イベントに固有のイベント値を受け取ります。
- 3) イベント検知を停止し機能の利用を終了するには、**AccelerationSensor.stopEventDetection()** メソッドを呼び出します。

メインスクリーンの向きの変化のイベントでは、機体を傾けたり逆さまにするなどしてメインスクリーンの向きが変化したことをアプリケーションプログラムに通知します。このイベントでは、イベント値

(**actionPerformed()** メソッドの第3引数) でメインスクリーンの向きが通知されます。向きは上空方向を 0 度とし、0、90、180、270 のいずれかに丸められます。

ダブルタップイベントでは、ユーザーが機体の特定の箇所をダブルタップ（指先などで2回軽くたたく）したことをアプリケーションプログラムに通知します。このイベントでは、イベント値でダブルタップされた箇所が通知されます。認識可能な箇所を示す値は、**AccelerationSensor** クラスに **DOUBLE_TAP** 定数として定義されています。どれを認識可能な箇所としてサポートするかはメーカーにより異なる場合がありますが、現在のプロファイルでは、認識可能な箇所として右側面、左側面、ディスプレイ面、背面の4箇所が規定されています。

加速度センサーのアクセス可否状態の変更イベントでは、加速度センサーへのアクセス可否の状態が変更したことをアプリケーションプログラムに通知します。このイベントでは、イベント値に

AccelerationSensor.ACCESSIBLE_TRUE あるいは **AccelerationSensor.ACCESSIBLE_FALSE** のいずれかが通知されます。**ACCESSIBLE_TRUE** は、加速度センサーにアクセスできない状態からアクセスできる状態に変更した場合に通知され、**ACCESSIBLE_FALSE** は加速度センサーにアクセスできる状態からアクセスできない状態に変更した場合に通知されます。

【Star-1.3】

Star-1.3 プロファイル以降、複数の加速度センサーデバイスを搭載している機種向けに **AccelerationSensor.getAccelerationSensor(int)** メソッドが追加されました。

このメソッドをサポートするかどうかは実装によって異なり、サポートする場合、**AccelerationSensor.getAccelerationSensor()** を呼び出した場合は、**getAccelerationSensor(0)** を呼び出した場合と同様の振る舞いとなります。

また複数の加速度センサーデバイスを扱うことが可能な機種では、特定の条件下ではアクセス可能とならない加速度センサーが存在する場合があります。

そのため、現在アクセスが可能であるかどうかを調べるメソッドとして、**AccelerationSensor.isAccessible()** が追加されました。加えて、加速度センサーのアクセス可否状態が変更されたことを示

スイベント `EVENT_ACCESSIBILITY` が `AccelerationSensor` クラスに追加され、そのイベント値である `ACCESSIBLE_TRUE` と `ACCESSIBLE_FALSE` が追加されています。

【Star-1.5】

Star-1.3 プロファイルまでは、計測できるロール角、ピッチ角の範囲は機種によって-180 度以上 180 度未満のものと-90 度以上 90 度以下のものがあります。

Star-1.5 プロファイルでは、ロール角、ピッチ角の範囲はどの機種においても-180 度以上 180 度未満に統一されました。

注意事項：

- 周期計測によってシステム内部に蓄積された計測結果は、以下の契機でクリアされます。
 - `AccelerationSensor.getData()` メソッドによって、周期計測の結果が取り出されたとき
 - 周期計測を停止した後、周期計測、加速度変化のイベント検知のいずれかを開始するか、ワンショット計測を行ったとき
 - `AccelerationSensor.disposeData()` メソッドによって明示的に計測結果のクリアが指示されるか、もしくは計測結果を蓄積したまま i アプリを終了したとき
- 周期計測を停止した後、これらの契機までは計測結果はシステム内部に蓄積されたままとなっており、`AccelerationSensor.getData()` メソッドで結果を取り出すことができます。
- 周期計測や加速度変化のイベント検知を開始している状態でサスペンド状態に遷移した場合は、周期計測や加速度変化のイベント検知は自動的に停止します。自動的に再開は行われないため、必要に応じてアプリケーションプログラム側で再開処理を行うようにしてください。その際、前項の規定により、周期計測が停止する前に蓄積された計測結果は周期計測を再開する前に取り出す必要があることに注意してください。
- 周期計測を行なっている状況では、ワンショット計測を行なったり、周期計測や加速度変化のイベント検知を新たに開始することはできません。また、加速度変化のイベント通知を行っている状況では、加速度変化のイベント検知を重ねて開始することはできません（ワンショット計測を行なったり、周期計測を新たに開始することは可能です）。
- `AccelerationData.getScreenOrientation()` メソッドや `EVENT_SCREEN_ORIENTATION` イベントで通知される「メインスクリーンの向き」とは、機体に対してディスプレイ部を普通に開状態とした状態でのメインスクリーンの向きのことを言います。機体に対してディスプレイ部を左右に折り曲げられるようなタイプの機種では、ディスプレイ部を機体と異なる向きに折り曲げている状態では、これら機能で取得できる値がディスプレイ部の向きと合致するとは限りません。
- 加速度センサーのアクセス可否状態が変化しても、周期計測の開始・停止状態は変化しません。加速度センサーにアクセスできない状態の時は周期計測中でも内部バッファに計測結果は蓄積されず、加速度センサーにアクセスできるようになると自動的に蓄積が再開されます。
- 周期計測を行っている状態で、`com.docomostar.opt.ui.Canvas2` クラス（第 5 章参照）を利用して座標系を変更した場合、蓄積された計測結果の一貫性が保証されなくなり、期待する計測結果が得られないことがあります。期待する計測結果を得るためには、一旦周期計測を停止させ蓄積された計測結果を取得してから、座標系を変更し、周期計測を再開してください。
- 周期計測や加速度変化のイベント検知と歩行方位取得機能の歩行検知計測機能は同時に利用することはできません。

第 17 章

ヘルスケア関連機能

メーカーによっては、携帯電話の付加機能として歩数計（万歩計）や身体活動量計などヘルスケア関連機能を搭載しているものがあります。このような機種のため、i アプリから携帯電話ネイティブの歩数計や身体活動量計などにアクセスするためのオプション API が定義されています。これらの機能を利用することで、i アプリから携帯電話ネイティブの歩数計や身体活動量計などが保持している情報を参照して i アプリで利用することができます。

ヘルスケア関連機能には、以下の機能があります。

- 歩数計機能 : ネイティブの歩数計（万歩計）にアクセスする機能です。
- 身体活動量計機能 : ネイティブの身体活動量計にアクセスする機能です。
- 脈拍計機能 : カメラデバイスを通して脈拍を計測する機能です。
- 身体情報アクセス機能 : ネイティブに登録された身体情報へアクセスする機能です。
- フォーム分析機能 : ネイティブのフォーム分析機能にアクセスする機能です。

注意事項：

- Star-1.1 プロファイル以降、脈拍計機能を除き、これらの機能を利用するには、ADF の **AccessUserInfo** キーにてこれらの機能を使用することを宣言（値"yes"を設定）する必要があります。また、ADF の **AppLabel** キーに利用する機能を指定する必要があります。特に歩数計機能においては、Star-1.0 以前のプロファイルではこれらの ADF 設定は不要だったため、Star-1.0 以前の i アプリを Star-1.1 対応機種上で動作させる際は注意して下さい。
詳細は 17.1 項～17.5 項の注意事項を参照して下さい。
- 身体活動量計機能は Star-1.1 プロファイルで新設されました。
- 脈拍計機能は Star-1.3 プロファイルで新設されました。
- 身体情報アクセス機能は Star-1.3 プロファイルで新設されました。
- フォーム分析機能は Star-1.5 プロファイルで新設されました。

17.1 歩数計機能

歩数計機能の API は、**com.docomostar.opt.device** パッケージで提供されます。歩数計機能は以下の 2 つのクラスで構成されます。

- **Pedometer** : 歩数計を表現するクラス
- **PedometerData** : 歩数計に蓄積されている歩数データを日毎に取りまとめて扱うためのクラス

なお、この機能では実際に歩数をカウントする機能、およびカウントした歩数データを日毎の履歴として蓄積する機能はあくまでも携帯電話ネイティブの歩数計が持っています。従って、

i アプリがこの機能を使用するためには、ユーザーが携帯電話ネイティブの歩数計をオンにしていることが前提となっています。

【Star-1.5】

Star-1.5 プロファイルにて、歩数計機能を提供する **Pedometer** クラス、**PedometerData** クラスは **com.docomostar.system** パッケージにも配置されました（**device** パッケージではなく **system** パッケージに配置された点に注意してください）。

これに伴い、**com.docomostar.opt.device** で提供される歩数計機能の API は非推奨となりました。

com.docomostar.system パッケージで提供される歩数計機能 API では、歩数データを取得する期間を指定するメソッドが新たに追加されました。新たに追加された機能を使用する場合は **com.docomostar.system** パッケージの歩数計機能 API を使用する必要があります。

【Star-2.0】

Star-2.0 プロファイルにて、携帯電話ネイティブの歩数計のオン/オフの切り替えを i アプリから制御できる機能が追加されました。なお、この機能は **com.docomostar.system** パッケージに配置された **Pedometer** クラスでのみ使用することができます。

(1) Pedometer クラスの利用

Pedometer クラスは、携帯電話ネイティブの歩数計にアクセスするためのクラスです。アプリケーションプログラムは、static メソッド **Pedometer.getPedometer()** を使用してこのクラスのオブジェクトを取得します。

Pedometer クラスは過去に蓄積された歩数データを日毎に取りまとめて保持しています。1 日分のデータが 1 つの **PedometerData** オブジェクトとして表現されており、それらを **Pedometer** クラスから取り出すためのメソッドとして以下が用意されています。

● **getData()**

ネイティブが保持している、全ての日の歩数データを **PedometerData** 配列として返します。

● **getData(int, int)**

ネイティブが保持している歩数データのうち、指定期間内のものを **PedometerData** 配列として返します。

● **getTodayData()**

ネイティブが保持している本日分の歩数データを、**PedometerData** オブジェクトとして返します。

また **Pedometer** クラスは、歩数計全体に関わる機能として、以下のようなメソッドも提供しています。

a) **getAttribute()**

歩数計の属性値を取得します。現在のプロファイルで規定されている属性には、ネイティブの歩数計のオンオフ状態を表す **DEV_PEDOMETER** があります。なお、歩数計のオンオフの切り替えを i アプリから行うことはできません。歩数計のオンオフは、ユーザーがメニュー設定等を使用して行う必要があります。

b) **getCount()**

ネイティブが保持している歩数データの数を取得します。この値は、歩数計が何日分の歩数データを保持しているかを表しています。

c) `getTotalDistance()`

ネイティブが保持している全歩数データの、累積歩行距離を取得します。距離算出に使用する歩幅の情報は、ユーザーがネイティブのメニュー設定等を使用して行います。

d) `getTotalSteps()`

ネイティブが保持している全歩数データの、累積歩数を取得します。

(2) `PedometerData` クラスの利用

`Pedometer` クラスの `getData()` メソッドや `getTodayData()` メソッドは、歩数データを `PedometerData` クラスのオブジェクト、もしくはその配列の形で返します。上述の通り、`PedometerData` オブジェクトは、1 つが 1 日分の歩数データを表します。`PedometerData.getCalendar()` メソッドを使用して、各オブジェクトがどの日に対応する歩数データを保持しているかを調べることができます。また、その日に歩数計によってカウントされた歩数は、`PedometerData.getNumberOfSteps()` メソッドを使用して取得することができます。

`PedometerData` オブジェクトからは、実際に歩数計でカウントされた歩数だけでなく、測定データから推測される参考情報として、以下のような値を取得することができます。

- ・歩行距離 (`getDistance()` メソッド)
- ・歩行により消費されたカロリー (`getConsumptionCalorie()` メソッド)
- ・歩行により燃焼された脂肪量 (`getAmountOfCombustionFat()` メソッド)
- ・歩数のうち、有酸素運動の歩行としてカウントされた歩数 (`getNumberOfAerobicsSteps()` メソッド)
- ・有酸素運動の歩行が行われた時間 (`getAerobicsTime()` メソッド)

(注) 有酸素運動の歩行とは、一定以上のテンポで一定時間以上続けて歩くことをいいます。有酸素運動の歩行は、ネイティブの歩数計により自動的に判断、カウントされます。

なお、実際の携帯電話で上記のうちどの項目をサポートするかは、メーカーの判断に委ねられます。

【Star-1.3】

Star-1.2 プロファイルまでは、`PedometerData` オブジェクトが保持している歩数データの日付情報は `Date` オブジェクトにより管理されています。

Star-1.3 プロファイル以降では日付情報の管理方法が変更されたため、`PedometerData` クラスで定義されたメソッドの内、`Date` オブジェクトを扱う `getDate()` が非推奨となり、代わりに `Calendar` オブジェクトを扱い同様の機能を持つ `getCalendar()` が新たに追加されました。

Star-1.3 プロファイル以降の携帯電話向けには、非推奨となった `getDate()` は使用せず、`getCalendar()` を使用するようして下さい。

【Star-1.5】

Star-1.5 プロファイルで `com.docomostar.system` パッケージに追加された `PedometerData` クラスは 1 日分の歩数データを一定の時間間隔毎に取りまとめて保持しています。なお、この時間間隔は機種によって異なりますが、`PedometerData.getInterval()` で取得することができます。

また、`PedometerData` クラスでは歩数データを取得するメソッドのバリエーションとして、歩数情報を取りまとめる周期を引数に持つメソッドと、歩数情報を取得する時間帯を引数に持つメソッドが追加されました。これらのバリエーションメソッドは歩数計でカウントされた歩数だけでなく、歩行距離や歩行により消費されたカロリーなどの参考情報の取得においても使用することができます。

なお、`com.docomostar.opt.device` パッケージに配置されている従来の `PedometerData` クラスには、Star-1.5 プロファイル以降もこれらのメソッドは追加されません。

【Star-2.0】

Star-2.0 プロファイルで、携帯電話ネイティブの歩数計のオン/オフの切り替えを i アプリから制御できる機能が `com.docomostar.system.Pedometer` クラスに追加されました。

i アプリから歩数計による計測を開始するには `Pedometer` クラスの `start()` メソッドを使用します。歩数計が計測停止中の状態で `start()` メソッドを実行すると、i アプリはサスペンドし、ユーザーに歩数計による計測を開始するかどうかを確認する画面が表示されます。この画面でユーザーが開始を許可すると、歩数計の計測が開始され、i アプリはレジュームし、`start()` メソッドは計測を開始したことを表す `true` を返します。歩数計が計測中の状態で `start()` メソッドを実行すると、i アプリはサスペンドすることなく `true` を返します。

i アプリから歩数計による計測を停止するには `Pedometer` クラスの `stop()` メソッドを使用します。歩数計が計測中の状態で `stop()` メソッドを実行すると、i アプリはサスペンドし、ユーザーに歩数計による計測を停止するかどうかを確認する画面が表示されます。この画面でユーザーが停止を許可すると、歩数計の計測が停止され、i アプリはレジュームし、`stop()` メソッドは計測を停止したことを表す `true` を返します。歩数計が計測停止中の状態で `stop()` メソッドを実行すると、i アプリはサスペンドすることなく `true` を返します。

なお、`start()` メソッドや `stop()` メソッドによって制御した携帯電話ネイティブの歩数計の計測状態は、i アプリが終了した後も継続されます。

注意事項：

- Star-1.0 プロファイルでは本機能を利用するにあたり特に ADF のキーを指定する必要はありませんでしたが、Star-1.1 プロファイル以降、本機能を利用するには `AccessUserInfo` キーにて歩数計機能を使用することを宣言（値"yes"を設定）する必要があります。また、ADF の `AppLabel` キーに歩数計機能を利用することを指定（値"pedometer"を設定）する必要があります。

17.2 身体活動量計機能

身体活動量とは、日々の生活の中で行われる運動の量を定量化するための考え方です。

参考ドキュメント：<http://www.mhlw.go.jp/bunya/kenkou/undou.html>

健康づくりのための運動指針 2006（エクササイズガイド 2006）

この考え方では、労働や家事、スポーツなど運動の強さを表す「運動強度（身体活動強度）」（単位 METs）、および所定の強度の運動をどの程度の時間行ったかを表す「身体活動量（エクササイズ）」（単位 Ex）という 2 つの単位を用いて身体活動の量を表現します。

例えば、1METs は「座って安静にしている状態」に相当する運動強度を表し、普通歩行は概ね 3METs に相当します。また、1 Ex は 1METs の運動を 1 時間継続した場合の身体活動量を表します。このため、普通歩行を 1 時間継続した場合の身体活動量は 3 Ex というように計算されます。

身体活動量計機能とは、携帯電話に搭載された歩数計などの各種センサーを利用して、ユーザーの日常生活における身体活動量のおおよその値を算出し、健康維持のための情報として提供するものです。

身体活動量計機能の API は、`com.docomostar.opt.device` パッケージで提供されます。身体活動量計機能は以下の 2 つのクラスで構成されます。

- `PhysicalActivityMonitor`：身体活動量計を表現するクラス
- `PhysicalActivityData`：身体活動量計に蓄積されている身体活動量データを日毎に取りまとめて扱うためのクラス

なお、この機能では実際に身体活動量を蓄積する機能はあくまでも携帯電話ネイティブの身体活動量計が持っています。従って、i アプリがこの機能を使用するためには、ユーザーが携帯電話ネイティブの身体活動量計をオンにしていることが前提となっています。

【Star-1.5】

Star-1.5 プロファイルにて、身体活動量計機能を提供する `PhysicalActivityMonitor` クラス、`PhysicalActivityData` クラスは `com.docomostar.system` パッケージにも配置されました（`device` パッケージではなく `system` パッケージに配置された点に注意してください）。

これに伴い、`com.docomostar.opt.device` で提供される身体活動量計機能の API は非推奨となりました。

`com.docomostar.system` パッケージで提供される身体活動量計機能 API と `com.docomostar.opt.device` パッケージで提供される身体活動量計機能 API とは機能的な差はありませんが、同時に利用することはできません。

また、Star-1.5 以前のプロファイルの `com.docomostar.opt.device.PhysicalActivityData` クラスで非推奨であった `getDate()` メソッド、`getExercise(Date, int)` メソッドは `com.docomostar.system.PhysicalActivityData` には存在しません。

【Star-2.0】

Star-2.0 プロファイルにて、携帯電話ネイティブの身体活動量計のオン/オフの切り替えを i アプリから制御できる機能が追加されました。なお、この機能は `com.docomostar.system` パッケージに配置された `PhysicalActivityMonitor` クラスでのみ使用することができます。

(1) `PhysicalActivityMonitor` クラスの利用

`PhysicalActivityMonitor` クラスは、携帯電話ネイティブの身体活動量計にアクセスするためのクラスです。アプリケーションプログラムは、static メソッド `PhysicalActivityMonitor.getPhysicalActivityMonitor()` を使用してこのクラスのオブジェクトを取得します。

`PhysicalActivityMonitor` クラスは過去に蓄積された身体活動量データを日毎に取りまとめて保持しています。1 日分のデータが 1 つの `PhysicalActivityData` オブジェクトとして表現されており、それらを `PhysicalActivityMonitor` クラスから取り出すためのメソッドとして `getData()` メソッドが用意されています。このメソッドはネイティブが保持している身体活動量データのうち、指定期間内のものを `PhysicalActivityData` 配列として返します。

また `PhysicalActivityMonitor` クラスは、身体活動量計全体に関わる機能として、以下のようなメソッドも提供しています。

a) `getAttribute()`

身体活動量計の属性値を取得します。現在のプロファイルで規定されている属性には、ネイティブの身体活動量計のオンオフ状態を表す `DEV_PHYSICAL_ACTIVITY_MONITOR` があります。なお、身体活動量計のオンオフの切り替えを i アプリから行うことはできません。身体活動量計のオンオフは、ユーザーがメニュー設定等を使用して行う必要があります。

b) `getCount()`

ネイティブが保持している身体活動量データの数を取得します。この値は、身体活動量計が何日分の身体活動量データを保持しているかを表しています。

c) `getMETs()`

現時点の運動強度 (METs: Metabolic Equivalents) を取得します。運動強度は身体運動時の代謝量が、安静時の何倍に相当するのかを示す尺度です。1.0 METs が安静時の代謝量を示します。

(2) PhysicalActivityData クラスの利用

PhysicalActivityMonitor クラスの `getData()` メソッドは、身体活動量データを PhysicalActivityData クラスのオブジェクト配列の形で返します。上述の通り、PhysicalActivityData オブジェクトは、1 つが 1 日分の身体活動量データを表します。このオブジェクトでは、1 日分の身体活動量を”エクササイズ”という単位で保持しています。エクササイズとは、METs(運動強度)に運動時間[h]をかけたもので、Ex と表記します。1.0METs の運動を 1.0 時間継続した場合の身体活動量は 1.0 Ex となります。PhysicalActivityData オブジェクト内部では、ある固定的な一定時間間隔ごとのエクササイズを配列として保持しています。

PhysicalActivityData オブジェクトからエクササイズを取り出すメソッドとして以下が用意されています。

- `getExercise()`

このオブジェクトが保持しているエクササイズの総計(1 日分)を取得します。

- `getExercise(Calendar, int)`

指定した時間帯(Calendar オブジェクトで指定される起点から `int` で指定される秒数の長さの時間帯)におけるエクササイズの合計を取得します。日付をまたがる指定を行った場合は、日付をまたがる直前までの時間帯が指定されたものとして合計値を計算します。

- `getExercise(int)`

このオブジェクトが保持しているエクササイズ(1 日分)に対して、アプリケーションプログラムが指定した時間間隔ごとに編成し直したエクササイズを配列として返します。

また、PhysicalActivityData オブジェクトに関する情報は、以下のメソッドを使用して取得することができます。

- `getCalendar()`

このオブジェクトがもつ日付を取得します。

- `getInterval()`

このオブジェクトが保持しているエクササイズの周期(時間間隔)を取得します。

【Star-1.2】

Star-1.1 プロファイルでは、**PhysicalActivityData** オブジェクトが保持しているエクササイズの日付情報は **Date** オブジェクトにより管理されています。

Star-1.2 プロファイル以降では日付情報の管理方法が変更されたため、**PhysicalActivityData** クラスで定義されたメソッドの内、**Date** オブジェクトを扱うメソッドが非推奨となり、代わりに **Calendar** オブジェクトを扱う同様の機能を持つメソッドが Star-1.2 プロファイルに追加されました。非推奨になったメソッドと、同様機能を持つ追加メソッドとの対応関係（非推奨となったメソッド→対応する追加メソッド）は以下のようになります。

```
getDate()                → getCalendar()
getExercise(Date, int)   → getExercise(Calendar, int)
```

Star-1.2 プロファイル以降の携帯電話向けには、上記の対応関係に従って非推奨となったメソッドは使用せず、追加されたメソッドを使用するようにして下さい。

また Star-1.2 プロファイルでは、**PhysicalActivityData** オブジェクトが保持しているエクササイズの内、「有酸素運動によって生じたエクササイズ」といった抽出条件を指定し、その条件を満たすものをフィルタリングして取得することができます。

このように特定の抽出条件を満たすエクササイズを取得するメソッドとして以下が追加されています。これらメソッドの **String** 型の引数には、後述する **FILTER_NONE** や **FILTER_AEROBICS** といった定数を抽出条件として指定します。

- **getExercise(String)**

このオブジェクトが保持しているエクササイズ(1 日分)の内、**String** で指定される抽出条件を満たすエクササイズの総計(1 日分)を取得します。

- **getExercise(Calendar, int, String)**

指定した時間帯 (**Calendar** オブジェクトで指定される起点から **int** で指定される秒数の長さの時間帯) におけるエクササイズの中で、**String** で指定される抽出条件を満たすエクササイズの合計を取得します。日付をまたがる指定を行った場合は、日付をまたがる直前までの時間帯が指定されたものとして合計値を計算します。

- **getExercise(int, String)**

このオブジェクトが保持しているエクササイズ(1 日分)の内、**String** で指定される抽出条件を満たすエクササイズに対して、**int** で指定される時間間隔ごとに編成し直したエクササイズを配列として返します。

上記のメソッドに指定する抽出条件として以下の定数が定義されています。

- ・ **FILTER_NONE**

抽出条件がないことを表します。この抽出条件を指定した場合は、オブジェクトが保持しているエクササイズ全体が抽出の対象となります。

- ・ **FILTER_AEROBICS**

オブジェクトが保持しているエクササイズの内、有酸素運動によって生じたエクササイズを抽出することを表します。

なお、**getExercise(String)**等に **FILTER_NONE** 以外の抽出条件を指定して特定のエクササイズが取得できるかどうかはメーカーによって異なります。携帯電話がどの抽出条件をサポートしているかは、**PhysicalActivityData.getAvailableFilter()**により調べることが出来ます。

【Star-1.5】

com.docomostar.system.PhysicalActivityMonitor オブジェクトと **com.docomostar.opt.device.PhysicalActivityMonitor** オブジェクトを同時に取得することはできません。一方のクラスで **getPhysicalActivityMonitor()**を呼び出しオブジェクトを取得した後、他方のクラスで **getPhysicalActivityMonitor()**を呼び出すと例外が発生します。

【Star-2.0】

Star-2.0 プロファイルで、携帯電話ネイティブの身体活動量計のオン/オフの切り替えを i アプリから制御できる機能が `com.docomostar.system.PhysicalActivityMonitor` クラスに追加されました。

i アプリから身体活動量計による計測を開始するには `PhysicalActivityMonitor` クラスの `start()` メソッドを使用します。身体活動量計が計測停止中の状態で `start()` メソッドを実行すると、i アプリはサスペンドし、ユーザーに身体活動量計による計測を開始するかどうかを確認する画面が表示されます。この画面でユーザーが開始を許可すると、身体活動量計の計測が開始され、i アプリはレジュームし、`start()` メソッドは計測を開始したことを表す `true` を返します。身体活動量計が計測中の状態で `start()` メソッドを実行すると、i アプリはサスペンドすることなく `true` を返します。

i アプリから身体活動量計による計測を停止するには `PhysicalActivityMonitor` クラスの `stop()` メソッドを使用します。身体活動量計が計測中の状態で `stop()` メソッドを実行すると、i アプリはサスペンドし、ユーザーに身体活動量計による計測を停止するかどうかを確認する画面が表示されます。この画面でユーザーが停止を許可すると、身体活動量計の計測が停止され、i アプリはレジュームし、`stop()` メソッドは計測を停止したことを表す `true` を返します。身体活動量計が計測停止中の状態で `stop()` メソッドを実行すると、i アプリはサスペンドすることなく `true` を返します。

なお、`start()` メソッドや `stop()` メソッドによって制御した携帯電話ネイティブの身体活動量計の計測状態は、i アプリが終了した後も継続されます。

注意事項：

- 本機能を利用するには ADF の `AccessUserInfo` キーにて身体活動量計機能を使用することを宣言（値 "yes" を設定）する必要があります。また、ADF の `AppLabel` キーに身体活動量計機能を利用することを指定（値 "physicalactivitymonitor" を設定）する必要があります。
- `PhysicalActivityData` オブジェクトが保持しているエクササイズの周期（時間間隔）は、メーカーにより異なります。また、同じ携帯電話であってもこのオブジェクトが何日前のデータを保持しているかによって異なる場合があります。どの日付のオブジェクトでどのような周期となるかは、メーカーの設計に委ねられます。
- 携帯電話の中には、抽出条件として `PhysicalActivityData.FILTER_AEROBICS` を指定して有酸素運動によって生じたエクササイズを抽出できるものがあります。どのような基準で有酸素運動によって生じたエクササイズと判断するかは、メーカーの設計に委ねられます。

17.3 脈拍計機能

脈拍計機能の API は、`com.docomostar.opt.device.Pulsemeter` インタフェースの実装クラスで提供されます。`Pulsemeter` インタフェースは一般的、汎用的な脈拍計機能を定義し、実装クラス側で機能の実装方式を加味した具体的な利用方法を定義しています。現在のプロファイルでは、実装クラスとして、携帯電話のカメラを使用して脈拍を測定する `com.docomostar.opt.device.CameraPulsemeter` が提供されています。本項では、`CameraPulsemeter` を中心に解説します。

`CameraPulsemeter` を利用した脈拍計機能では、カメラデバイスに指を当てて撮影することによって脈拍の測定を行います。携帯電話システムは、カメラデバイスから取り込まれる指の画像をリアルタイムに解析して、脈拍データを取得します。

脈拍を測定するには、まず最初に `CameraPulsemeter` クラスの `static` メソッド `getPulsemeter()` を使用して、`CameraPulsemeter` オブジェクトを取得します。このメソッドの引数には、使用するカメラデバイスのカメラ ID (`Camera.getCamera()` メソッドでも使用する、カメラデバイスを特定するための ID) を指定します。

取得した **CameraPulsemeter** オブジェクトに対して、**start()** メソッドを呼び出すと脈拍の測定を開始します。**start()** メソッド自体は脈拍の測定を開始するとすぐにアプリケーションプログラムに復帰し、以降の測定はアプリケーションプログラムとは非同期で行われます。この状態でカメラデバイスを指に当て、しばらく静止状態を保つことで測定結果が得られるようになります。

測定結果を取得するには、**getPulserate()** メソッドを使用します。このメソッドは、測定結果が得られるまでは戻り値として -1 を返し、測定結果が得られたらそれを 1 分あたりの脈拍数に換算して返します。なお、**start()** メソッドで測定を開始しても、指が画角内に適切に収まっていなかったり、光量の過不足などの要因で測定が正しく行えていない場合もあります。

CameraPulsemeter クラスには、このような状況を把握するために **getStatus()** メソッドが用意されています。脈拍の測定を開始したら、適宜このメソッドを呼び出して、測定が正常に行われているかどうかを確認するようにしてください。

測定が終了したら、**stop()** メソッドを呼び出してカメラデバイスを停止させてください。

以下に、脈拍計機能を使用するコード例を示します。

例: 脈拍計機能の利用

```
try {
    CameraPulsemeter pm = CameraPulsemeter.getPulsemeter(0); // 脈拍計オブジェクト
                                                                // を取得 (カメラ ID:0)

    int result;
    pm.start(); // 測定を開始する
    while (true) {
        try { Thread.sleep(1000); } catch (InterruptedException ie) {}
        result = pm.getPulserate(); // 測定結果を得る
        if (result > 0) break; // 結果が負でなければ測定完了
        int state = pm.getStatus(); // 測定状況を確認する
        if (state == CameraPulsemeter.STATUS_CORRECT_OPERATION) { // 正常に測定中
        } else if (state == CameraPulsemeter.STATUS_NO_FINGER) { // 指が写されて
                                                                // いない

            // ユーザーに、カメラに指を写すよう促す必要がある
        } else if (state == CameraPulsemeter.STATUS_DARK) { // 光量が少なすぎる
            // もう少し明るいところで測定するよう促す必要がある
        } else if ... // 以降省略

            // 他のステータスも同様に判定する STATUS_STOP の場合はサスペンドなどの
            // 要因により測定が停止しているので、処理を打ち切る必要がある
        }
    }
    pm.stop(); // 測定を終了する
} catch (DeviceException de) {
    // カメラデバイスの競合やサスペンドなどが発生した場合にこの例外が発生する可能性がある
}
```

注意事項：

- **CameraPulsemeter** クラスによる脈拍計機能はカメラデバイスを使用するため、カメラデバイスを使用する他の機能と同時に使用することはできません。
- **start()**メソッドにより開始された測定は、i アプリがサスペンド状態に遷移すると自動的に停止します。

17.4 身体情報アクセス機能

歩数計や身体活動量計などの機能をサポートする携帯電話の中には、ユーザーの身長や体重といった身体情報をネイティブに保持しておき、それらを利用して消費カロリーや燃焼脂肪量などの参考情報をユーザーに提供できるものがあります。

身体情報アクセス機能では、携帯電話ネイティブが保持している身体情報の設定と参照を行うための機能を提供します。

この機能のAPIは、**com.docomostar.opt.system.BodyProfiles** クラスで提供されます。現在のプロファイルでは、身体情報として身長と体重の情報にアクセスするためのAPIが提供されています。

BodyProfiles クラスには、身体情報の各項目の設定メソッド (**setBodyHeight()**、**setBodyWeight()**) と参照メソッド (**getBodyHeight()**、**getBodyWeight()**) が用意されています。また、各項目に設定可能な最小値、最大値や分解能（設定値の刻み幅）を調べるための **getAttribute()**メソッドも用意されています。

各々のメソッドの詳細はAPI リファレンスを参照してください。

注意事項：

- 本機能を利用するには ADF の **AccessUserInfo** キーにて身体情報アクセス機能を使用することを宣言（値"yes"を設定）する必要があります。また、ADF の **AppLabel** キーに身体情報アクセス機能を利用することを指定（値"bodyprofiles"を設定）する必要があります。
- 本機能をサポートしている端末の中には、身体情報の項目のうち、特定の項目のみサポートしているものがあります。

17.5 フォーム分析機能

フォーム分析機能とは、携帯電話に搭載された各種センサーを利用して、ユーザーの身体の動きに関する情報を提供する機能です。フォーム分析機能には、ランニング中のユーザーの身体の動きの分析を行うランニングフォーム診断モードと、歩行中のユーザーの身体の動きの分析を行うウォーキングフォーム診断モードとがあります。この機能を利用することで、ユーザーの姿勢のバランスや歩行運動の詳細情報などを取得することができます。

フォーム分析機能のAPIは`com.docomostar.opt.system.formanalyzer`パッケージで提供されます。

このパッケージでは携帯電話ネイティブのフォーム分析機能にアクセスするための **FormAnalyzer** クラス、フォーム分析機能が蓄積するデータを表す **FormData** クラス、およびフォーム分析機能により得られるバランスや歩行情報など個別測定項目ごとの詳細情報を保持する各クラスが定義されています。フォーム分析機能を構成するクラスの機能概要は以下のとおりです。

クラス	機能概要
FormAnalyzer	フォーム分析機能にアクセスするクラスです。ネイティブのフォーム分析機能の開始や停止、蓄積された計測データの取得機能を提供します。
FormData	フォーム分析機能により取得したデータを表すクラスです。フォーム分析機能で計測したデータを保持しています。
BodyTrunkRotation	フォーム分析機能により取得した体幹回転に関するデータを表すクラスです。詳細情報として、腰の左右の回転量を取得することができます。
FrontRearBalance	フォーム分析機能により取得した前後バランスに関するデータを表すクラスです。詳細情報として、前後の重心の偏りや歩行時の前後方向の移動量や回転量を取得することができます。
LandingImpact	フォーム分析機能により取得した着地衝撃に関するデータを表すクラスです。詳細情報として左右の衝撃度を取得することができます。
SideBalance	フォーム分析機能により取得した左右バランスに関するデータを表すクラスです。詳細情報として左右の重心の偏りや左右方向の移動量や回転量を取得することができます。
UpdownBalance	フォーム分析機能により取得した上下バランスに関するデータを表すクラスです。詳細情報として歩行時の上下方向の移動量を取得することができます。
WalkingInfo	フォーム分析機能により取得した歩行情報に関するデータを表すクラスです。詳細情報として歩行距離や歩行速度、歩幅などの情報を取得することができます。

(1) FormAnalyzer クラスの利用

FormAnalyzer クラスは、携帯電話ネイティブのフォーム分析機能にアクセスするためのクラスです。アプリケーションプログラムは **static** メソッド **FormAnalyzer.getFormAnalyzer()** を使用してこのクラスのオブジェクトを取得します。**getFormAnalyzer()** メソッドの引数には、診断モード (**FormAnalyzer** クラスの **MODE** 定数) を指定します。

フォーム分析を開始するには **FormAnalyzer.start()** メソッドを呼び出します。**start()** メソッドには、計測中にサスペンドが発生した場合に、フォーム分析を継続するかどうかを指定します。**true** を設定しておくことで、サスペンド中でもフォーム分析を継続することができます。開始されたフォーム分析は **FormAnalyzer.stop()** メソッドを呼び出すと停止します。

フォーム分析を開始すると、携帯電話ネイティブのフォーム分析機能は一定時間ごとにフォーム分析の計測を行い、その結果がシステム内部のバッファに蓄積されていきます。蓄積された計測データを取得するには、**FormAnalyzer.getData()** メソッドを使用します。

FormAnalyzer.getData() メソッドは、**FormData** オブジェクトの形で、ネイティブのフォーム分析機能が保持している計測データを返します。**getData()** メソッドの引数には、取得したい計測データの計測開始日時と終了日時を指定します。開始日時および終了日時には **null** を指定することもでき、その場合は指定なしとして扱われます。開始日時および終了日時双方に **null** を指定するとネイティブのフォーム分析機能が保持している全ての計測データを取得することができます。

バッファに蓄積された計測データを削除するには **disposeData()** メソッドを呼び出します。なお、バッファに保持可能な計測データ数を超えてフォーム分析が続けられた場合は、バッファ内の古いデータから順次上書きされていきます。

なお、サポートされる診断モードは機種によって異なります。携帯電話がどの診断モードをサポートしているかは、`FormAnalyzer.getAvailableMode()` メソッドを使用して調べることができます。

(2) FormData クラスの利用

`FormAnalyzer` クラスの `getData()` メソッドは、計測データを `FormData` クラスのオブジェクトとして返します。`FormData` オブジェクトは、左右バランスや上下バランスといったデータ項目や、このオブジェクトが保持する最古の計測データの計測開始日時や最新の計測データの計測終了日時を保持しています。

`FormData` オブジェクトから計測データを取り出すメソッドとして以下が用意されています。

- `getSideBalance()`
左右バランスを求めるためのデータを取得します。`SideBalance` オブジェクトを返します。
- `getUpdownBalance()`
上下バランスを求めるためのデータを取得します。`UpdownBalance` オブジェクトを返します。
- `getFrontRearBalance()`
前後バランスを求めるためのデータを取得します。`FrontRearBalance` オブジェクトを返します。
- `getBodyTrunkRotation()`
体幹回転を求めるためのデータを取得します。`BodyTrunkRotation` オブジェクトを返します。
- `getWalkingInfo()`
歩行情報を求めるためのデータを取得します。`WalkingInfo` オブジェクトを返します。
- `getLandingImpact()`
着地衝撃を求めるためのデータを取得します。`LandingImpact` オブジェクトを返します。
- `getOptionData()`
オプションデータを取得します。なお、このメソッドのサポート有無、およびサポートされる場合の値の意味は機種によって異なります。

`FormData` オブジェクトに関する情報は以下のメソッドを使用して取得することができます。

- `getAvailableDataType()`
携帯電話がサポートしている計測データの種類を取得します。
- `getStartTime()`
`FormData` オブジェクトから得られる計測データのうち、最古の計測データの計測開始日時を取得します。
- `getEndTime()`
`FormData` オブジェクトから得られる計測データのうち、最新の計測データの計測終了日時を取得します。

`FormData` オブジェクトから得られる `SideBalance`、`UpdownBalance`、`FrontRearBalance`、`BodyTrunkRotation`、`WalkingInfo`、`LandingImpact` の各オブジェクトには、各オブジェクトが保持する詳細データを取得するための `getData()` メソッドが定義されています。引数には取得する詳細情報の種類を指定します。利用可能なオブジェクト種別や各オブジェクトから取得可能な詳細情報の種類は機種によって、もしくは診断モードによって異なります。指定可能な詳細情報の種類は各オブジェクトの `getAvailableDataType()` メソッドにより調べることができます。

注意事項：

- 本機能を利用するには ADF の `AccessUserInfo` キーにてフォーム分析機能を使用することを宣言（値 "yes" を設定）する必要があります。また、ADF の `AppLabel` キーにフォーム分析機能を利用することを指定（値 "formanalyzer" を設定）する必要があります。
- 本機能をサポートする端末の中には、フォーム分析による計測データのうち、特定の項目のみサポートしているものがあります。

第 18 章

タッチパネルデバイス

タッチパネルデバイスは、指で触れたり離したり、触れた状態で移動させることで、選択操作やカーソル移動が可能になる入力デバイスです。タッチパネルデバイスを利用すると、タッチパネルに触れる操作（タッチ操作）、離す操作（リリース操作）、触れた状態で指を移動させる操作（ムーブ操作）などのユーザー操作を i アプリに伝えることができます。タッチパネルデバイスの機能は、`com.docomostar.opt.ui.TouchDevice` クラスで提供されます。なお、本項で解説するタッチパネルデバイス機能はカレントフレームに `Canvas` が設定されている場合のみ使用することができます。

【Star-2.0】

Star-2.0 プロファイルにて、タッチパネルデバイス機能を提供する `TouchDevice` クラスは `com.docomostar.device` パッケージにも配置されました（`ui` パッケージではなく `device` パッケージに配置された点に注意してください）。また、タッチパネル操作イベントを表す `TouchEventObject` クラスが `com.docomostar.ui` パッケージに追加されました。

これに伴い `com.docomostar.opt.ui.TouchDevice` クラスは非推奨になりました。

タッチパネルデバイスは通常は無効化されています。タッチパネルデバイスの有効、無効を切替えるには、`TouchDevice.setEnabled()` メソッドを使用します。

`TouchDevice.isEnabled()` メソッドにより、現在タッチパネルデバイスが有効であるか否かを確認することもできます。また、タッチパネルデバイスはユーザーの設定により使用が禁止されている場合があります。ユーザー設定によりタッチパネルデバイスの使用が禁止されている場合は i アプリからもタッチパネルデバイスを使用することはできませんが、この設定状況は `TouchDevice.isAvailable()` メソッドで確認することができます。

タッチパネルデバイスを利用できる状態では、アプリケーションプログラムはユーザーがタッチパネルに行った操作をイベントの通知により知ることができます。アプリケーションプログラムが取得可能なタッチパネルの操作には

- タッチ操作
- ムーブ操作（開始）
- ムーブ操作（終了）
- リリース操作
- 2 本指を広げる操作
- 2 本指をつまむ操作

があります（2本指を広げる操作、つまむ操作はタッチパネルデバイスをサポートする機種の中でもオプションの機能です）。タッチパネルデバイスを利用できる状態でこれらの操作が行なわれると、タッチパネル操作イベントがカレントフレームの**Canvas**の**processEvent()**メソッドに通知されるようになります。これらの操作に対応するイベント種別の定数は**com.docomostar.ui.Display**クラスに定義されています。

タッチ、ムーブ、リリースの各操作のイベントは原則としてタッチ操作、リリース操作の順に発生し、ムーブ操作が行われた場合はタッチ操作イベントとリリース操作イベントの間でムーブ操作（開始）イベントとムーブ操作（終了）イベントが発生します。ただしユーザーがタッチパネルに触れた状態でカレントフレームが**Canvas**以外に切り替わると、その後のムーブ操作やリリース操作のイベントはiアプリに通知されません。また、ユーザーがタッチパネルに触れた状態でカレントフレームが**Canvas**に切り替わっても、その後のムーブ操作やリリース操作のイベントはiアプリに通知されません。この場合では、リリース操作後に新たにタッチ操作が行われるとタッチ操作イベントからiアプリに通知されます。

2本指を広げる操作、つまむ操作のイベントは、それらの操作が行われている間、一定の間ごと、もしくは一定の移動距離ごとに通知されます。その際のイベント通知間隔は、メーカーにより異なります。

タッチパネル操作イベントの通知を受けて**TouchEvent.getX()**メソッドおよび**TouchEvent.getY()**メソッドを呼び出すことで、ユーザーが触れているタッチパネル上の位置を取得することができます。これらのメソッドで取得されるタッチパネル上の位置は画面の左上を原点とした絶対座標となります。なお、タッチパネルにタッチしていない状態では、これらメソッドは直前にリリース操作が行なわれた座標を返します。

注意事項：

- 2本指を広げる操作、つまむ操作として処理される場合を除き、タッチパネル上の複数個所に指が触れていた場合の振る舞いはメーカーによって異なります。
- タッチパネルの操作と同時にポインティングデバイスをマウスモードで操作した場合の振る舞いはメーカーによって異なります。
- **com.docomostar.opt.ui.Canvas2** クラス（第5章参照）で座標系を制御した場合、タッチパネルデバイスで扱う座標系に影響を与えるかどうかはメーカーにより異なります。
- メーカーによっては、カレントフレームが**Panel**の場合にもタッチパネルを使用出来る場合があります。ただしその場合は、携帯電話システムがタッチパネル操作イベントを自動的に高レベルイベントに変換するため、iアプリ側ではタッチパネルのための特別な処理を実装する必要はありません。
- 2本指を広げる操作、つまむ操作を行った場合に**getX()**、**getY()**の各メソッドがどのような座標を返すかはメーカーにより異なります。

【Star-1.1】

2本の指を広げる操作、つまむ操作に関する規定は、**Star-1.1** プロファイルにて追加されました。また、iアプリからサスペンドメソッドを呼び出してサスペンドした状態でユーザーがタッチ操作を行うと、iアプリがレジュームします。タッチパネルデバイスが無効である状態でユーザーがタッチ操作を行った場合にレジュームするかどうかは、メーカーにより異なります。

【Star-2.0】

Star-2.0 プロファイルで追加されたタッチパネルデバイス機能では、タッチパネルデバイスを利用できる状態でタッチパネル操作が行われると、タッチパネル操作イベントがカレントフレームの **Canvas** の **onTouchEvent()** メソッドに通知されるようになります。このとき、**onTouchEvent()** メソッドの引数には行われたタッチパネル操作イベントに対応する **TouchEventObject** オブジェクト設定されています。**TouchEventObject** オブジェクトには **getType()** メソッドが定義されており、このメソッドから行われたタッチパネル操作の種別を取得することができます。

アプリケーションプログラムで取得可能なタッチパネルの操作には

- タッチ操作
- ムーブ操作
- リリース操作

があります。

タッチ、ムーブ、リリースの各操作のイベントはタッチ操作、リリース操作の順に発生し、ムーブ操作が行われた場合はタッチ操作イベントとリリース操作イベントの間でムーブ操作イベントが発生します。ムーブ操作イベントは、ムーブ操作が行われている間、一定の時間ごと、もしくは一定の移動距離ごとに通知されます。その際のイベント通知間隔はメーカーによって異なります。

ユーザーがタッチパネルに触れた状態でカレントフレームが **Canvas** 以外に切り替わると、その後のムーブ操作やリリース操作は **i** アプリに通知されません。またユーザーがタッチパネルに触れた状態でカレントフレームが **Canvas** に切り替わっても、その後のムーブ操作やリリース操作のイベントは **i** アプリに通知されません。この場合では、リリース操作後に新たにタッチ操作が行われるとタッチ操作イベントから **i** アプリに通知されます。

TouchEventObject オブジェクトの **getX()** メソッドおよび **getY()** メソッドを呼び出すことで、ユーザーが触れているタッチパネル上の位置を取得することができます。これらのメソッドで取得されるタッチパネル上の位置は画面の左上を原点とした絶対座標となります。

また、タッチパネル操作イベントに対応する点には固有の **ID** が割り振られており、この **ID** は **TouchEventObject** オブジェクトの **getPointerId()** メソッドで取得することができます。この **ID** はタッチ、ムーブ、リリースの一連の操作中は同じ値のため、アプリケーションプログラムはどの点がどのように動いたかを追跡することができます。なお、この **ID** はタッチ操作イベントが発生した時点で新たに割り当てられ、リリース操作イベントが発生した時点でクリアされます。

注意事項：

- **com.docomostar.device.TouchDevice** クラスの **isEnabled()** メソッドが **true** を返す状態では、**com.docomostar.opt.ui.TouchDevice** クラスの **isEnabled()** メソッドが **true** を返す状態であっても、タッチパネル操作のイベントは常に **Canvas.onTouchEvent()** メソッドに通知され、**Canvas.processEvent()** メソッドに通知されることはありません。また、**com.docomostar.opt.ui.TouchDevice** クラスの **getX()**、**getY()** の各メソッドは負の値を返します。

第 19 章

シンセサイザ操作機能

i アプリでは、アプリケーションプログラムから直接音源デバイスに対してメッセージを発行し、リアルタイム発音を行なわせることができるシンセサイザ操作機能をオプション API として規定しています。シンセサイザ操作機能を利用すると、**AudioPresenter** による MFi 形式データや MIDI 形式データの再生とは異なり、アプリケーションプログラムが任意のタイミングで特定の音色の発音を行なったり、リアルタイムにピッチベンドなどの効果を指示することができるようになります。また、シンセサイザ操作機能では UCS (User Customized Sound) データの利用もサポートしており、音源デバイスにプリセットされた音色だけでなくコンテンツ開発者が独自に作成した音色を使用することもできます。

シンセサイザ操作機能は `com.docomostar.opt.ui.Synthesizer` クラスで提供されます。**Synthesizer** クラスは **MediaPresenter** インタフェースを実装していますが、その機能はシンセサイザ操作機能に特化しており、他の **MediaPresenter** インタフェースを実装したクラスに比べていくつかの機能が制限されています。

i アプリが **Synthesizer** オブジェクトを取得するには、**Synthesizer** クラスの static メソッドである `getSynthesizer()` メソッドを呼び出します。シンセサイザ操作機能 (**Synthesizer** オブジェクト) は **AudioPresenter** と同時に利用することはできますが、複数の **Synthesizer** オブジェクトを同時に利用することはできません。また、ポート番号の引数を持つ `getSynthesizer()` メソッドは、ポート番号として 0 のみが指定可能です。

以下、**Synthesizer** オブジェクトを使用して音源デバイスにメッセージを発行する方法と、UCS データを用いた発音を行なう方法について解説します。

【Star-1.5】

Star-1.5 プロファイルにて、シンセサイザ操作機能を提供する **Synthesizer** クラスは `com.docomostar.ui` パッケージにも配置されました。

これに伴い、`com.docomostar.opt.ui.Synthesizer` クラスは非推奨になりました。これらのクラスに機能的な差はありませんが、同時に利用することはできません。

19.1 音源デバイスへのメッセージの発行

Synthesizer オブジェクトに対して `play()` メソッドを呼び出すと、音源デバイスはメッセージ受付中状態になります。i アプリは、シンセサイザ操作機能をメッセージ受付中状態とした上で、`sendMessage()` メソッドを利用して音源デバイスにメッセージを発行します。

sendMessage() メソッドには、**byte** 配列の形でメッセージを指定します。1 つのメッセージは 5 バイトで構成され、以下のようにデータを格納します。

- 1 バイト目：コマンド種別 (**Synthesizer** クラスの **COMMAND** 定数で定義されたメッセージコマンドの値)
- 2 バイト目：チャンネル指定
- 3 バイト目：データ 0 値 (コマンド種別毎に規定されているパラメータ)
- 4 バイト目：データ 1 値 (コマンド種別毎に規定されているパラメータ)
- 5 バイト目：データ 2 値 (コマンド種別毎に規定されているパラメータ)

コマンド種別は全てのメッセージで指定しますが、その他のデータはコマンド種別毎に指定可否やデータの意味が異なります。各コマンドで指定するデータの詳細は、API リファレンスを参照してください。

アプリケーションプログラムは、1 回の **sendMessage()** メソッド呼び出しで複数のメッセージを発行することもできます。1 つのメッセージを構成するデータは 5 バイト固定であるため、複数のメッセージを発行する場合はメッセージ数×5 バイトのデータを用意します。なお、1 回の **sendMessage()** メソッド呼び出しで発行可能なメッセージ数の上限は、**getAvailableMessageNum()** メソッドで調べることができます。

メッセージ受付を終了するには、**stop()** メソッドを呼び出します。

注意事項：

- 音源デバイスへ発行するメッセージのフォーマットは **MIDI** メッセージとは異なります。各メッセージのフォーマットの詳細は API リファレンスを参照してください。
- **Synthesizer** クラスは **MediaPresenter** インタフェースを実装していますが、**getMediaResource()** メソッド、および **setMediaListener()** メソッドは使用することができません。また、**setAttribute()** メソッドによって制御できる属性は、**AudioPresenter.SET_VOLUME** と **AudioPresenter.PRIORITY** のみです。これら以外の属性を指定しても属性の設定は行われません。
- **Synthesizer** オブジェクトの発音の優先度は、初期状態では **AudioPresenter.NORM_PRIORITY** です。なお、優先度は **Synthesizer.setAttribute()** メソッドで変更することができます。
- 音源デバイスがメッセージ受付中状態で i アプリがサスペンド状態になった場合、音源デバイスはメッセージ受付中状態から解除されます。i アプリがレジュームしても自動的にメッセージ受付中状態にならないため、必要に応じて明示的に **play()** メソッドを呼び出して受付中状態にする必要があります。
- **Synthesizer** オブジェクトを複数並列に同時再生することはできません。**AudioPresenter** オブジェクトとの同時再生はサポートされていますが、**AudioPresenter** オブジェクト同士に比べて同時再生数に厳しい制限を持つ場合があります。**AudioTrackPresenter** オブジェクトとの同時再生については、メーカーによってはサポートされません。
- 音色バンク変更コマンドのメッセージを発行する際、バンク番号には **Synthesizer** クラスに定義されている **BANK** 定数を指定します。

19.2 UCS データによる発音

音源デバイスはデフォルト状態では音源デバイスのプリセット音色を使用しますが、音源デバイスに任意の UCS データを登録し、ノートメッセージで発音させることも可能です。音源デバイスに UCS データを登録するには、`play()` メソッドを呼び出す前（メッセージ受付中でない状態）に `Synthesizer` クラスの static メソッド `setUCSData()` を呼び出します。

`setUCSData()` メソッドには、UCS データをバイト配列の形で指定します。`setUCSData()` メソッドでは複数の UCS データを一度に登録することができるため、実際には UCS データを渡すメソッド引数の型は `byte[][]` となっています。

UCS データは、音源デバイスに応じたオーサリングツールによって作成したデータです。1 つの UCS の音色はメインボイスとサブボイスから構成されていますが、シンセサイザ操作機能では、1 つの UCS データ（1 つの `byte` 配列）には 1 つのボイスしか含めることができません。このため、`setUCSData()` メソッドではメインボイスとサブボイスは各々別の UCS データとして取り扱い、メインボイスとサブボイス（メインボイスの UCS データとサブボイスの UCS データ）の依存関係を別の引数で指定するようになっています。

UCS データの配列 (`byte[][]` 型)

UCS データを指定します。この引数は複数の UCS データを指定できるようにするために、`byte` の二次元配列となっています。メインボイスとサブボイスから構成されている UCS 音色は、個々のボイスを別々の UCS データとして指定します。

一度登録した UCS データをクリアするには、この引数に `null` を指定します。

サブボイス番号の配列 (`byte[]` 型)

第 1 引数で指定した各 UCS データのメインボイス、サブボイスの依存関係を指定します。第 1 引数で指定した UCS データ毎に、そのボイスのサブボイスを UCS データ番号（UCS データの配列インデックス）で指定します。例えば、第 1 引数の配列 `m` 番目のボイス（メインボイス）に `n` 番目の UCS データをサブボイスとして関連付ける場合、サブボイス番号の配列番号 `m` 番目に `n` を指定します。なお、ボイスに関連付けるサブボイスが不要な場合（そのボイス自体がサブボイスである場合など）には、サブボイス番号として `SUBVOICE_NOT_EXIST` を指定するか、そのボイスのインデックス番号と同じ UCS データ番号を指定します。

一度登録した UCS データをクリアする場合は、この引数にも `null` を指定します。

例：UCS データの設定方法

```
byte[] ucsData0 = ...;    // メインボイス UCS データ 0
byte[] ucsData1 = ...;    // メインボイス UCS データ 1
byte[] ucsData2 = ...;    // メインボイス UCS データ 2
byte[] ucsData3 = ...;    // メインボイス UCS データ 3
// UCS データを配列で設定する
byte[][] ucsDataArray = new byte[][]{ucsData0, ucsData1, ucsData2, ucsData3};
```

```
// サブボイス番号を設定する
// メインボイス UCS データ 0 : サブボイスとして UCS データ 1 を関連付ける
// メインボイス UCS データ 1 : サブボイス不要(SUBVOICE_NOT_EXIST 指定する方法)
// メインボイス UCS データ 2 : サブボイスとして UCS データ 3 を関連付ける
// メインボイス UCS データ 3 : サブボイス不要(メインボイスの INDEX 番号と同じ UCS データ番号を指定する方法)

byte[] subVoiceNum = new byte[]{1, Synthesizer.SUBVOICE_NOT_EXIST, 3, 3};

// UCS データを音源デバイスに登録する
Synthesizer.setUCSData(ucsDataArray, subVoiceNum);

Synthesizer synthesizer = Synthesizer.getSynthesizer();
// メッセージ受付開始
synthesizer.play();
//メッセージの発行
synthesizer.sendMessage(...);

// 以下省略
```

なお、**setUCSData()**メソッドで設定可能な UCS データ数の上限、UCS データに含まれる Wave データの合計サイズの上限は携帯電話によって異なります。これらの上限値は、それぞれ **getAvailableUCSDataNum()**、**getAvailableUCSDataSize()**メソッドで取得することができます。

注意事項：

- 音源デバイスへ UCS を設定するには、携帯電話の音源デバイスに対応した UCS データを作成するオーサリングツールが必要です。NTT ドコモでは、UCS データを作成するオーサリングツールは一般公開していません。

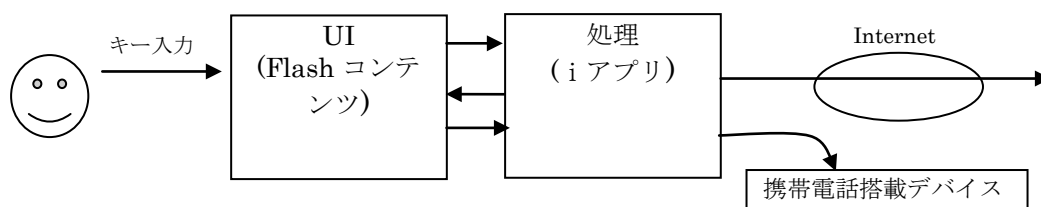
第 20 章

i アプリ-Flash 連携機能

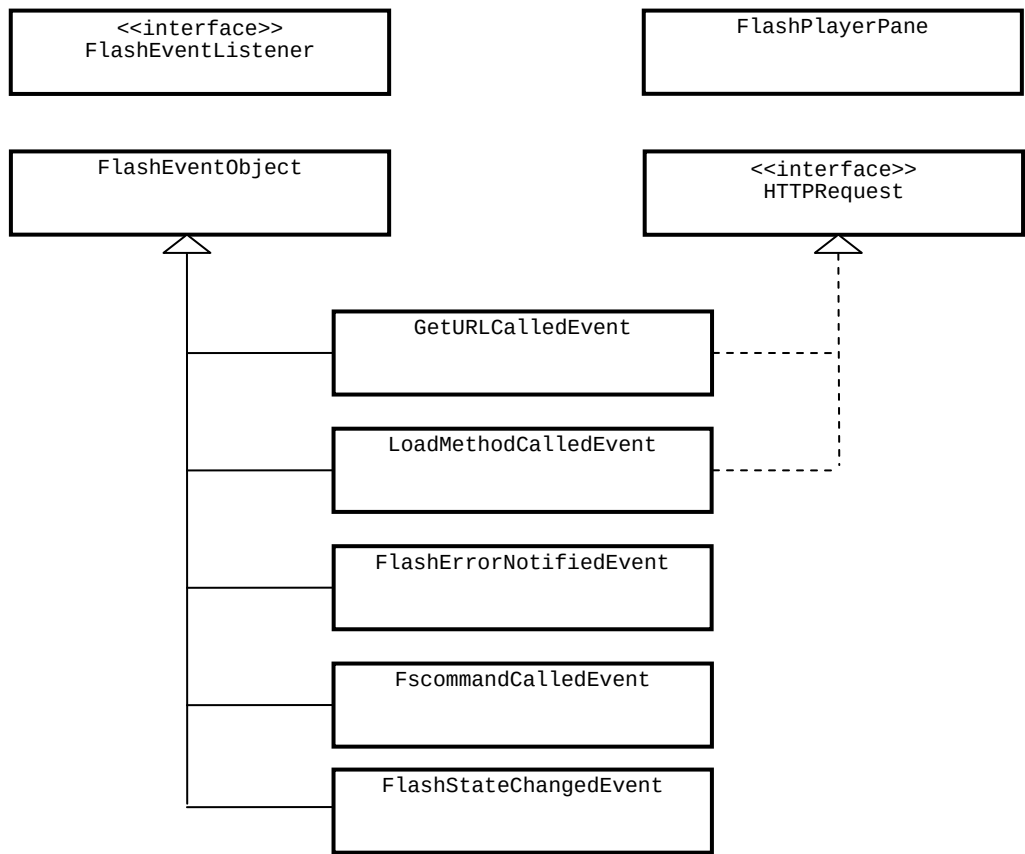
本章では i アプリ-Flash 連携機能について説明します。i アプリ-Flash 連携機能とは、i アプリから Flash プレイヤを呼び出し、Flash コンテンツ (SWF ファイル) を再生するというものです。i アプリ-Flash 連携機能を利用すると Flash コンテンツと i アプリの間でデータのやり取りを行うことができ、Flash コンテンツを i アプリのリッチなユーザーインターフェースとして組み合わせたり、Flash のミニゲームに対してより強力な i アプリの機能を組み合わせたりすることができます。

20.1 i アプリ-Flash 連携概要

i アプリは、`com.docomostar.ui.flashplayer.FlashPlayerPane` を通じて Flash プレイヤを呼び出し、指定した Flash コンテンツを i アプリの実行中に再生することができます。`FlashPlayerPane` クラスは Flash コンテンツを再生するためのフレーム (表示面) で、i アプリからは `Panel` や `Canvas` と同様に `Display.setCurrent()` メソッドを使用して画面に表示します。また、Flash コンテンツはユーザーからのキー入力を受け取ることができます。さらに、この際に、Flash コンテンツ中に含まれるアクションスクリプトと i アプリの記述によって相互にデータのやり取りを行うことが可能です。



i アプリ-Flash 連携機能は、`com.docomostar.ui.flashplayer` パッケージで提供されます。i アプリ-Flash 連携機能の API 構成を以下に示します。



各クラス、インタフェースの機能概要は以下の通りです。

com.docomostar.ui.flashplayer パッケージ

クラス、インタフェース	機能概要
FlashEventListener	プレイヤーから通知されるイベントのリスナを定義します。
HTTPRequest	プレイヤーから渡される HTTP 情報を定義します。
FlashErrorNotifiedEvent	プレイヤーのエラー通知イベントを定義します。
FlashEventObject	プレイヤーのイベントを表すクラスです。
FlashPlayerPane	Flash コンテンツを i アプリ上で再生するための表示面を定義します。 Flash コンテンツを再生するプレイヤーを i アプリ上で操作することが可能です。プレイヤーの再生領域は表示面の前面になります。
FlashStateChangedEvent	コンテンツ再生の可否が変化したことを表すイベントを定義します。すなわち、FlashPlayerPane のカレントから非カレントへの切り替わり、もしくはその逆をイベントとして通知します。
FscommandCalledEvent	プレイヤーの fscommand 実行イベントを定義します。
GetURLCalledEvent	プレイヤーの getURL 実行イベントを定義します。
LoadMethodCalledEvent	プレイヤーの load 系メソッド実行イベントを定義します。

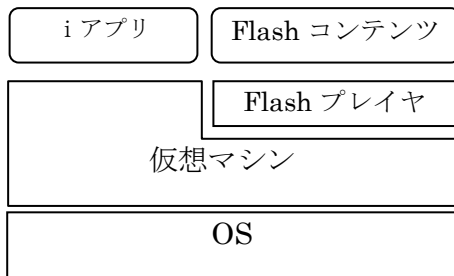
注意事項：

- i アプリから呼び出す Flash コンテンツ中のアクションスクリプトからは、直接インターネットと通信を行ったり、携帯電話のデバイス等を操作したりすることはできません。Flash コンテンツからそれらの

操作を行いたい場合は、i アプリにそれらを実行するコードを用意しておき、アクションスクリプトから i アプリのコードを呼び出すことにより操作することができます。詳細は、20.6 項を参照してください。

20.2 Flash プレイヤ

Flash コンテンツは **FlashPlayerPane** 上に配置されている Flash プレイヤ上で再生されます。このプレイヤはブラウザなどからの Flash 再生などで使用されるものと同じです。ただし、仮想マシンがプレイヤのホスト環境として動作します。



Flash プレイヤを実行する際はワークメモリが必要になります。i アプリ-Flash 連携機能では、このワークメモリはヒープから割り当てられます。アプリケーションプログラムは **FlashPlayerPane** オブジェクト生成時に Flash プレイヤの初期ワークメモリを指定することもでき、指定しない場合は 256K バイトが初期ワークメモリとして割り当てられます。また、Flash プレイヤ実行中にワークメモリが不足した場合にはワークメモリを自動的に拡張する機能を備えています。ただし、ヒープの容量制限を越えるワークメモリを割り当てて Flash プレイヤを実行することはできません。ワークメモリの確保が出来なかった場合は i アプリにイベントとして通知されます。

注意事項：

- i アプリから呼び出される Flash コンテンツは“LocalTrusted”で再生が行われます。
- Flash コンテンツのメタデータに記載されている背景色透過識別子の指定は無視されます。
- Flash コンテンツから、動画の再生や XMLSocket 通信など i モード向け Flash Lite でサポートされていない機能を利用することはできません。
- Flash コンテンツ（アクションスクリプト）のうち一部の機能は i アプリ側のアプリケーションロジックと連携させて処理を行うものがあります。それらについては 20.6 項を参照してください。

20.3 Flash コンテンツの再生

i アプリから Flash コンテンツを再生するには、`com.docomostar.ui.flashplayer.FlashPlayerPane` クラスを使用します。**FlashPlayerPane** クラス以外の方法で Flash コンテンツを再生することはできません。**FlashPlayerPane** のインスタンスは複数生成することも可能です。ただし、一度に再生できるのは、カレントフレームに指定されている **FlashPlayerPane** に設定された 1 つの Flash コンテンツのみです。

FlashPlayerPane オブジェクトに Flash コンテンツを設定するには、Flash コンテンツのバイナリデータ（byte 配列）を引数に指定して `set()` メソッドを呼び出します。また、**FlashPlayerPane** クラスでは Flash コンテンツの再生を制御するメソッドとして `play()`（開始）、`stop()`（停止）、`pause()`（一時停止）、`restart()`（再開）が用意されています。

なお、再生を停止した **FlashPlayerPane** オブジェクトに再度 **play()** メソッドで再生を開始させる場合には、同じコンテンツを繰り返し再生させる場合であっても、**set()** メソッドで再度 **Flash** コンテンツを設定しなおす必要があります。

Flash コンテンツの再生を行うコード例は以下のとおりです。

例: **Flash** プレイヤによる **SWF** コンテンツの再生

```
// ネットワークや ScratchPad、JAR リソース等から SWF 形式データを
```

```
// バイト列として取得しておく
```

```
byte[] swfData = ...
```

```
// FlashPlayerPane インスタンス(Frame のサブクラス)を生成。
```

```
FlashPlayerPane fpp = new FlashPlayerPane();
```

```
fpp.set(swfData);
```

```
Display.setCurrent(fpp); // FlashPlayerPane をカレントフレームにする。
```

```
fpp.play(); // stop(), pause(), restart()などにより再生をコントロール。
```

なお、**FlashPlayerPane** に **Flash** コンテンツを設定した時点では **Flash** コンテンツの内容はまだ画面には表示されません。**FlashPlayerPane** をカレントフレームに設定した上で **FlashPlayerPane.play()** を呼び出した時点で描画が開始されます。

Flash コンテンツは **i** アプリの描画領域の全体を使って再生されます (**DrawArea** キーによって描画領域が制限されていればその中で、全画面表示が指定されていれば全画面で再生されます)。つまりブラウザでの **Flash** 再生で言うインタラクティブ再生のみがサポートされます。

注意事項：

- インライン再生を行うことはできません。また、再生指示はカレントフレームに設定された **FlashPlayerPane** オブジェクトに対してのみ行うことができます。
- **Flash** コンテンツ再生中に別のコンテンツを設定した場合、再生操作 **play()** を実行しないかぎり設定したコンテンツが自動再生されることはありません。
- **Flash** コンテンツの再生を停止した場合、停止時のフレームが表示されたままになっています。
- 再生中に、**i** アプリがサスペンドしたりダイアログが表示されたりした場合、再生は一時停止されます。レジュームしたりダイアログが非表示となっても再生は自動的に再開されません。また、再生中または一時停止中に他のフレームが **Display.setCurrent()** メソッドでカレントフレームに設定された場合、再生は停止します。なお、このようにして停止または一時停止している **FlashPlayerPane** オブジェクトが複数存在している場合、それらを再度カレントフレームに設定しても **Flash** プレイヤの描画内容が復元できない場合があります（その場合、描画内容は **FlashPlayerPane** オブジェクトの背景色で塗りつぶされます）。
- **Flash** コンテンツを **Flash** プレイヤに設定する際に、背景色を指定することができますが、背景色を指定しなかった場合には、**Flash** コンテンツのステージ色が背景色として使用されます。**Flash** コンテンツのステージ色が指定されていない場合には、**Flash** プレイヤのデフォルトの背景色（メーカー依存）が使用されます。
- **i** アプリは、**FlashPlayerPane** オブジェクトの **setQuality()** メソッドに **QUALITY** 定数を指定することで、プレイヤのレンダリング品質を指定することができます。現在のレンダリング品質は、**getQuality()** メソッドで調べることができます。

- i アプリは **FlashPlayerPane** オブジェクトを通じて、コンテンツのフレーム情報（総フレーム数やフレーム間隔、現在表示しているフレーム番号など）を取得することができます。また、ループ再生などの制御を行うこともできます。

20.4 Flash コンテンツでのユーザー操作

FlashPlayerPane がカレントフレームになっているときは、ファンクションキーのキーイベントは i アプリ側で受け取ります。つまり、**FlashPlayerPane** に対するキーイベントが発生します。ファンクションキーのイベントを取得するには、**Panel** と同様に、**SoftKeyListener** オブジェクトを使用します。ファンクション以外のキーについてのキーイベント（0-9、*、#、上下左右、選択（決定）、およびオプションキーとしてクリアキーがサポートされます）は **Flash** プレイヤが受け取ります。**Flash** コンテンツ中にテキストフィールドがあるとき、テキストフィールドを操作すると、IME が起動し文字の入力が可能となります。

注意事項：

- ポインティングデバイスやタッチパネルなど、キー以外の操作デバイスを備える機種において、それらのデバイスを利用した操作がサポートされるかどうかは機種によって異なります。
- **Flash** コンテンツのテキストフィールドは、アクションスクリプトの **TextField.maxChars** プロパティを使用して、入力可能な最大文字数を設定することができますが、i アプリ上で **Flash** コンテンツを動作させる場合には、**maxChars** プロパティの設定値より長いテキストを初期値として設定することはできません。制限長を超えた初期値を設定した場合、テキストフィールドの操作を行うことができなくなるため注意してください。

20.5 Flash コンテンツからの携帯電話リソースの利用

Flash コンテンツから携帯電話のバイブレータを動作させたり、サウンドを再生したりすることができます。

バイブレータの制御は、i アプリからのバイブレータ制御指示と **Flash** コンテンツからのバイブレータ制御指示は後から行われた指示が優先されます。また、i アプリは **PhoneSystem.getAttribute()** メソッドを使用して現在バイブレータが動作しているかどうかを取得することができますが、その際は **Flash** 側が実行したバイブレータ動作についても区別せず、動作状態が取得できます。

Flash コンテンツからのサウンド再生は、**AudioPresenter** におけるポート指定がなく最も高い優先順位（**AudioPresenter.MAX_PRIORITY**）が指定されたプレゼンタと同等のものとして扱われます。**Flash** コンテンツからのサウンド再生と i アプリからのサウンド再生を同時に行った場合にどのプレゼンタが有効なものとして再生されるかは「i アプリコンテンツ開発ガイド 詳細編」4.5.3 項を参照してください。

なお、i アプリ-Flash 連携機能によって実行されている **Flash** コンテンツでは、アクションスクリプトの **fscommand2** 関数で規定されている携帯電話情報のうち以下がサポートされます。

- 時刻
- 電波状態（最大値、現在値）
- 電源状態（電池の最大値、現在値、充電状況）
- 端末種別・機種情報

上記の機能のうち、電波状態または電源状態を利用する **Flash** コンテンツを i アプリ-Flash 連携機能によって実行する i アプリは、ADF の **GetSysInfo** キーに値 **"yes"** を設定する必要があります。

注意事項：

- Flash コンテンツがパイプレータを動作させているときやサウンドを再生させているときに Flash コンテンツの再生を停止もしくは一時停止させた場合、あるいは i アプリがサスペンド状態になった場合、パイプレーション動作やサウンド再生も停止します。なお、Flash コンテンツの再生再開時、もしくはレジューム時にパイプレータ動作やサウンド再生は自動的に再開されませんので必要に応じて明示的に再開処理をする必要があります。
- Flash コンテンツで再生可能なサウンドは、SMF 形式ファイルと、MFi 形式ファイルのみです。

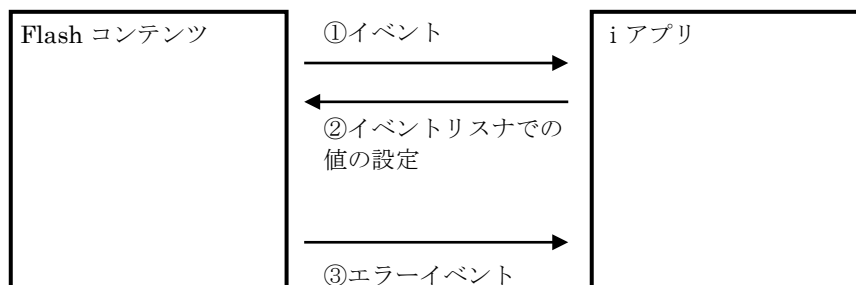
20.6 i アプリとのイベントおよびデータのやり取り

Flash コンテンツ中のアクションスクリプトから特定の操作を実行することで、i アプリに対してイベントを通知することができます（下図の①）。このようなイベントを発生させる特定の操作とは以下の関数を呼び出した場合、あるいはクラスを使用した場合です。

- `getURL`
- `fscommand`
- `loadMovie`、`loadMovieNum`、`loadVariables`、`loadClip`、`LoadVars`、`sendAndLoad` などロード系関数

アクションスクリプトでこれら呼び出したり使用した場合、あらかじめ設定しておいた i アプリ側のイベントリスナが呼び出されます。イベントハンドラの引数には、これらの種別および呼び出し時に与えられた引数が渡されます。

Flash コンテンツと i アプリの実行中のやり取りの図を以下に示します。



Flash プレイヤからのイベントを受け取るためには、アプリケーションプログラムは `FlashPlayerPane` オブジェクトの `addFlashEventListener()` メソッドを呼び出してリスナ（`FlashEventListener` インタフェースを実装したオブジェクト）を登録します。イベントが発生すると、リスナオブジェクトの `notificationReceived()` メソッドが呼び出されます。

`notificationReceived()` メソッドは、引数としてイベントオブジェクトを受け取ります。イベントの種別毎にイベントクラス（`FlashEventObject` クラスのサブクラス）が用意されており、現在のプロファイルでは以下のものが定義されています。

- `FlashErrorNotifiedEvent` : Flash プレイヤでエラーが発生したことを通知するイベント
- `FlashStateChangedEvent` : Flash プレイヤの再生可否状態が変化したことを通知するイベント
- `FscommandCalledEvent` : `fscommand` 関数が呼び出されたことを通知するイベント
- `GetURLCalledEvent` : `getURL` 関数が呼び出されたことを通知するイベント
- `LoadMethodCalledEvent` : `load` 系関数が呼び出されたことを通知するイベント

イベントの種類によっては、i アプリ側でそのイベントに対して値を設定することができます。具体的には、`com.docomostar.ui.flashplayer.LoadMethodCalledEvent` クラスの `sendData(byte[] data)` メソッドを呼び出すことで、Flash コンテンツ側に i アプリ側からバイト配列データを受け渡すことができます（上図の②）。

何らかのエラー（例えばメモリ不足などのエラー）が発生した場合、Flash プレイヤは Flash コンテンツの再生を停止した上で、`FlashErrorNotifiedEvent` イベントとして i アプリ側に通知します（上図の③）。

注意事項：

- ②でのデータの送信は、イベント発生時にのみ実行できます。つまり、i アプリにとって、Flash コンテンツの実行と非同期に任意のタイミングで Flash コンテンツにデータを渡すことはできません。
- リスナのイベントハンドラは直列化されて呼び出されます。すなわち、イベントハンドラからリターンする前に次のイベントに対応するイベントハンドラが呼び出されることはありません。
- ユーザーインタラクションがないタイミングにおいても、再生中の Flash コンテンツは i アプリに対してイベント発行を行うことができます。
- リスナのイベントハンドラ実行中に Flash プレイヤが停止した場合、イベントハンドラの処理は実行されますが完了通知はプレイヤ側には渡りません。
- `getURL` などのアクションスクリプト関数は、本来はブラウザから呼び出された Flash コンテンツにおいて、特定の意図（例えば `getURL` ではサーバーからドキュメントを取得したり、サーバーのアプリケーションを起動させる、など）を持って呼び出すものですが、i アプリとの連携時には、これらの操作の呼び出しに対して i アプリがどのような動作をするかは、i アプリ次第です。i アプリ側の実装によっては、アクションスクリプトの本来の機能とは異なる意味を持つ処理をそのアクションスクリプト関数に割り当てることがもできます。

Flash コンテンツからのイベント処理を行うコード例は以下の通りです。

例: Flash プレイヤーからのイベントの取得とデータの送信

```
public class SampleListener implements FlashEventListener {
    // Flash プレイヤーからのイベントを受け取るリスナメソッド
    public void notificationReceived(FlashEventObject event) {
        // イベントタイプを判別
        if (event.getType() == FlashEventObject.FLASH_LOAD_METHOD_CALLED) {
            LoadMethodCalledEvent lmcEvent = (LoadMethodCalledEvent) event;
            // Flash 側から送られてきた情報をイベントオブジェクトの getURL() メソッドで取得。
            if ((lmcEvent.getURL()).equals("Some Command String")) {
                // Flash プレイヤーに情報を送信。
                lmcEvent.sendData("Flash プレイヤーに送信するデータ".getBytes());
            }
        }
    }
}

// FlashPlayerPane を生成する。
FlashPlayerPane fpp = new FlashPlayerPane();
// イベントリスナを登録する
fpp.addFlashEventListener(FLASH_LOAD_METHOD_CALLED, new SampleListener());
// Flash プレイヤーに SWF データを設定し再生を行う。
```

```
byte[] swfData = ...
fpp.set(swfData);
fpp.play();
```

20.7 Flash Lite 共有オブジェクトのアクセス

i アプリから呼び出した Flash コンテンツ中のアクションスクリプトにおいて、Flash Lite 共有オブジェクト(SharedObject)を使用すると、携帯電話にデータを保存することができます。Flash Lite 共有オブジェクトの保存には、ScratchPad が使用されます。具体的には、ADF の UseMSO キーで ScratchPad の番号を指定することで、指定された番号の ScratchPad 領域が Flash Lite 共有オブジェクトの保存に使用されるようになります。Flash Lite 共有オブジェクト領域として指定された ScratchPad は、i アプリから ScratchPad としてアクセスすることはできません（アクセスしようとする場合例外が発生します）。

Flash Lite 共有オブジェクトのデータストレージ領域（以下、MSO）として使用する ScratchPad の番号は、ADF の UseMSO キーに整数値（0 から始まる、ScratchPad の URL に指定する番号）で指定します。以下のように指定した場合には、データストレージ領域（MSO）として 2048 バイトが使用可能です。

```
指定例： SPSize = 1024,2048
         UseMSO = 1
```

注意事項：

- Flash Lite 共有オブジェクト領域に保存できるデータの個数はメーカーに依存しますが、最低 20 個のデータが保存できることが保証されています。保存できる個数を超えた場合あるいは容量がオーバーした場合には、データの書き込みはなされません。
- フル+ミニアプリでは、先頭（0 番目）の ScratchPad がフルアプリとミニアプリからの共有 ScratchPad となりますので、先頭の ScratchPad を UseMSO で指定すると ScratchPad の共有ができなくなります。
- ミニアプリでは i アプリ-Flash 連携機能を使用できないため、ミニのみアプリにおいて UseMSO キーで Flash Lite 共有オブジェクト領域が指定されていた場合、その指定は無視されます。ミニのみアプリで UseMSO キーで指定された領域は、Flash Lite 共有オブジェクト領域ではなく、ScratchPad として使用されます。
- i アプリの削除時にそれに結びついた ScratchPad 上の Flash Lite 共有領域は削除されます。
- Flash Lite 共有オブジェクトは、i アプリから起動された場合と、その他の方法で起動された場合とで厳密に区別されます。つまり、データ BOX に保存されている Flash コンテンツを i アプリから呼び出して再生する場合、データ BOX 中に保存されていた Flash Lite 共有オブジェクトにアクセスすることはできません。i アプリから起動された Flash コンテンツからは必ず ScratchPad から割り当てられた領域が使用されます。また、i アプリからデータ BOX に Flash コンテンツを保存する場合、i アプリと対応付けられている Flash Lite 共有オブジェクトの内容がデータ BOX にコピーされることはありません。
- i アプリのバージョンアップ時に、ADF の UseMSO キーの値が変更されている場合には、変更前の Flash Lite 共有オブジェクト領域の内容および変更後の Flash Lite 共有オブジェクト領域の内容は初期化されます。Flash Lite 共有オブジェクト領域として割り当てられていた ScratchPad 領域をバージョンアップによって通常の ScratchPad 領域に変更したとしても、バージョンアップ前に保存されていたデータを読み出すことはできません。
- i アプリ-Flash 連携機能によって動作している Flash コンテンツが Flash Lite 共有オブジェクトへの書き込みを行う場合、実際の物理ストレージへのデータの反映は SharedObject.flush()、SharedObject.clear()、SharedObject.close() が明示的に呼び出された際に行われます。これらを使用しない場合にはデータは保存されないため注意が必要です。
- アクションスクリプトの記述において、SharedObject の addListener() メソッドで登録された関数定義が外部から参照を保持されていない場合、その関数定義が Flash プレイヤ側のガベージコレクション機構によって意図していないタイミングで回収されてしまう場合があります。この問題が発生すると、Flash Lite 共有オブジェクトが利用可能となっても addListener() メソッドで登録された関数は実行さ

れません。この問題を避けるには、**SharedObject** の **addListener()** メソッドで登録する関数について、必要に応じて外部から参照を保持するようにしてください。

例：

```
var handler;
function process(sharedObjectName) {
    function loadCompleteHandler(mySO:SharedObject) {
        ...
    }
    handler = loadCompleteHandler;
    ...

    var soDummy:SharedObject = SharedObject.getLocal(sharedObjectName);
    SharedObject.addListener( sharedObjectName, handler );
}
```

第 21 章

ユーティリティ API

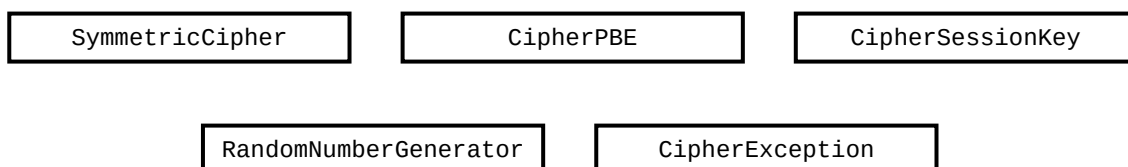
この章では、これまでに解説した機能カテゴリに含まれない、オプションのユーティリティ機能について解説します。

21.1 暗号化機能

暗号化機能では、i アプリが取り扱う任意のバイト列データを暗号化・復号するための機能を提供します。この機能を利用して、i アプリと外部（サーバーや外部機器など）のデータ交換を、より安全に行うことが可能となります。

i アプリがサポートする暗号化機能は、PKCS#5 (RFC2898) で規定されているパスワードベースの暗号化機能（Password Based Encryption : PBE）に基づいています。これは、共通鍵暗号方式に属する暗号化機能です。処理対象データは乱数から生成された鍵（セッション鍵）を用いて暗号化されます。データを送信する際には、セッション鍵で暗号化されたデータとともに、パスワードベースで暗号化を施したセッション鍵を相手に送ります。データを解読する際には、パスワードによってセッション鍵を復号し、そのセッション鍵を用いて暗号化データを復号します。

暗号化機能は、`com.docomostar.security` パッケージで提供されます。暗号化機能の API 構成を以下に示します。



以下に、各クラスの機能を解説します。

クラス	概 要
SymmetricCipher	共通鍵暗号による暗号化・復号機能を提供します。鍵（セッション鍵）は、 CipherSessionKey クラスで表現されます。処理対象のデータは、本クラスの機能を使用して暗号化または復号を行います。
CipherPBE	パスワードベースの暗号化でセッション鍵を取り扱う機能を提供します。この機能を使用して、暗号化を行うための新たなセッション鍵を生成したり、暗号化されたセッション鍵（バイト列）からデータ復号用のセッション鍵を取得することができます。
CipherSessionKey	セッション鍵を表すクラスです。このクラスのオブジェクトは、 CipherPBE クラスを使用して取得します。またこのクラスのオブジェクトからは、オブジェクトのセッション鍵を使用して暗号化・復号を行うための SymmetricCipher オブジェクトや、暗号化されたセッション鍵のバイト列データを取り出すことができます。

RandomNumberGenerator	乱数ジェネレータです。指定された長さの乱数を取得する機能を提供します。
CipherException	暗号化機能に関する例外が発生した際にスローされる例外クラスです。この例外クラスは getStatus() メソッドを備えており、例外が発生した大まかな理由を調べることができます。

第 22 章

ジャイロセンサー

ジャイロセンサー機能は、携帯電話の機体の回転運動を角速度（時間あたりの角度の変化量）の数値として取得する機能です。角速度の大きさや変化を調べることで機体の動き、つまりユーザーがどのように機体を動かしているかを知ることができ、ユーザーの動きに連動したアクションを i アプリに与えることができるようになります。角速度の単位は、deg/s（角度毎秒）です。

ジャイロセンサーAPI では、角速度を x、y、z の 3 方向の回転軸に対して取り扱えるよう API を定義しています。これは 3 軸のジャイロセンサーデバイスを前提とした仕様ですが、実際に携帯電話に搭載されるジャイロセンサーデバイスは 2 軸以下のものであることもあり、その場合は角速度が取得できる回転軸方向に制限が生じる可能性があります。

ジャイロセンサー機能は、`com.docomostar.opt.device` パッケージで提供されます。ジャイロセンサー機能のクラス構成は以下の通りです。

`com.docomostar.opt.device` パッケージ

GyroSensor

GyroData

クラス	概 要
GyroSensor	ジャイロセンサー機能を提供するクラスです。アプリケーションプログラムは、このクラスを使用して角速度の計測を行います。角速度の計測には、アプリケーションプログラムが指示する間隔で繰り返し計測を行う周期計測があります。（加速度センサー機能にあるワンショット計測はサポートされません） 計測結果は、次項に記載する GyroData クラスのオブジェクトとして表現されます。
GyroData	ジャイロセンサーの計測結果を表すクラスです。このオブジェクトから、回転軸毎の角速度値を取り出すことができます。 周期計測をサポートするため、1 つの GyroData オブジェクトには複数件の計測結果が格納される場合があります。

ジャイロセンサー機能を利用するには、まず **GyroSensor** クラスの static メソッド `getGyroSensor()` を呼び出して **GyroSensor** オブジェクトを取得します。以下に、ジャイロセンサーAPI の利用方法について解説します。

(1) ジャイロセンサー機能の能力サポート範囲

GyroSensor クラスには、ジャイロセンサー機能がサポートしている能力を知るための以下のようなメソッドが用意されています。

メソッド	概 要
<code>getAvailableData()</code>	ジャイロセンサー機能で、どのデータを計測できるかを調べるメソッドです。 <code>x,y,z</code> 各回転軸のデータ項目のうち、どの項目をサポートしているかを調べることができます。
<code>getIntervalResolution()</code>	周期計測における、実行可能な最短の周期間隔を調べるメソッドです。周期計測ではアプリケーションプログラムが周期間隔を指定しますが、指定された値はシステム内部でこのメソッドの返す値の倍数に丸められます。
<code>getMaxDataSize()</code>	周期計測において、システム内部に蓄積可能な計測データの最大件数を調べるメソッドです。周期計測ではアプリケーションプログラムとは非同期に、定期的に角速度の計測が行われ、その結果がシステム内部のバッファに蓄積されていきます。このメソッドは、システム内部に何件分の計測データを保持できるかを返します。 なお、このバッファに保持可能な最大件数を超えて周期計測が続けられた場合は、バッファ内の古いデータから順次上書きされていきます。
<code>getMaxDataValue()</code>	各計測データ項目について、取り得る値の最大値を調べるメソッドです。
<code>getMinDataValue()</code>	各計測データ項目について、取り得る値の最小値を調べるメソッドです。

`getAvailableData()`メソッドで返されるデータ項目種別は、GyroSensor クラスの定数として定義されています。

(2) 周期計測の開始と終了、測定結果の取得

ジャイロセンサー機能では、周期計測による角速度の計測をサポートしています。周期計測とは、アプリケーションプログラムが指示する間隔でシステムが自動的に繰り返し計測を行ってその結果を蓄積している、アプリケーションプログラムが必要とするタイミングでその結果を取り出す方法です。

周期計測を行うには、GyroSensor.start()メソッドにより周期計測を開始した上で、任意のタイミングでGyroSensor.getData()メソッドを呼び出します。start()メソッドには、第1引数として周期計測の間隔をミリ秒単位で指定します。第2引数には、サスペンド中に周期計測を継続するかどうかを指定します。第2引数にtrueを設定しておくことで、サスペンド中でも周期計測を継続できます。ただしこの機能はオプションであり、サスペンド時の周期計測の自動継続をサポートしない機種もあります。開始された周期計測は、GyroSensor.stop()メソッドを呼び出すと停止します。

getData()メソッドは、GyroData オブジェクトの形で測定結果を返します。このメソッドを呼び出すと、システムはシステム内部に蓄積された計測データをGyroData オブジェクトにコピーしてアプリケーションプログラムに返します。GyroData オブジェクトはi アプリ実行環境内に1つだけ存在し、繰り返し再利用されます。従って、getData()メソッドは何度呼び出しても同じオブジェクトを返します。GyroData オブジェクトの利用方法については、次項を参照してください。

周期計測の計測結果が何件蓄積されているかは、GyroSensor.getDataSize()メソッドを使用して調べることができます。

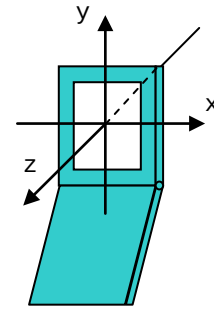
(3) 測定結果の利用 (GyroData クラス)

ジャイロセンサー機能の計測結果は、GyroData クラスで表現されます。このクラスには、ジャイロセンサー機能で測定した各回転軸毎の角速度の値を取り出すためにgetAngularVelocityX()、getAngularVelocityY()、getAngularVelocityZ()の各メソッドが用意されています。このメソッドで取得される値の単位は、deg/s(角度毎秒)です。

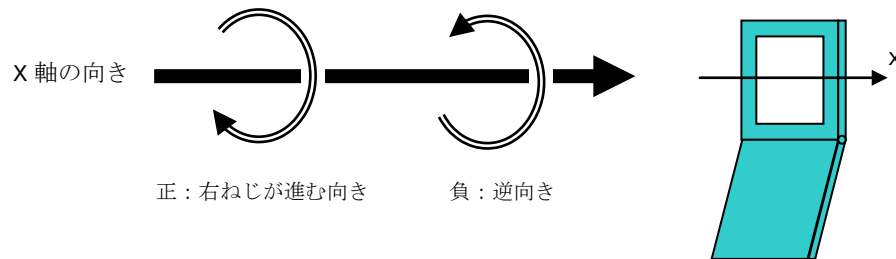
角速度の x, y, z の各回転軸の方向はメインスクリーンを基準に定められています。 x 方向はメインスクリーン上を左から右に向かう方向、 y 方向はメインスクリーン上を下から上に向かう方向、 z 方向はメインスクリーンの奥（背面側）から手前に向かう方向になります。

x, y, z の各方向：

- x ： メインスクリーン上を左から右に向かう方向
- y ： メインスクリーン上を下から上に向かう方向
- z ： メインスクリーンの奥から手前に向かう方向



また角速度は各回転軸の正方向に対して右ねじが進む向きに回転した場合を正とします。



周期計測の結果を `GyroSensor.getData()` メソッドで取得する場合には、蓄積された件数分の計測結果が1つの `GyroData` オブジェクトの中に格納されます。このため、各回転軸のデータ項目値を取得する各メソッドは、結果を配列の形で返すようになっています。`getData()` メソッド用に割り当てられた `GyroData` オブジェクトは、周期計測でシステムが蓄積可能な件数を格納するのに十分な長さの配列をあらかじめ保持しています。実際に蓄積されている件数が少ない状況では配列の先頭要素から順に有効なデータが設定されますが、その有効な要素数は `size()` メソッドによって調べることができます。

注意事項：

- サスペンド中に周期計測を継続する機能に対応していない端末では、`GyroSensor.start()` メソッドの第2引数に `true` を指定すると `UnsupportedOperationException` の例外が発生します。
 - 周期計測によってシステム内部に蓄積された計測結果は、以下の契機でクリアされます。
 - `GyroSensor.getData()` メソッドによって、周期計測の結果が取り出されたとき
 - 周期計測を停止した後、周期計測を開始したとき
 - `GyroSensor.disposeData()` メソッドによって明示的に計測結果のクリアが指示されるか、もしくは計測結果を蓄積したままアプリを終了したとき
- 周期計測を停止した後も、これらの契機までは計測結果はシステム内部に蓄積されたままとなっており、`GyroSensor.getData()` メソッドで結果を取り出すことができます。
- 周期計測を行なっている状況では、周期計測を新たに開始することはできません。
 - 回転軸の方向は、機体に対してディスプレイ部を普通に開状態とした状態でのメインスクリーンの向きを基準とします。ディスプレイ部を左右に折り曲げられるようなタイプの機種や特殊な形状に開くことのできる機種では、ディスプレイ部を機体と異なる向きに折り曲げたり特殊な形状に開いた状態ではディスプレイ部の向きと回転軸の向きとが合致するとは限りません。

【Star-1.2】

本機能は Star-1.2 プロファイルで新設されました。

【Star-1.5】

Star-1.5 プロファイルにて、ジャイロセンサー機能を提供する **GyroSensor** クラス、**GyroData** クラスは **com.docomostar.device.location** パッケージにも配置されました。これに伴い、Star-1.5 プロファイル以降では、**com.docomostar.opt.device** パッケージ配下のジャイロセンサー機能は非推奨になりました。

com.docomostar.device.location パッケージに新設されたジャイロセンサー機能では、ジャイロセンサー機能の有効化・無効化状態を切り替える **setEnabled()** メソッドが **GyroSensor** クラスに追加されました。また、周期計測に加え、ワンショット計測をサポートしています。

com.docomostar.device.location パッケージに新設されたジャイロセンサー機能を利用するには、まず **GyroSensor** クラスの static メソッド **getGyroSensor()** メソッドを呼び出して **GyroSensor** オブジェクトを取得します。その後、周期計測、ワンショット計測を開始する前に、**setEnabled()** メソッドを使用して、ジャイロセンサー機能を有効化する必要があります（従来の **GyroSensor** クラスと使用方法が異なるため注意してください）。

ワンショット計測は、アプリケーションプログラムが任意のタイミングで **GyroSensor.getCurrentData()** メソッドを呼び出すことにより行われます。また、**GyroSensor.start()** メソッドを呼び出して周期計測を開始した上で、任意のタイミングで **peekLatestData()** メソッドを呼び出すと、蓄積された計測結果のうち、最新の 1 件のみを取得することができます。

getCurrentData()、**peekLatestData()** の各メソッドは、**GyroData** オブジェクトの形で計測結果を返します。システムは、**getCurrentData()**、**peekLatestData()**、**getData()** の各メソッドごとに専用の **GyroData** オブジェクトを割り当てて保持しており、個々のメソッドはそれぞれ常に同じオブジェクトを返すようになっています。

GyroData オブジェクトの利用方法については、**com.docomostar.opt.device** パッケージに配置されている **GyroData** オブジェクトの利用方法を参照してください。

なお、**com.docomostar.opt.device** パッケージに配置されている従来の **GyroSensor** クラスには、Star-1.5 プロファイル以降もこれらのメソッドは追加されません。

【Star-1.5】（2011 年春以降発売モデル）

Star-1.5 プロファイルに対応する携帯電話の中でも 2011 年春以降に発売する携帯電話には、ジャイロセンサー機能の中でもオプションな以下の機能を搭載したものがあります。

- ジャイロセンサーからイベントを検知し、そのイベントをアプリケーションプログラムに通知する機能
- ジャイロセンサーのキャリブレーションを行う機能

このため、**GyroSensor2** クラスと **GyroSensor2EventListener** インタフェースが i アプリオプション API として **com.docomostar.opt.device.location** パッケージに定義されました。

ジャイロセンサーからのイベント検知機能を利用するには、アプリケーションプログラムで以下のような手順を踏みます。

1. **GyroSensor2EventListener** インタフェースを実装したリスナーオブジェクトを用意し、**GyroSensor2.setEventListener()** メソッドでシステムに登録します。
2. 検知するイベントの種別を引数に指定して **GyroSensor2.startEventDetection()** メソッドを呼び出し、ジャイロセンサーからのイベント検知を開始します。複数の種別のイベントを検知したい場合は、**startEventDetection()** メソッドの引数にそれらイベント種別の論理和を指定します。指定したイベントが発生すると、リスナーオブジェクトの **actionPerformed()** メソッドが呼び出されることでアプリケーションプログラムにイベントが通知されます。**actionPerformed()** メソッドは、イベント発生源となった **GyroSensor2** オブジェクト、発生したイベントの種別、および各イベントに固有のイベント値を受け取ります。
3. イベント検知を停止し機能の利用を終了するには、**GyroSensor2.stopEventDetection()** メソッドを呼び出します。このメソッドを呼び出すと、開始したイベント検知は全て停止します。

Star-1.5 プロファイルでは、キャリブレーションに関する以下のイベントが定義されています。

- キャリブレーションが必要になった (`GyroSensor2.EVENT_CALIBRATION_REQUIRED`)
- キャリブレーションが完了した (`GyroSensor2.EVENT_CALIBRATION_COMPLETED`)
- キャリブレーションが中断した (`GyroSensor2.EVENT_CALIBRATION_INTERRUPTED`)

キャリブレーションとは、ジャイロセンサーが角速度を正しく計測できない状態になった場合に再度正しく計測できる状態に戻すことを言います。ジャイロセンサーは周囲の温度の影響などにより計測精度が悪化し、正しく角速度を計測できなくなることがあります。そのような場合にキャリブレーションを行うことで、正しく角速度を計測できるようになります。

キャリブレーションイベントをサポートする機種では、キャリブレーションを実行すべき状況になると、i アプリ実行環境は `EVENT_CALIBRATION_REQUIRED` イベントを通知します。このイベントの通知を受けたら、ユーザーにキャリブレーションに必要な動作（携帯電話を水平にした状態で静止させる、など）を取るよう促し、`GyroSensor2.calibrate()` メソッドを呼び出してキャリブレーションを実行してください。キャリブレーションが完了すると、i アプリ実行環境は `EVENT_CALIBRATION_COMPLETED` イベントを通知します。何らかの理由でキャリブレーションが中断した場合には `EVENT_CALIBRATION_INTERRUPTED` イベントが通知されます。

なお、どのイベントを検知可能かは機種により異なる場合があります、`GyroSensor2.getAvailableEvent()` メソッドにより調べることができます。

注意事項：

- 周期計測を開始している状態で `setEnabled()` メソッドによりジャイロセンサーを無効にした場合は、周期計測は自動的に停止します。自動的に再開は行われなため、必要に応じてアプリケーションプログラム側で再開処理を行うようにしてください。その際、周期計測が停止する前に蓄積された計測結果は、周期計測を再開する前に取り出す必要があることに注意してください。
- `setEnabled()` メソッドを呼び出すことで `com.docomostar.device.location.GyroSensor` オブジェクトを保持したままジャイロセンサの有効、無効を切り替えることができます。一般にジャイロセンサーが有効な状態では消費電力が大きくなるため、機能を使用しない間はジャイロセンサー機能は無効にしておくようにすることを推奨します。
なお、`com.docomostar.opt.device.GyroSensor` クラスでは `com.docomostar.opt.device.GyroSensor.getGyroSensor()` メソッドを呼び出すことにより有効化され、i アプリ終了時に無効化されます。
- 周期計測を行っている状態で、`com.docomostar.opt.ui.Canvas2` クラス（第5章参照）を利用して座標系を変更した場合、蓄積された計測結果の一般性が保証されなくなり、期待する計測結果が得られないことがあります。期待する計測結果を得るためには、一旦周期計測を停止させ蓄積された計測結果を取得してから、座標系を変更し、周期計測を開始してください。
- `com.docomostar.device.location.GyroSensor` クラスと `com.docomostar.opt.device.GyroSensor` クラスのオブジェクトを同時に生成することはできません。一方のクラスのオブジェクトを生成するメソッド（`getGyroSensor()`）を呼び出した後、他方のクラスでこのメソッドを呼び出すと例外が発生します。
- `GyroSensor2` クラスをサポートしている場合、`com.docomostar.device.location.GyroSensor.getGyroSensor()` メソッドは常に `GyroSensor2` オブジェクトを返します。ただし、`com.docomostar.opt.device.GyroSensor.getGyroSensor()` メソッドが `GyroSensor2` オブジェクトを返すことはありません。
- ジャイロセンサー機能の周期計測あるいはワンショット計測とキャリブレーションを同時に実行することはできません。

第 23 章 フルキーボード

携帯電話の機種によっては入力デバイスとして、QWERTY 配列キーに代表される、文字（英字）キーを配した操作デバイス（フルキーボード）を搭載しているものがあります。このような機種では、フルキーボードの操作を i アプリへ伝える機能をサポートしたものもあります。フルキーボード機能は `com.docomostar.opt.ui.FullKeyboard` クラスで提供されます。なお、本項で解説するフルキーボード機能は低レベル UI（Canvas）利用時のみ使用することができます。

フルキーボードに対して、通常の携帯電話で使用される 0～9 の数字と*、#の計 12 個のキーを配した操作デバイスを 12 キーと呼びます。実際にはこれら 12 個のキー以外にも決定キーや上下左右キーなども操作可能なキーに含まれますが、ここでは簡易的に 12 キーとします。

フルキーボード機能に対応した携帯電話では、フルキーボードが有効な状態（フルキーモード）、12 キーが有効な状態（12 キーモード）のいずれかの状態となっています。通常はフルキーボードが無効化されており、12 キーモードの状態となっています。これらのモードを切り替えるには、`FullKeyboard.setEnabled()` メソッドを使用します。このメソッドの引数に `true` を指定した場合はフルキーモードとなり、`false` を指定した場合は 12 キーモードとなります。

`FullKeyboard.isEnabled()` メソッドにより、現在フルキーボードが有効であるか否かを確認することもできます。

フルキーモードでは、フルキーボードの操作により `Display.KEY_PRESSED_EVENT`、`Display.KEY_RELEASED_EVENT` に加え、キー操作により文字を検出したことを表す `Display2.KEY_TYPED_EVENT` のイベントが、`Canvas.processEvent()` に通知されます。`Display.KEY_PRESSED_EVENT`、`Display.KEY_RELEASED_EVENT` のイベントでは、`FullKeyboard` クラスで定義されているキーコードがパラメータとして通知されます。また `Display2.KEY_TYPED_EVENT` のイベントでは、検出された文字（Unicode 文字）がパラメータとして通知されます。

また、`Canvas.getKeypadState()` メソッドによりフルキーボードのキー状態を取得することができます。キー状態は、フルキーモードか 12 キーモードかによらず取得できます。つまり、フルキーモードであっても 12 キーのキー状態を取得することができ、12 キーモードであってもフルキーボードのキー状態を取得することができます。

フルキーボードのキーは、キーグループ 8～15（キー値 0x0100～0x01ff）として規定されています。フルキーボードに対応した携帯電話では、このキーグループは初期状態で有効となっています。

【Star-1.3】

Star-1.2 プロファイルでは、**FullKeyboard** クラスに 0x0100～0x014d の値のキーコードが定数として用意されています。

Star-1.3 プロファイルではこれらに加えて、0x014e～0x015a の値のキーコードが定数として追加されました。それぞれの値がどのキーを示すかは、『i アプリコンテンツ開発ガイド オプション API リファレンス編』を参照してください。

注意事項：

- フルキーボードを搭載していても携帯電話の機種によっては **FullKeyboard** クラスで定義されているキーを全て備えているとは限りません。
- 機種によっては、1つの入力デバイスでフルキーボードと 12 キーを兼用している場合があります。兼用しているキーが操作された場合、どちらのモードであるかによって、**Display.KEY_PRESSED_EVENT**、**Display.KEY_RELEASED_EVENT** のイベントとして **Canvas.processEvent()** に通知される際のキーの値は異なります。
ただしキー状態を取得する際には、どちらのモードであっても、それぞれのキーが押された状態であるとみなされ、それぞれのキーの値を取得できます。
- フルキーボードのキーには、2つの文字が割り当てられている場合があります。通常、ユーザーは **Fn** キーなどと呼ばれる特殊なキーと組み合わせることで、どちらの文字を入力するかを選択します。フルキーボードの実装には、**Fn** キーに独自のキーコードを割り当てるものと、**Fn** キー自体にはキーコードを割り当てないものの 2 パターンがあります。**Fn** キーにキーコードが割り当てられていない機種では、**Fn** キー単体の操作ではイベントやキー状態の通知は行われません。**Fn** キーと他のキーとを組み合わせる操作した際に決定された値が、i アプリに通知されます。
一方、**Fn** キーにキーコードが割り当てられている機種では、**Fn** キー単体の操作によりイベントやキー状態の通知が行われます。**Fn** キーと他のキーとを組み合わせる操作しても、それにより決定された値は i アプリに通知されません。組み合わせとして操作されたキーは、それが単体で操作された際のイベントやキー状態が i アプリに通知されます。

【Star-1.2】

本機能は Star-1.2 プロファイルで新設されました。

第 24 章

赤外線リモコン受信機能

i アプリ基本 API では、`com.docomostar.device` パッケージの `IrRemoteControl` クラス、および `IrRemoteControlFrame` クラスで赤外線リモコン機能を提供しています。ただし、これらの API はリモコン信号を送信する機能のみをサポートしており、外部から送信されたリモコン信号を受信する機能は持っていません。

本項で解説する赤外線リモコン受信 API では、外部から送信されたリモコン信号を携帯電話側で受信する機能を提供します。

なお、本項で解説する赤外線リモコン受信 API は、API の簡素化のため、リモコン信号パラメータの設定方法や設定内容は必ずしも `IrRemoteControl` クラスの概念と対称形になっていません。以降、赤外線リモコン受信 API の使用方法について解説します。なお、以下に記載されている赤外線リモコン機能の用語等については、『i アプリコンテンツ開発ガイド 詳細編』の赤外線リモコンの項も参考にしてください。

赤外線リモコン受信 API は、`com.docomostar.opt.device` パッケージで定義された、赤外線リモコン受信機能を扱う `IrReceiver` クラスと、赤外線リモコン信号の波形パターンをパラメータとして与えるための `IrReceiverParam` クラスで提供されます。

赤外線リモコン受信 API を使用するには、まず最初に `IrReceiver` クラスの static メソッド `getIrReceiver()` を呼び出して `IrReceiver` オブジェクトを取得します。

リモコン信号を受信するためのオペレーションは、大きく以下の 2 つに分かれます。

- `IrReceiverParam` によるリモコン信号パラメータの生成
- `IrReceiver` によるリモコン信号の受信と受信データの取り出し

(a) リモコン信号パラメータの生成

リモコン信号はスタート部、ストップ部、データ部（論理 0）、データ部（論理 1）の 4 つの要素で構成されています。またこれらの各要素は、それぞれが High 区間（赤外線信号が ON になっている区間）と Low 区間（赤外線信号が OFF になっている区間）の 2 つの区間から成り立つ波形で表現されます。赤外線リモコン受信機能では、これから受信しようとしている赤外線リモコン信号の波形パターンの情報を、`IrReceiverParam` オブジェクトの形で、パラメータとしてシステムに通知します。

i アプリはスタート部、ストップ部、データ部（論理 0）、データ部（論理 1）のそれぞれ（計 4 つ）について `IrReceiverParam` オブジェクトを作成します。`IrReceiverParam` では、コンストラクタの引数に各波形の High 区間、Low 区間の長さ（単位 μ 秒）を設定します。

なお、i アプリはこれらの各リモコンパラメータについて、最小値と最大値の 2 つの値を設定する必要があります。リモコン信号送信側デバイスの精度や誤差等を考慮して、適切な幅を持たせた値を設定するようにしてください。

IrReceiverParam オブジェクトに設定されている内容は、アクセッサメソッドを使用して確認することができます。

参考情報として、スタート部、ストップ部、データ部（論理 0）、データ部（論理 1）の各パラメータが、i アプリ基本 API の赤外線リモコン機能（**IrRemoteControl**、**IrRemoteControlFrame**）のどの設定項目に対応するものであるかを下記に記載します。

- ・ スタート部の High 区間長
この項目は、**IrRemoteControlFrame.setStartHighDuration()**メソッドによる設定項目に対応します。
- ・ スタート部の Low 区間長
この項目は、**IrRemoteControlFrame.setStartLowDuration()**メソッドによる設定項目に対応します。
- ・ ストップ部の High 区間長
この項目は、**IrRemoteControlFrame.setStopHighDuration()**メソッドによる設定項目に対応します。
- ・ ストップ部の Low 区間長
この項目は、**IrRemoteControlFrame** クラスの設定項目には存在しません
（**IrRemoteControlFrame** では、ストップ部には Low 区間を設定できません）。基本 API の赤外線リモコン機能では、1 フレーム分の信号を送出後、次のフレームの信号が送出されるまでの無発光時間がこのパラメータに相当します。
- ・ データ部（論理 0）の High 区間長
この項目は、**IrRemoteControl.setCode0()**メソッドで設定する論理 0 の High 区間長に対応します。
- ・ データ部（論理 0）の Low 区間長
この項目は、**IrRemoteControl.setCode0()**メソッドで設定する論理 0 の Low 区間長に対応します。
- ・ データ部（論理 1）の High 区間長
この項目は、**IrRemoteControl.setCode1()**メソッドで設定する論理 1 の High 区間長に対応します。
- ・ データ部（論理 1）の Low 区間長
この項目は、**IrRemoteControl.setCode1()**メソッドで設定する論理 1 の Low 区間長に対応します。

なお本機能では、データフレームに格納されるビットデータの波形を High 先行パターン、Low 先行パターンのいずれとして解釈すべきであるかを指定することはできません。本機能では、ビットデータ部分は High 先行パターンの波形で送信されてくることを前提としています。

(b) リモコン信号の受信と受信データの取り出し

i アプリは、**IrReceiver.receive()**メソッドを呼び出すことでリモコン信号の受信を開始します。このメソッドの引数には、前項で生成した 4 つの **IrReceiverParam** オブジェクトと、データ部（論理 0）波形およびデータ部（論理 1）波形の基準長（T 時間）に関する情報を指定します。一般的に家電製品協会フォーマットと呼ばれるリモコン信号では、データ部（論理 0）の波形とデータ部（論理 1）の波形は概ね以下のように設計されます。

データ部（論理 0）High 区間長 = データ部（論理 0）Low 区間長 = データ部（論理 1）High 区間長
（この長さを T とします）

データ部（論理 1）Low 区間長 = 3T

データ部（論理 0）波形およびデータ部（論理 1）波形の基準長では、上記式の基準となる時間 T の長さを指定します（前項で解説した他のパラメータと同様、リモコン信号送信側デバイスの精度や誤差等を考慮した最大値と最小値の 2 つを指定します）。

receive() メソッドによって受信が開始されると、リモコン信号は **receive()** メソッドに指定されたパラメータに従って解釈され、その内容がシステム内部のバッファに蓄積されます。

i アプリは、蓄積されたデータを、**IrReceiver.getInputStream()** で取得したストリームを使って読み出すことができます。このストリームからは、**IrReceiver** クラス独自の形式でリモコン信号の読み出しを行います。このストリームから読み出すデータは、以下の 4 パターンの 4 ビットデータの羅列となります。

- 0xE（2 進表現：1110）： スタート部を表すパターン
- 0xF（2 進表現：1111）： ストップ部を表すパターン
- 0x0（2 進表現：0000）： 論理 0 を表すパターン
- 0x1（2 進表現：0001）： 論理 1 を表すパターン

例えば、0x A5 20 FF 01（2 進表現：10100101 00100000 11111111 00000001）というビットデータを含むデータフレームは、このストリームを介して読み出すと以下のような 17 バイトのデータになります。

0x E1 01 00 10 10 01 00 00 01 11 11 11 10 00 00 00 1F

アプリケーションプログラムでは、ストリームから読み出したデータを 4 ビットずつ区切って解釈し、上記パターンに従った変換を行ってデータフレームの内容を得るようにしてください。

リモコン信号の受信を停止するには、**IrReceiver.cancel()** メソッドを使用します。

注意事項：

- データフレームに格納されるビットデータ長は必ずしもバイト単位（8 ビットの倍数）である必要はありません。端数ビットを含むデータを受信した場合、ストリームから読み出したデータ（読み出したデータを上記パターンにしたがって変換した結果）は、末尾がバイト単位になるようにゼロパディングされます。
- **IrReceiver** クラスでは、ビットデータを含まないデータフレームを受信して読み出すこともできます。その場合、ストリームからは、スタート部のパターンとストップ部のパターンのみからなる 1 バイトのデータ（0xEF）が読み出されます。

【Star-1.3】

本機能は Star-1.3 プロファイルで新設されました。

第 25 章

歩行方位取得機能

歩行方位取得機能は、携帯電話が備える加速度センサーや電子コンパスを用いて、i アプリ実行中の任意の時点、もしくは i アプリ実行中の特定の期間内で携帯電話ネイティブの歩数計機能が蓄積する歩数情報や携帯電話の方位角を取得する機能です。歩数情報や方位角を調べることで、携帯電話を持つユーザーがどの向きに歩を進めているかを知ることができるため、建物の中や地下など GPS による位置情報取得が困難な環境に移動した場合においても動作可能な簡易的なナビゲーション機能を実現することができます。

歩行方位取得機能は `com.docomostar.device.location` パッケージで提供されます。歩行方位取得機能のクラス構成は以下のとおりです。

`com.docomostar.device.location` パッケージ



クラス	概 要
WalkingNavigation	歩行方位取得機能を提供するクラスです。アプリケーションプログラムはこのクラスを使用して歩数情報や方位角の計測を行います。計測の方法として、アプリケーションプログラムが指示している間、歩行(携帯電話ネイティブの歩数計で一歩がカウントされたこと)が検知される度に計測を行って結果を蓄積する歩行検知計測と、歩行の検知に関わらず現在の歩数情報と方位角を取得できるワンショット計測があります。 計測の結果は次項に記載する <code>WalkingNavigationData</code> クラスのオブジェクトとして表現されます。
WalkingNavigationData	歩行方位取得機能の計測結果を表すクラスです。このオブジェクトは携帯電話ネイティブの歩数計が保持するその日 0 時 0 分 0 秒から計測時点までの歩数情報と、携帯電話の向きを表す方位角を保持しています。以降、本書では歩数情報と方位角を合わせて歩行情報と呼びます。 歩行検知計測では、計測期間内の 1 歩ごとに計測データが記録されるため、1 つの <code>WalkingNavigationData</code> オブジェクトには複数件の計測結果が格納される場合があります。

歩行方位取得機能を利用するには、まず `WalkingNavigation` クラスの static メソッド `getWalkingNavigation()` を呼び出して `WalkingNavigation` オブジェクトを取得します。以下に、`WalkingNavigation` クラスを使用した、歩行方位取得機能について解説します。

(1) ワンショット計測と歩行検知計測

ワンショット計測とは、必要なタイミングでアプリケーションプログラムがシステムに計測指示を出し、計測を行う方法です。また、歩行検知計測とは、アプリケーションプログラムが指示している間、ユーザーが1歩歩行するごとにシステムが自動的に計測を行ってその結果を蓄積していき、アプリケーションプログラムが必要とするタイミングでその結果を取り出す方法です。

ワンショット計測は、アプリケーションプログラムが任意のタイミングで **WalkingNavigation.getCurrentData()** メソッドを呼び出すことにより行われます。

また歩行検知計測を行うには、**WalkingNavigation.start()** メソッドにより歩行検知計測を開始した上で、任意のタイミングで **WalkingNavigation.getData()** メソッドあるいは **WalkingNavigation.peekLatestData()** メソッドを呼び出します。**getData()** メソッドは全ての計測結果を返しますが、**peekLatestData()** メソッドは蓄積されている計測結果のうち最新の1件のみを返します。開始された歩行検知計測は **WalkingNavigation.stop()** メソッドを呼び出すと停止します。

getCurrentData()、**getData()**、**peekLatestData()** の各メソッドは **WalkingNavigationData** オブジェクトの形で計測結果を返します。システムはあらかじめ **getCurrentData()**、**getData()**、**peekLatestData()** の各メソッドごとに専用の **WalkingNavigationData** オブジェクトを割り当てて保持しており、個々のメソッドはそれぞれ常に同じオブジェクトを返すようになっています。**WalkingNavigationData** オブジェクトの利用方法については、次項を参照してください。

WalkingNavigation には歩行検知計測においてシステム内部に蓄積可能な最大データ数を調べるための **getMaxDataSize()** メソッドが定義されています。最大データ数を超えて歩行検知計測が行われた場合、蓄積されたデータの古いものから順次上書きされていきます。

歩行検知計測の計測結果が何件蓄積されているかは、**WalkingNavigation.getDataSize()** メソッドを使用して調べることができます。

(2) 計測結果の利用

歩行方位取得機能の計測結果は、**WalkingNavigationData** クラスで表現されます。このクラスには、歩行方位取得機能で計測した歩行情報を取り出すための以下のメソッドが用意されています。

getPedometerData()	携帯電話ネイティブの歩数計が保持している、1日分の歩数情報（その日0時0分0秒から計測時点までの歩数情報）を取得します。 このメソッドは com.docomostar.system.PedometerData オブジェクトの配列を返します。 PedometerData オブジェクトの詳細は17.1項を参照してください。
getPhoneAzimuth()	携帯電話の向きの方位角の値を取得します。 このメソッドは方位角を表す double 型の値の配列を返します。 個々の値の単位は DEGREE です。

WalkingNavigationData オブジェクトは計測した歩行情報を内部で配列の形で保持しているため、**getPedometerData()**、**getPhoneAzimuth()** の各メソッドもそれぞれデータを配列の形で返すようになっています。

ワンショット計測では、**getPedometerData()**、**getPhoneAzimuth()** メソッドが返す配列の先頭要素にそれぞれ、**WalkingNavigation.getCurrent()** メソッド呼び出し時点における、携帯電話ネイティブの歩数計が保持している歩数情報と携帯電話の向きの方位角が設定されています。

歩行検知計測では、計測を開始した後、歩行（携帯電話ネイティブの歩数計で一歩がカウントされたこと）が検知される度に計測を行って結果が蓄積されます。そのため、歩行検知計測を開始した後、10歩カウントされている状態では、**getPedometerData()**、**getAzimuth()** メソッドが返す配列の有効な長さはどちらも10になります。また、**getPedometerData()** メソッドが返す **PedometerData** オブジェクトには歩数計機能（17.1項）における **PedometerData** オブジェクトと同様、1日分（0時0分0秒からその時点までの累積歩数）の歩数情報が格納されています。

たとえば、携帯電話ネイティブの歩数計で 1000 歩カウントされている状態で、歩行検知計測を開始し 3 歩歩いた時点では `getPedometerData()`、`getPhoneAzimuth()` の各メソッドが返す配列の各要素には以下のように歩数情報が格納されています。

・ 配列 0 番目	1001 歩分の歩数情報を格納した PedometerData オブジェクト。 1001 歩目をカウントした際の携帯電話の向きの方位角。
・ 配列 1 番目	1002 歩分の歩数情報を格納した PedometerData オブジェクト。 1002 歩目をカウントした際の携帯電話の向きの方位角。
・ 配列 2 番目	1003 歩分の歩数情報を格納した PedometerData オブジェクト。 1003 歩目をカウントした際の携帯電話の向きの方位角。

また、`WalkingNavigation.getData()` メソッドを呼び出して計測結果を取得すると、システム内部の蓄積バッファはクリアされるため、`getData()` メソッドを呼び出した後、更に 3 歩歩いた時点で計測結果を取得した際には、以下のように歩数情報が格納されています。

・ 配列 0 番目	1004 歩分の歩数情報を格納した PedometerData オブジェクト。 1004 歩目をカウントした際の携帯電話の向きの方位角。
・ 配列 1 番目	1005 歩分の歩数情報を格納した PedometerData オブジェクト。 1005 歩目をカウントした際の携帯電話の向きの方位角。
・ 配列 2 番目	1006 歩分の歩数情報を格納した PedometerData オブジェクト。 1006 歩目をカウントした際の携帯電話の向きの方位角。

`WalkingNavigation.getData()` メソッド用に割り当てられた `WalkingNavigationData` オブジェクトは、歩行検知計測でシステムが蓄積可能な件数を格納するのに十分な長さの配列をあらかじめ保持しています。実際に蓄積されている件数が少ない状況では配列の先頭要素から順に有効なデータが設定されますが、その有効な要素数は `size()` メソッドにより調べることが出来ます。

なお、計測結果に含まれる方位角は、携帯電話本体の向いている方を磁北から時計回りを正とする角度として表されます。i アプリ起動直後の携帯電話本体の向いている方は概ねディスプレイの上部が向いている方角ですが、計測を開始する前に、方位デバイスの基準方向を変更することで、設定した方位デバイスの基準方向が反映された方角を取得することが出来ます。方位デバイスの基準方向の詳細は 5 章を参照してください。

(3) 歩行検知計測停止イベント

歩行検知計測中に何らかの理由により歩行検知計測が停止したことをイベントとして i アプリに通知することが出来ます。歩行検知計測停止をイベントとして受け取るには

`StarApplication.addEventListener()` メソッドを使用して、イベント種別

`StarEventObject.STAR_WALKING_NAVIGATION_STOPPED` に対応するリスナーオブジェクト

(`StarEventListener` インタフェースを実装したオブジェクト) を登録します。i アプリ実行中に歩行検知計測が停止すると登録したリスナーオブジェクトの `updateStarApplication()` メソッドが呼び出されます。

注意事項：

- 歩行方位取得機能を利用する i アプリは、ADF の `AccessUserInfo` キーにてこの機能を使用することを宣言（値 "yes" を設定）する必要があります。
- 歩行検知計測によってシステム内部に蓄積された計測結果は、以下の契機でクリアされます。
 - `WalkingNavigation.getData()` メソッドによって、歩行検知計測の結果が取り出されたとき
 - 歩行検知計測を停止した後、歩行検知計測を開始したとき

- `WalkingNavigation.disposeData()`メソッドによって明示的に計測結果のクリアが指示されるか、もしくは計測結果を蓄積したままiアプリを終了したとき

歩行検知計測を停止した後も、これらの契機までは計測結果はシステム内部に蓄積されたままとなり、`WalkingNavigation.getData()`メソッドで結果を取り出すことができます。

- 歩行検知計測と以下の機能は同時に利用することはできません。
 - 電子コンパス機能の周期計測
 - 加速度センサ機能の周期計測あるいは加速度変化のイベント検知
- 歩行検知計測でシステム内部に蓄積可能な歩行情報の数は機種によって異なりますが、機種によっては数10件というオーダーの場合もあります。そのため、`getMaxDataSize()`メソッドを呼び出して、システム内部に蓄積可能な最大データ数を調べた上で、適切な頻度で計測結果を取り出すようにしてください。

【Star-1.5】

本機能はStar-1.5プロファイルで新設されました。

第 26 章

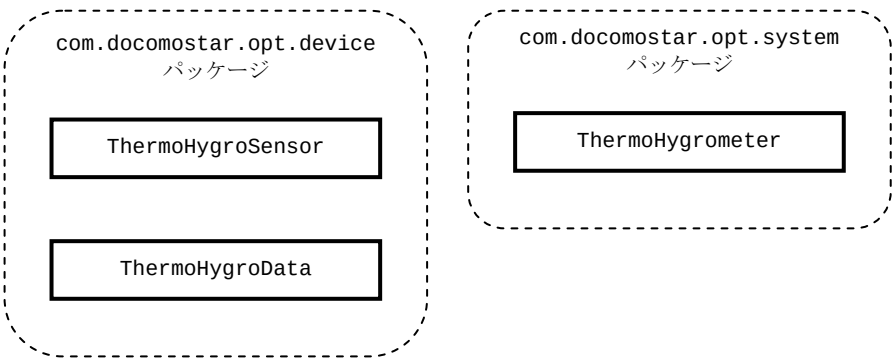
温湿度計機能

メーカーによっては、携帯電話の付加機能として、温度や湿度を測定可能なセンサーデバイスを搭載しているものがあります。このような機種のため、i アプリから温湿度センサーや携帯電話ネイティブの温湿度計にアクセスするためのオプション API が定義されています。これらの機能を利用することで、温湿度センサや携帯電話ネイティブの温湿度計から温度や湿度を取得して i アプリで使うことができます。

温湿度計機能では、以下の 2 つの機能が提供されています。

- 現在の温度・湿度を取得するワンショット計測機能
- 携帯電話ネイティブの温湿度計が普段定期的に測定、保存している温度・湿度を取得するネイティブ保存情報取得機能

以下に、温湿度計機能のクラス構成を示します。



com.docomostar.opt.device パッケージ

クラス	機能概要
ThermoHygroSensor	温湿度センサーを表すクラスです。現在の温度・湿度を得るワンショット計測機能を提供します。
ThermoHygroData	温湿度計機能による計測結果を表すクラスです。ワンショット計測機能により取得する温度・湿度情報も、ネイティブ保存情報取得機能により取得する温度・湿度情報も、このクラスのオブジェクトとして取得することができます。

com.docomostar.opt.system パッケージ

クラス	機能概要
ThermoHygrometer	携帯電話ネイティブの温湿度計を表すクラスです。携帯電話ネイティブの温湿度計が保存している温度・湿度を取得する、ネイティブ保存情報取得機能を提供します。

(a) ワンショット計測

温度・湿度のワンショット計測機能の API は `com.docomostar.opt.device` パッケージで提供されます。ワンショット計測は以下の 2 つのクラスで構成されます。

- `ThermoHygroSensor` : 温湿度センサーを表現するクラス
- `ThermoHygroData` : 温湿度センサーで計測された温度、湿度、計測日時を表すクラス

ワンショット計測機能を利用するには、まず `ThermoHygroSensor` クラスの static メソッド `getThermoHygroSensor()` を呼び出して `ThermoHygroSensor` オブジェクトを取得します。現在の温度・湿度を取得するには `ThermoHygroSensor` クラスの `getCurrentData()` を呼び出して `ThermoHygroData` オブジェクトを取得します。

`ThermoHygroData` オブジェクトから以下のメソッドで情報を取得することができます。

`getTemperature()` : 温度を取得します。
`getHumidity()` : 湿度を取得します。
`getCalendar()` : 温度・湿度が計測された時点の日時を取得します。

また、`ThermoHygroSensor` クラスには温湿度センサー機能がサポートしている機能・能力を知るための以下のようなメソッドが用意されています。

`getAvailableData()` : 温湿度センサー機能で、どのデータを取得できるかを調べるメソッドです。温度、湿度のデータ項目のうち、どの項目をサポートしているかを調べることができます。このメソッドで返されるデータ項目種別は、`ThermoHygroSensor` クラスの定数として定義されています。
`getMaxDataValue()` : 各計測データ項目について、取り得る値の最大値を調べるメソッドです。
`getMinDataValue()` : 各計測データ項目について、取り得る値の最小値を調べるメソッドです。

(b) 携帯電話ネイティブの温湿度計からの情報取得

温湿度計機能は、`com.docomostar.opt.system.ThermoHygrometer` クラスで提供されます。アプリケーションプログラムが `ThermoHygrometer` オブジェクトを取得するには `ThermoHygrometer` クラスの static メソッドである `getThermoHygrometer()` メソッドを呼び出します。

`ThermoHygrometer` オブジェクトからはネイティブの温湿度計が現在蓄積している温度・湿度情報や、温湿度計がサポートしている計測可能データの種類の取得することができます。

ネイティブの温湿度計から温度・湿度情報を取得するメソッドとして、`ThermoHygrometer.getData()` メソッドが用意されています。このメソッドはネイティブの温湿度計が現在保持するすべての温度・湿度情報を取得するものと、特定の期間内に計測された温度・湿度情報を取得するために開始時刻と終了時刻を指定するものがあります。`getData()` メソッドは、ネイティブの温湿度計が保存している温度・湿度情報を

ThermoHygroData オブジェクトの配列として返します。ThermoHygroData オブジェクトの使い方については前項のワンショット計測を参照してください。

ネイティブ温湿度計からどのデータ項目が取得可能かを調べるには `getAvaliableData()` メソッドを呼び出します。また、`getCount()` メソッドを呼び出すことで、温湿度計が現在保持している温度・湿度情報の個数を取得することができます。

なお、この機能では実際に温度・湿度を計測する機能、および計測した温度・湿度データを蓄積する機能はあくまでも携帯電話ネイティブの温湿度計が持っています。従って、i アプリがこの機能を使用するためには、ユーザーが携帯電話ネイティブの温湿度計をオンにしていることが前提となっています。

携帯電話ネイティブの温湿度計がオンになっているかどうかは `ThermoHygrometer` クラスの `getAttribute()` メソッドで調べることができます。ただし i アプリから温湿度計のオン・オフを制御することはできません。

注意事項：

- 温湿度センサーを搭載している機種であっても、本章で説明した機能をサポートしていない場合もあります。

【Star-1.5】

本機能は Star-1.5 プロファイルで新設されました。

第 27 章

音楽プレイヤー機能

音楽プレイヤー機能は、データ BOX に保存されている音楽コンテンツの再生や管理を i アプリから行なえるようにするための機能です。音楽プレイヤー機能を利用すると、i アプリのリソース制約に制限されずにフル楽曲などの大容量の音楽コンテンツを再生したり、楽曲名やアーティスト名などで音楽コンテンツをフィルタリングしたプレイリストを作成したりというように、一般的な音楽プレイヤーが備える機能を i アプリで実現できるようになります。また、音楽プレイヤー機能の音楽再生機構は i アプリと並行してバックグラウンドで動作するため、音楽コンテンツを i アプリの BGM として再生したり、音楽コンテンツの再生中に音楽コンテンツに含まれるジャケット画像やタイトル、アーティスト名を取得して画面に描画するなど、自由にユーザーインターフェースを構築することができるようになります。

注意事項：

- 音楽プレイヤー機能をサポートする機種で共通して利用可能な音楽コンテンツはフル楽曲コンテンツです。また、メーカーによってはフル楽曲コンテンツに加え、MP3、WMA をサポートする場合があります。なお、フル楽曲コンテンツのファイル仕様、作成方法は NTT ドコモから一般公開は行っておりません。

27.1 音楽プレイヤー機能の API 構成

音楽プレイヤー機能は、以下のパッケージに属するクラス、インタフェースから構成されます。

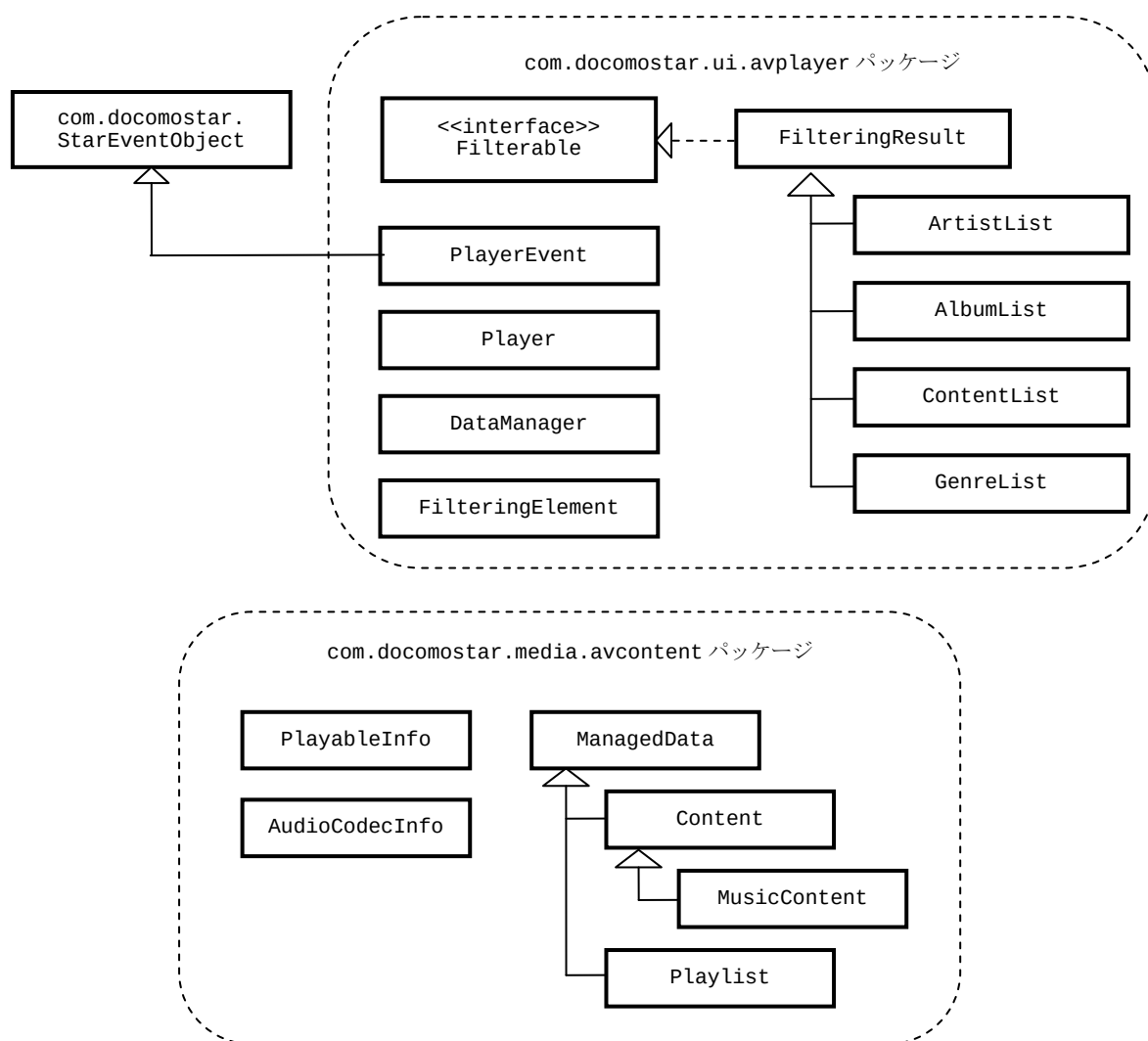
com.docomostar.ui.avplayer パッケージ

音楽コンテンツの再生や停止といった音楽プレイヤー機能や、音楽プレイヤーで利用するデータベース管理機能、音楽コンテンツのフィルタリング結果を表すクラスを規定しています。

com.docomostar.media.avcontent パッケージ

音楽プレイヤーで扱われるデータ（音楽コンテンツそのものやプレイリストなど）を表すクラスを規定しています。

以下に、音楽プレイヤー機能のクラス・インタフェース構成を図示します。



各クラス、インタフェースの機能概要は以下の通りです。

com.docomostar.ui.avplayer パッケージ

クラス、インタフェース	機能概要
DataManager	音楽プレイヤー機能で利用するデータベース（データ BOX など）を管理するクラスです。データベースの更新機能、データベースからコンテンツやプレイリストを取得する機能を提供します。
Player	音楽プレイヤーを表すクラスです。アプリケーションプログラムはこのクラスを使用してコンテンツの再生制御を行います。
PlayerEvent	再生開始や停止、曲の切り替わりなど、音楽プレイヤーから通知されるイベントを表すクラスです。
Filterable	フィルタリング可能なコンテンツが実装するインタフェースです。検索対象として、アーティスト、アルバム、ジャンルのいずれかを指定することができます。
FilteringElement	フィルタリングの際の検索条件を表すクラスです。
FilteringResult	コンテンツに対してフィルタリング操作を行った、フィルタリング結果を表すクラスです。 Filterable インタフェースを実装しているため、フィルタリング結果に対して更に条件を追加してフィルタリングすることができます。ただし、再フィルタリング時にどのような条件を指定できるかは機種によって異なります。

ContentList	コンテンツのフィルタリング結果のうち、コンテンツの一覧を表します。音楽プレイヤーで再生可能なコンテンツを保持しています。
AlbumList	コンテンツのフィルタリング結果のうち、アルバムの一覧を表します。このアルバム一覧が保持する各コンテンツのアルバム名のリストを取得したり、更にアルバム名を検索条件としてフィルタリングを行い、そのフィルタリング結果をコンテンツ一覧 (ContentList) として取得することができます。
ArtistList	コンテンツのフィルタリング結果のうち、アーティストの一覧を表します。このアーティスト一覧が保持する各コンテンツのアーティスト名のリストを取得したり、更にアーティスト名を検索条件としてフィルタリングを行い、そのフィルタリング結果をアルバム一覧 (AlbumList) として取得することができます。
GenreList	コンテンツのフィルタリング結果のうち、ジャンルの一覧を表します。このジャンル一覧が保持する各コンテンツのジャンル名のリストを取得したり、更にジャンル名を検索条件としてフィルタリングを行い、そのフィルタリング結果をコンテンツ一覧 (ContentList) として取得することができます。

com.docomostar.media.avcontent パッケージ

クラス	機能概要
ManagedData	音楽コンテンツそのものやプレイリストなど、音楽プレイヤー機能で利用するデータベース（データ BOX など）によって管理されているデータを表す基底クラスです。
Content	音楽プレイヤーで再生可能なコンテンツを表すクラスです。
MusicContent	音楽プレイヤーで再生可能なコンテンツのうち、音楽コンテンツを表すクラスです。
Playlist	プレイリストを表すクラスです。
AudioCodecInfo	コンテンツのオーディオコーデック情報を表すクラスです。
PlayableInfo	コンテンツの再生制限情報を表すクラスです。

27.2 音楽コンテンツ管理データベースへのアクセス

音楽プレイヤー機能では、音楽コンテンツを管理するデータベース（データ BOX など）へアクセスするための **DataManager** クラスが定義されています。**DataManager** クラスはデータベースからの音楽コンテンツやプレイリストの取得、音楽コンテンツのアーティスト、アルバムあるいはジャンルによるフィルタリング、データベースの更新などの機能を提供します。

また、取得したコンテンツやフィルタリング結果からプレイリストを生成することができ、生成したプレイリストを音楽プレイヤーに設定することで、プレイリストに含まれる音楽コンテンツを音楽プレイヤーで再生することができます。

音楽コンテンツ管理データベースへアクセスする i アプリは、ADF の **AccessUserInfo** キーにてこの機能を使用することを宣言（値 "yes" を設定）する必要があります。

なお、**DataManager** クラスのメソッドは全て **static** メソッドであり、アプリケーションプログラムは **DataManager** クラスのオブジェクトを生成することはできません。

(1) 音楽コンテンツの取得とフィルタリング

i アプリがデータベースからコンテンツを取得する方法には、検索条件を指定せずコンテンツ一覧やアルバム一覧などを取得する方法と、検索条件を指定してコンテンツ一覧やアルバム一覧などを取得する方法とがあります。

検索条件を指定せずにデータベースからコンテンツ一覧を取得するには、**DataManager** クラスの static メソッド **getAllContents()** を使用します。引数には、ファイル種別として **FilteringResult** クラスに定義された **FILE** 定数を指定します。ただし、現在のプロファイルではファイル種別として、音楽コンテンツを表す **FILE_MUSIC** のみ定義されています。**getAllContents()** メソッドに **FilteringResult.FILE_MUSIC** を指定して呼び出すと、全ての音楽コンテンツを含む **ContentList** オブジェクトを返します。

取得した **ContentList** オブジェクトからは **Content** オブジェクトの配列の形でコンテンツを取得することができ、(3)項で解説するミュージック情報を取得することができます。

また、**DataManager** クラスを使用して、全音楽コンテンツを対象にアルバム一覧、アーティスト一覧、ジャンル一覧を取得することもできます。検索条件を指定せずにデータベースからアルバム一覧、アーティスト一覧、ジャンル一覧を取得するには、それぞれ **getAllAlbums()**、**getAllArtists()**、**getAllGenres()** の各メソッドを使用します。これらのメソッドにファイル種別として **FilteringResult.FILE_MUSIC** を指定して呼び出すと、それぞれ **AlbumList**、**ArtistList**、**GenreList** の各オブジェクトを返します。

検索条件を指定してデータベースからコンテンツ一覧を取得するには、**DataManager** クラスの static メソッド **getContents()** を使用します。このメソッドに以下の項目を設定して呼び出すと、検索条件にマッチするコンテンツを含む **ContentList** オブジェクトを返します。

- ファイル種別
取得するコンテンツのファイル種別を指定します。現在のプロファイルでは **FilteringResult.FILE_MUSIC** のみ指定可能です。
- 検索対象（フィルタリング対象タイプ）
コンテンツのどの情報を検索の対象とするかを指定します。現在のプロファイルでは、以下のいずれかが指定可能です。
 - ・アーティスト (**Filterable.TYPE_ARTIST**)
 - ・アルバム (**Filterable.TYPE_ALBUM**)
 - ・ジャンル (**Filterable.TYPE_GENRE**)
- 検索文字列
検索したい文字列を指定します。指定した文字列と完全一致したコンテンツのみが検索条件にマッチしたことになります。なお、全角・半角は区別されますが、大文字・小文字が区別されるかどうかは機種によって異なります。

また、検索条件を指定してデータベースからアルバム一覧、アーティスト一覧、ジャンル一覧を取得するには、それぞれ **getAlbums()**、**getArtists()**、**getGenre()** の各メソッドを使用します。これらのメソッドに検索条件を指定して呼び出すと、検索条件にマッチした **AlbumList**、**ArtistList**、**GenreList** の各オブジェクトを返します。検索条件の指定方法は、上述の **getContents()** メソッドと同様です。

各種一覧を表す **ContentList**、**AlbumList**、**ArtistList**、**GenreList** の各クラスは、**Filterable** インタフェースを実装した **FilteringResult** クラスのサブクラスであるため、これらのクラスのオブジェクトからプレイリストを生成したり、アーティスト、アルバム、ジャンルを検索条件に指定してフィルタリングしたりすることができます。

検索条件を指定してフィルタリングするには **Filterable** インタフェースに定義されている **filter()** メソッドを使用します。このメソッドにフィルタリング結果タイプ、検索対象、検索文字列を指定して呼び出すことで、指定された検索条件でフィルタリングすることができます。フィルタリング結果タイプはフィルタリング結果として何の一覧を取得するのかを指示する情報で、**filter()** メソッドが返す **FilteringResult** オブジェクトの実際の型に対応します。指定可能なフィルタリング結果タイプとメソッドが返すオブジェクトの型は以下のとおりです。

フィルタリング結果タイプ	メソッドが返すオブジェクトの型
アーティスト (Filterable.TYPE_ARTIST)	ArtistList
アルバム (Filterable.TYPE_ALBUM)	AlbumList

ジャンル (Filterable.TYPE_GENRE)	GenreList
コンテンツ (Filterable.TYPE_CONTENT)	ContentList

filter()メソッドにフィルタリング結果タイプ、検索対象や検索文字列を与えるには、以下の2つの方法があります。

- これらの項目を **filter()**メソッドの引数として列挙する。
- これらの項目をカプセル化する **com.docomostar.ui.avplayer.FilteringElement** オブジェクトを生成し、**filter()**メソッドの引数に指定する。

各フィルタリング結果オブジェクト (**ContentList**、**AlbumList**、**ArtistList**、**GenreList** の各オブジェクト) に対して、更に別の検索条件を指定してフィルタリングすることができる場合があります(*1)。これら一覧から再フィルタリングを行う際に、携帯電話が検索条件としてどの検索対象をサポートしているかを調べるために **FilteringResult** クラスに **getAvailableTargetType()**メソッドが定義されています。また、各フィルタリング結果を得るまでに使用された全ての検索条件を取得する **getFilteringElements()**メソッドも定義されています。このメソッドは、使用された検索条件を **FilteringElement** オブジェクトの配列の形で返します。

- (*1) フィルタリング結果に対して、さらにどのような条件で再フィルタリングの操作が行えるかは機種によって異なります。どの機種でも行える再フィルタリング操作は、「アーティスト名を指定してフィルタリングを行った結果に対するアルバム名を指定しての再フィルタリング」のみです。どの機種でもサポートされる再フィルタリングの条件は API リファレンスの **Filterable** インタフェース説明を参照してください。

(2) プレイリストの取得

プレイリストとは、データ BOX に含まれる音楽コンテンツを再生する順番をリスト化したもので、音楽プレイヤー機能ではプレイリストを使用して、どのコンテンツを再生するのかを指定します。

i アプリがプレイリストを取得する方法には以下の2通りがあります。

- 音楽管理データベースに保存されているプレイリストを取得する
- 音楽管理データベースから取得したフィルタリング結果オブジェクト (**FilteringResult** のサブクラスのオブジェクト) からプレイリストを生成する

音楽管理データベースに保存されているプレイリストを取得するには、引数に音楽コンテンツを表す **FilteringResult.FILE_MUSIC** を指定して、**DataManager.getAllPlaylists()**メソッドを呼び出します。このメソッドは音楽データベースに保存されているプレイリストの一覧を **Playlist** オブジェクトの配列の形で返します。

フィルタリング結果からプレイリストを生成するには、各フィルタリング結果オブジェクトの **getPlaylist()**メソッドを使用します。**ContentList**、**ArtistList**、**AlbumList**、**GenreList** の各オブジェクトに対してこのメソッドを呼び出すと、フィルタリング結果内の音楽コンテンツが含まれるプレイリストを **Playlist** オブジェクトとして取得することができます。

また、**Playlist** オブジェクトに対して **getContents()**メソッドを呼び出すことによって、プレイリストに含まれるコンテンツの一覧を **Content** オブジェクトの配列の形で取得することもできます。

(3) ミュージック情報の取得

ミュージック情報とは、音楽コンテンツが持つアーティスト名、アルバム名、ジャケット画像、再生制限情報などをいいます。i アプリでは音楽コンテンツ管理データベースから取得した音楽コンテンツからミュージック情報を取得することができます。

アプリケーションプログラムは **ContentList** から **getContents()** メソッドを使用して各々のコンテンツに対応する **Content** オブジェクトを取得することができますが、**DataManager** クラスへのアクセスで **FilteringResult.FILE_MUSIC** をした場合（現在のプロファイルではこの値のみ指定可能）は、このオブジェクトは **Content** クラスのサブクラスである **MusicContent** クラスのオブジェクトとなっています。アプリケーションプログラムは **MusicContent** クラスのメソッドを使用してミュージック情報を参照することができます。

i アプリから取得可能なミュージック情報は以下のとおりです。

- | | |
|---------------------------------------|--------------------------|
| ・ファイル名 | ・ファイル種別 |
| ・タイトル | ・アーティスト |
| ・アルバム | ・年 |
| ・ジャンル | ・コメント |
| ・トラック番号および総トラック数 | ・作曲者 |
| ・作詞者 | ・権利者 |
| ・販売元 | ・権利情報 |
| ・レーベル | ・URL 情報 |
| ・保存日時 | ・ファイルサイズ |
| ・取得元 | ・ファイル制限 |
| ・再生時間 | ・本体と外部メモリ間の移動可否 |
| ・オーディオコーデック (AudioCodecInfo) | ・着信音設定可否（まるごと、オススメ、おこのみ） |
| ・再生制限情報 (PlayableInfo) | ・待受用画像およびその保存可否 |
| ・ジャケット画像およびその保存可否 | |

なお、**MusicContent** オブジェクトから取得したミュージック情報はオブジェクト内にキャッシュされ、キャッシュのために Java ヒープを消費します。ジャケット画像や待受画像は他のミュージック情報に比べメモリの消費量が多いため、取得した画像が不要になったら、**Content.unload()** メソッドを呼び出し、キャッシュしたミュージック情報を解放するようにしてください。

(4) 音楽管理データベースの更新

音楽プレイヤー機能では、メディアアクセスによるオーバーヘッドをなくすため、音楽プレイヤー機能を利用する際にネイティブの音楽管理データベースのコピーを作ります。コピーされた音楽管理データベースは、外部メモリが挿抜されたり、コンテンツ取得保存機能によってコンテンツを取得するなどした契機で、ネイティブの音楽管理データベースとの間で不整合が生じる場合があります。コピーされた音楽管理データベースとネイティブの音楽管理データベースとの間で情報に不整合が生じた場合、i アプリはコピーの音楽管理データベースを更新する必要があります。

音楽プレイヤー機能では、音楽管理データベースの更新が必要かどうかを調べるメソッドが **DataManager.isDirtyDataBase()** として定義されています。音楽プレイヤーからデータベースの更新が必要である旨のイベント通知を受信した場合（27.4 項も参照）や i アプリがレジューム復帰した際にこのメソッドを呼び出して、音楽管理データベースの更新が必要かどうかチェックするようにしてください。音楽管理データベースの更新が必要な場合は **DataManager** クラスの static メソッド **updateDataBase()** メソッドを使用します。

なお、音楽管理データベースの更新が必要な状態では、音楽管理データベースからプレイリストやコンテンツを新たに取得することも、それまでに取得したプレイリストやコンテンツを利用することもできませ

ん。音楽管理データベースを更新したのち、再度 **DataManager** クラスのメソッドを使用してプレイリストやコンテンツを取得しなおしてください。

注意事項：

- 音楽コンテンツ管理データベースにアクセスする i アプリは、ADF の **AccessUserInfo** キーにてこの機能を使用することを宣言（値"yes"を設定）する必要があります。
- フィルタリング結果から生成したプレイリストを i アプリから保存することはできません。
- 音楽コンテンツ管理データベースから取得したコンテンツオブジェクトは以下の契機で無効化されます。
 - **Player.powerOff()**メソッド（27.3 項参照）によって、音楽プレイヤーを終了した場合
 - 外部メモリの挿抜が行われた場合やコンテンツ取得保存機能などによって音楽コンテンツ管理データベースの更新が必要になった場合

オブジェクトが無効化されると、そのオブジェクトにキャッシュされていないミュージック情報を取得することはできません。オブジェクト無効化後にキャッシュされていないミュージック情報を取得する必要がある場合は、再度 **DataManager** クラスからオブジェクトを取得する必要があります。

27.3 音楽プレイヤー制御

i アプリから音楽プレイヤー機能を利用して音楽コンテンツを再生するには、**com.docomostar.ui.avplayer.Player** クラスを使用します。**Player** クラスを使用するには、まずこのクラスの static メソッド **getInstance()** を使用して **Player** オブジェクトを取得します。**Player** オブジェクトを取得した時点では音楽プレイヤーは利用不可能状態となっています。この状態で **Player.powerOn()** メソッドを呼び出すと、音楽プレイヤーが起動し、音楽プレイヤーは停止状態になります。起動した音楽プレイヤーは **Player.powerOff()** メソッドを呼び出すと終了します。

音楽プレイヤーで音楽コンテンツを再生するには、まず前項で解説した方法でプレイリスト（**Playlist** オブジェクト）を作成します。**Player.setPlaylist()** メソッドを用いてプレイリストを設定した後に、**play()** メソッドを呼び出すことで再生が開始されます。音楽コンテンツの再生は **stop()** を呼び出すと停止し、また **pause()** メソッドを呼び出すと一時停止します。

音楽プレイヤーでは、再生開始や再生停止、一時停止のほか、以下のような操作をすることができます。

- 早送り（**fastForward()**メソッド）
- 巻き戻し（**fastRewind()**メソッド）
- 1 曲送り（**skipForward()**メソッド）
- 1 曲戻し（**skipBackword()**メソッド）
- コンテンツインデックス設定（**setContentIndex()**メソッド）
- コンテンツ内での再生位置設定（**setPosition()**メソッド）

また、音楽プレイヤーではいくつかの属性を持っており、属性値をアプリケーションプログラムから設定することで動作を制御できます。音楽プレイヤーの属性を設定するには

`Player.setAttribute()`メソッドを使用します。また、`getAttribute()`メソッドにより現在の属性値を取得することもできます。規定されている属性には以下があります。

- ボリューム (`Player.DEV_VOLUME`)
- 再生モード (`Player.DEV_PLAY_MODE`)
再生モードには通常再生、リピートモード、シャッフルモードなどがあり、`Player` クラスに `ATTR_PLAY_MODE` 定数として定義されています。

各項目に設定可能な属性値の詳細は `Player` クラスの API リファレンスを参照してください。なお、再生モードのうち、「1 曲終了モード」は音楽プレイヤー機能をサポートする携帯電話の中でもオプションな機能です。

音楽プレイヤーを終了すると、終了時点での音楽コンテンツの再生位置、設定されていたプレイリスト、音量や再生モードの属性値などの情報はレジューム情報として携帯電話システム内部に自動的に保存されます。`Player.powerOn()`メソッドによって音楽プレイヤーを起動する際に携帯電話システム内部にレジューム情報が存在する場合、レジューム情報が読み込まれて、一時停止状態になります。この状態で `play()`メソッドを呼び出すと、読み込まれたレジューム情報に従い前回の続きから再生を再開することができます。

なお、ネイティブの音楽プレイヤーアプリケーションもレジューム情報に対応しており、ネイティブの音楽プレイヤーと音楽プレイヤーはレジューム情報を共有します。そのため、音楽プレイヤー起動時に読み込まれるレジューム情報は、必ずしも直前の音楽プレイヤー終了時点の情報とは限りません。

注意事項：

- 1 曲送り、1 曲戻しは音楽プレイヤー機能をサポートする携帯電話の中でもオプションな機能です。これらの操作をサポートしない機種では `skipForward()`メソッドあるいは `skipBackward()`メソッドを呼び出すと `UnsupportedOperationException` の例外が発生します。
- 再生期限が切れた音楽コンテンツや、再生期間外の音楽コンテンツを再生することはできません。また再生回数制限が設定された音楽コンテンツは、残り再生可能回数に関わらず再生することはできません。なお、音楽コンテンツに設定された再生制限の内容はミュージック情報の 1 項目 (`PlayableInfo`) として参照することができます。
- 再生制限により再生不可の音楽コンテンツがプレイリストに含まれている場合、再生不可の音楽コンテンツの再生はスキップされ、次の音楽コンテンツの再生が開始されます。`Player.setContentIndex()`メソッドによって再生不可の音楽コンテンツをカレントコンテンツに設定して再生しようとした場合は例外が発生します。
- 音楽コンテンツ管理データベースから取得したコンテンツオブジェクトは以下の契機で無効化されます。
 - `Player.powerOff()`メソッドによって、音楽プレイヤーを終了した場合
 - 外部メモリの挿抜が行われた場合やコンテンツ取得保存機能などによって音楽コンテンツ管理データベースの更新が必要になった場合オブジェクトが無効化されると、そのオブジェクトにキャッシュされていないミュージック情報を取得することはできません。オブジェクト無効化後にキャッシュされていないミュージック情報を取得する必要がある場合は、再度 `DataManager` クラスからオブジェクトを取得する必要があります。
- 音楽プレイヤー起動中に音声着信などにより `i` アプリがサスペンドした場合や、`Camera` クラス、`CodeReader` クラスを用いてカメラ機能を起動した場合、音楽プレイヤーは一時停止状態になります。レジューム後は、サスペンド直前の音楽プレイヤーの状態が再生状態の場合にのみ自動的に再生状態になり再生が継続されます。
- 音楽プレイヤーでの音楽再生と、`AudioPresenter` クラスの `play()`メソッドや `AudioTrackPresenter` クラスの `play()`メソッド、`PhoneSystem` クラスの `playSound()`メソッド、`Synthesizer` クラスの `play()`メソッドによる音再生、および `Flash` コンテンツ再生における音再生は同時には再生されません。この場合、先に指示した再生音は停止し、後から指示した再生音が鳴動します。

27.4 音楽プレイヤーイベント

音楽プレイヤーの状態（再生状態や停止状態など）が変化した場合や、何らかの理由で再生・早送り・巻き戻しなどの処理の継続が困難なエラーが発生した場合（音楽管理データベースの更新を行う必要が生じた場合も含みます）などに、そのことをイベントとして i アプリに通知することができます。

i アプリが音楽プレイヤーからのイベントを受信するには、事前に `StarApplication.addEventListener()` メソッドに `StarEventObject.STAR_AVPLAYER_NOTIFIED` を指定してイベントリスナオブジェクトを登録しておく必要があります。音楽プレイヤーのイベントリスナを登録した状態で音楽プレイヤーイベントが発生すると、i アプリ実行環境は登録されたイベントリスナオブジェクトの `updateStarApplication()` を呼び出します。このとき、`updateStarApplication()` メソッドの引数には `PlayerEvent` オブジェクトが設定されています。

`PlayerEvent` オブジェクトには状態変化イベントやエラーイベントを表すサブ種別を取得する `getSubType()` メソッド、イベントの詳細情報を取得する `getDetailInfo()` メソッドが定義されており、これらのメソッドを呼び出すことで音楽プレイヤーから通知されたイベントの内容を知ることができます。

27.5 Bluetooth 接続された外部機器からの操作イベント

Bluetooth を搭載した携帯電話の中には、音楽プレイヤーに対する音楽の再生や停止などの操作が可能な AVRCP(Audio/Video Remote Control Profile)プロファイルに対応したものがあります。このような機種では、事前に登録を完了した Bluetooth 外部機器（典型的にはリモコン機能付き Bluetooth ヘッドフォンなど）からの操作コマンド(オペレーション)を i アプリに通知することができます。

i アプリが Bluetooth 外部機器からのオペレーションを受信するには、事前に `StarApplication.addEventListener()` メソッドに `StarEventObject.STAR_BLUETOOTH_OPERATION_RECEIVED` を指定してイベントリスナオブジェクトを登録しておく必要があります。このようにして音楽プレイヤーのイベントリスナを登録した状態で、あらかじめ AVRCP 対応機器として携帯電話に登録および接続が行われた Bluetooth 外部機器からのオペレーションを受信すると、i アプリ実行環境は登録されたイベントリスナの `updateStarApplication()` メソッドを呼び出します。このとき、`updateStarApplication()` の引数には `com.docomostar.device.BluetoothEvent` オブジェクトが設定されています。

`BluetoothEvent` オブジェクトには受信したオペレーションを取得する `getOperation()` メソッドが定義されており、i アプリでこのメソッドを呼び出すことで受信したオペレーションを取得し、音楽プレイヤーを制御することができます。現在のプロファイルでは、受信可能なオペレーションとして以下のものが `BluetoothEvent` クラスに定義されています。

<code>OPERATION_AVRCP_PLAY</code>	: AVRCP プロファイルによる play
<code>OPERATION_AVRCP_PAUSE</code>	: AVRCP プロファイルによる pause

OPERATION_AVRCP_STOP : AVRCP プロファイルによる stop
OPERATION_AVRCP_FORWARD : AVRCP プロファイルによる forward
OPERATION_AVRCP_BACKWARD : AVRCP プロファイルによる backward

注意事項：

- 音声着信などにより i アプリがサスペンドしている間に受信した **Bluetooth** 外部機器からのオペレーションは、i アプリには通知されず、i アプリ実行環境から直接音楽プレイヤーに通知されます。そのため、レジューム直後の音楽プレイヤーの状態は i アプリが意図した状態であるとは限りません。必要に応じて、**Player.getStatus()** メソッドを呼び出して音楽プレイヤーの状態を確認するようにしてください。

【Star-1.5】

音楽プレイヤー機能は Star-1.5 プロファイルで新設されました。

第 28 章

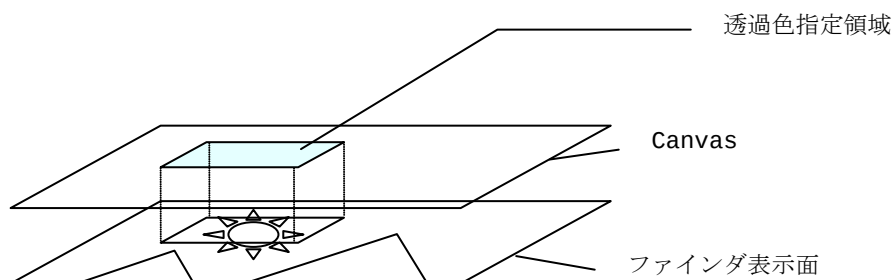
カメラファインダ機能

カメラファインダ機能は、携帯電話システムがカメラデバイスから定期的に画像を取得し、リアルタイムにディスプレイ上に表示する機能です。カメラファインダ機能を用いると、カメラデバイスを通して得られる景色の上に、付加情報（文字や画像）を重ねて表示するような i アプリを容易に作成することができます。

カメラファインダ機能は、`com.docomostar.opt.device.CameraFinder` クラスで提供されます。アプリケーションプログラムが `CameraFinder` オブジェクトを取得するには、`CameraFinder` クラスの static メソッドである `getCameraFinder()` メソッドにカメラ ID（個々のカメラデバイスに固定的に割り当てられた、0 から始まる整数）を指定して呼び出します。

カメラファインダ機能では、カメラから得られる画像を、ファインダのようにリアルタイムに **Canvas** に自動表示することができます。カメラを立ち上げて画像（ファインダデータ）を表示するには `start()` メソッドを呼び出します。このメソッドを実行すると、デバイスリソースの獲得や初期化が行われて **START** 状態（カメラデバイス起動中かつファインダ表示面（**Canvas** の背景にある、カメラファインダ機能の画像表示面）にファインダデータが表示されている状態）になります。**START** 状態時は一定間隔でファインダデータが更新されファインダ表示面に描画されます。

カメラファインダ機能における、**Canvas** とファインダ表示面の関係について説明します。カメラファインダ機能を通して得られる画像（ファインダデータ）は、**Canvas** の背面に存在するファインダ表示面に描画されます。カメラファインダ機能が **START** 状態の間は、**Canvas** は機種毎に固有に定められている特定の色を透過色として取り扱うようになります。**Canvas** 上でその透過色の描画が行われている部分は、背面のファインダ表示面の内容が透過描画されることになり、結果としてその内容がディスプレイに表示されます。



上記で触れたように透過色は機種毎に固有に定められており、その色

(`Graphics.getColorOfName()`メソッドなどで得られる、色を表す機種固有の整数値)は `CameraFinder.getTransparentColor()`メソッドで取得することができます。たとえば、アプリケーションプログラムが `CameraFinder.getTransparentColor()`メソッドが返す色で `Canvas` 全体を塗りつぶすと、`Canvas` 全面に渡りファインダ表示面の内容が表示されることになります。

なお、透過色として使用する色をアプリケーションプログラムから変更することはできません。

カメラデバイスを停止するには `stop()`メソッドを呼び出します。このメソッドを実行すると `STOP` 状態 (カメラデバイス停止中かつファインダ表示面にファインダデータが表示されていない状態) になります。

`CameraFinder` クラスは、`STOP` 状態にときに以下のメソッドを使用することで、ファインダデータの表示領域、フォーカスモードを設定することができます。

- `CameraFinder.setViewFinderArea()` : ファインダの表示領域の設定
- `CameraFinder.setFocusMode()` : フォーカスの設定

なお、`START` 状態のときにこれらのメソッドを使用しても設定は行われず無視されます。また、`CameraFinder` で設定した表示領域、フォーカスモードは、ネイティブのカメラ機能、`Camera` オブジェクト、`CodeReader` オブジェクトの設定に影響を与えません。

設定可能なファインダの表示領域サイズや端末に設定可能なフォーカスモードのリストは、`START/STOP` 状態に関わらず以下のメソッドを使用することで取得することができます。

- `CameraFinder.getAvailableViewFinderSizes()` : 表示領域サイズのリストの取得
- `CameraFinder.getAvailableFocusModes()` : フォーカスモードリストの取得

また、`START/STOP` 状態に関わらず以下のメソッドを使用することで、現在のファインダの表示領域、フォーカスモードおよびカメラファインダの表示領域サイズに対する画角を取得することができます。

- `CameraFinder.getViewFinderArea()` : ファインダの表示領域の設定状態の取得
- `CameraFinder.setFocusMode()` : フォーカスモードの設定状態の取得
- `CameraFinder.getViewAngleSize()` : 表示領域サイズに対する画角の取得

注意事項 :

- `Camera` オブジェクトまたは `CodeReader` オブジェクトと `CameraFinder` オブジェクトは互いの状態に影響を与えません。カメラファインダ機能の `START/STOP` 状態に関わらず `Camera` オブジェクトからは撮影画像を、`CodeReader` オブジェクトからはコード認識結果を取り出すことができます。
- ネイティブのカメラ機能およびコード認識機能が起動している場合に、同時にカメラファインダ機能を利用することはできません。
- 動作認識機能とカメラファインダ機能を同時に起動することはできません。
- カメラ起動音または停止音が鳴動している間、`AudioPresenter` クラスの `play()`メソッドや `AudioTrackPresenter` クラスの `play()`メソッド、`PhoneSystem` クラスの `playSound()`メソッドによる音再生は無視されます。
- `AudioPresenter` クラスの `play()`メソッドや `AudioTrackPresenter` クラスの `play()`メソッド、`PhoneSystem` クラスの `playSound()`メソッドによる音再生中にカメラを起動または停止した場合、音

再生は停止してカメラの起動音または停止音が優先して鳴動します。また、携帯電話がネイティブ機能の音再生を優先する設定となっている状態で、音楽プレーヤーやデジタルテレビなどのネイティブ機能が i アプリのバックグラウンドで音再生を行っている場合、 i アプリから `CameraFinder.start()` メソッドを実行すると例外が発生します。

【Star-1.5】

本機能は Star-1.5 プロファイルで新設されました。

第 29 章

音声合成機能

音声合成機能は、i アプリから設定したテキスト（文字列データ）を携帯電話ネイティブの音声合成エンジンを使用して音声として携帯電話のスピーカーなどから出力する機能です。設定したテキストをそのまま音声出力することに加えて、テキストの一部について属性や強弱などを変更して音声出力することや、テキストの一部をアプリケーション固有の読み方で音声出力することもできます。この機能を使用することで、i アプリにおける音声ガイダンスなどの機能を、サウンドファイルを個別に準備することなく容易に実現することができます。

音声合成機能は `com.docomostar.opt.ui.SpeechSynthesizer` クラスで提供されます。`SpeechSynthesizer` クラスは `MediaPresenter` インタフェースを実装していますが、その機能は音声合成機能に特化しており、他の `MediaPresenter` インタフェースを実装したクラスに比べていくつかの機能が制限されています。

音声合成機能を利用するには、アプリケーションプログラムはまず最初に `static` メソッド `SpeechSynthesizer.getSpeechSynthesizer()` を使用して `SpeechSynthesizer` オブジェクトを取得します。以下に音声合成機能 API の利用方法について解説します。

音声合成機能に関する属性の設定

音声合成機能では発音を制御するためのいくつかの属性を持っており、属性値をアプリケーションプログラムから設定することで動作を制御できます。属性の設定は `SpeechSynthesizer.setAttribute()` メソッドを使用して行います。現在のプロファイルで規定されている属性には以下があります。

- ボリューム (`SpeechSynthesizer.VOLUME`)
i アプリの音量設定値の何%の音量で出力するかを 0（無音）～100（最大）の範囲で指定します。初期値は 100 です。
- ピッチ (`SpeechSynthesizer.PITCH`)
音程を -100（低く）～0（標準）～100（高く）の範囲で指定します。初期値は 0 です。
- 再生速度 (`SpeechSynthesizer.SPEED`)
再生速度を -100（遅く）～0（標準）～100（早く）の範囲で指定します。初期値は 0 です。
- 動作モード (`SpeechSynthesizer.SPEECH_MODE`)
動作モードには指定したテキストをそのまま音声出力するプレーンモード (`SPEECH_MODE_PLANE`) と、テキストの一部において上述の属性や強弱などを変更して音声出力することが可能なタグモード (`SPEECH_MODE_TAG`) があります。デフォルトの動作モードはプレーンモードです。タグモードについては後述します。

なお、これらの属性は次に音声出力を開始する際に有効になります。現在出力中の音声には影響を与えません。

音声出力するテキストの設定、およびタグモード、ルビ情報の使用

音声出力するテキストは `SpeechSynthesizer.setText()` メソッドで設定します。

動作モードとしてプレーンモードを指定した場合、`setText()` メソッドで設定されたテキストが `setAttribute()` メソッドで指定した属性に従って音声出力されます。

動作モードとしてタグモードを指定した場合、記法が定められた記号やタグをテキスト中に含ませることで、その記号やタグを記述した位置で発声方法（イントネーションや再生速度など）を制御することができます。現在のプロファイルで有効な記号やタグは以下の通りです。

- 文の終端記号（句点）
句点を適切に使い分けることで、文末のイントネーションを変更させることができます。

終端記号	説明
。	通常のイントネーションで出力する終端記号です。 全角のみ指定可能です。
？	疑問のイントネーションで出力する終端記号です。 全角、半角のどちらでも指定することができます。
！	驚きのイントネーションで出力する終端記号です。 全角、半角のどちらでも指定することができます。

- タグ記号
テキスト中のタグ記号が記述された位置で、発声の属性や強弱を変更することができます。
タグ記号は半角英数字で装飾内容と合わせて”<”（ASCII コード 0x3C）と”>”（同コード 0x3E）の間に記述します。有効なタグの表記方法とその意味は以下の通りです。

タグ記号表記	説明
<emphasis=数値>	左記の表記以降のテキストを強調して発声します。 -100（弱く）～0（初期値）～+100（強く）の範囲で指定します。 数値を記載せず<emphasis>と記述すると<emphasis=0>と記述した場合と同じように扱われます。 例) <emphasis=+50>おはようございます。
<pitch=数値>	左記の表記以降のテキストのピッチを変更して発声します。 -100（低く）～0（初期値）～+100（高く）の範囲で指定します。 setAttribute()メソッドでピッチ属性（PITCH）を設定している場合、設定されている値にこのタグ記号で指定した値を加算した値が使用されます。 数値を記載せず<pitch>と記述すると<pitch=0>と記述した場合と同じように扱われます。 例) <pitch=-20>こんにちは。
<rate=数値>	左記の表記以降のテキストの再生速度を変更して発声します。 -100（遅く）～0（初期値）～+100（早く）の範囲で指定します。 setAttribute()メソッドで再生速度属性（SPEED）を設定している場合、設定されている値にこのタグ記号で指定した値を加算した値が使用されます。 数値を記載せず<rate>と記述すると<rate=0>と記述した場合と同じように扱われます。 例) <rate=+30>こんばんは。

<volume=数値>	左記の表記以降のテキストのボリュウムを変更して発声します。 0（無音）～+100（初期値）の範囲で指定します。 setAttribute()メソッドでボリュウム属性 (VOLUME) を設定している場合、設定されている値にこのタグ記号で指定した値を乗算し 100 で割った値が使用されます。 数値を記載せず<volume>と記述すると<volume=+100>と記述した場合と同じように扱われます。 例) <volume=80>おはよう。
<break=数値>	左記の表記の箇所で、数値指定された長さの分だけ無音で出力します。 数値の単位は msec で、0（なし）～+10000（長く）の範囲で指定します。分解能（設定値の刻み幅）は 100msec で、100 未満は切り捨てられます。
<clear>	左記の表記以前に記述された emphasis、pitch、rate、volume の各タグによる装飾を全てクリアします。

<emphasis>、<pitch>、<rate>、<volume>の各タグによる装飾は次に同じタグにより装飾を行うか、<clear>により装飾をクリアするまで有効になります。また、指定可能な数値の範囲外の値を指定した場合や"<"から">"の間に上記で規定されていないタグ記号を用いた場合、そのタグは有効なタグとみなされず、文章として扱われ出力されます。有効なタグとみなされない場合の詳細は API リファレンスの SpeechSynthesizer クラスの説明を参照してください。

また、音声合成機能では、あるテキストを音声出力する際に、携帯電話ネイティブの音声合成エンジンが持つ読み方ではなく、アプリケーションで指定した固有の読み方で出力することができます。そのテキストの表記とアプリケーション固有の読み方の組をルビ情報といいます。

ルビ情報は setRuby()メソッドの引数に「表記」と「読み方」の組の配列 (String の二次元配列) で指定します。

表記「今日」の読み方として「コンニチ」、「明日」の読み方として「ミョウニチ」をルビ情報として設定するには以下のようにします。

```
setRuby(new String[][] { { "今日", "コンニチ" }, { "明日", "ミョウニチ" } });
```

なお、音声合成エンジンによっては設定できないルビ情報が存在する場合があります。ルビ情報が設定可能かどうか調べるには isAvailableRuby()メソッドを使用します。isAvailableRuby()メソッドが false を返すルビ情報を setRuby()メソッドに指定することはできません。また、setRuby()メソッドで登録可能なルビ情報の数には上限があります。登録可能なルビ情報の最大数は getAvailableRubyLength()メソッドで確認できます。

音声合成機能イベントのリスナーオブジェクトの登録

次項で解説するように、音声合成機能ではアプリケーションプログラムから音声出力開始が指示された後、実際の音声出力はアプリケーションプログラムとは非同期に行われます。このため音声合成機能では、メディアリスナーを使用して音声出力の開始や完了、中断をイベントとして i アプリに通知する機構が備えられています。

これらのイベント通知を受け取るためには、アプリケーションプログラムで com.docomostar.ui.MediaListener インタフェースを実装したリスナーオブジェクトを作成し、SpeechSynthesizer.setMediaListener()メソッドを使用してシステムに登録します。音声出力イベントが発生すると、リスナーオブジェクトの mediaAction()メソッドに通知されます。

mediaAction()メソッドの引数に渡されるイベントタイプ定数は SpeechSynthesizer クラスに定義されており、以下のものが規定されています。

- SPEECH_PLAYING
音声出力が開始されたことを表します。

- **SPEECH_COMPLETE**
設定したテキストの音声出力が完了したことを表します。
- **SPEECH_STOPPED**
音声出力が停止されたことを表します。

音声出力の開始と停止

音声出力の開始は、**SpeechSynthesizer.play()**メソッドを使用して行います。このメソッドは、設定した属性やテキスト、ルビ情報に従って音声出力を開始するとすぐにアプリケーションプログラムに復帰し、以降の音声出力はアプリケーションプログラムとは非同期で行われます。その際、イベントリスナーには音声出力開始イベントが通知されます。**play()**メソッドを呼び出した時点で、それまでに**setText()**メソッドで設定したテキストと**setRuby()**メソッドで設定したルビ情報はクリアされます。

アプリケーションプログラムは、必要に応じて**SpeechSynthesizer.stop()**メソッドを呼び出すことで、音声出力を停止することができます。

設定されたテキストが最後まで出力されると、音声出力完了イベントがイベントリスナーに通知されます。音声出力中に**stop()**メソッドを呼び出したり、何からの要因で音声出力が停止した場合には、音声出力停止イベントがイベントリスナーに通知されます。

注意事項：

- **SpeechSynthesizer** クラスは **MediaPresenter** インタフェースを実装していますが、**getMediaResource()**メソッドは使用することができません。また、**setAttribute()**メソッドによって制御できる属性は、**SpeechSynthesizer** クラスの **VOLUME**、**PITCH**、**SPEED**、**SPEECH_MODE** です。これら以外の属性を指定しても属性の設定は行われません。
- この機能と、**AudioPresenter** によるサウンド再生など音の鳴動を伴う他の機能を同時に利用することはできません。この場合、先発の要求は停止され、後発の要求が実行されます。音声合成機能による音声出力が行われている状態で**SpeechSynthesizer.play()**メソッドを呼び出した場合の振る舞いは機種によって異なります。
- **setText()**メソッドに指定可能なテキストの長さは機種によって異なります。指定可能なテキストの長さは**getAvailableBytes()**メソッドで取得することができます。なお、バイト数は携帯電話ネイティブの音声合成エンジンが対応する文字エンコーディングで評価されます。携帯電話ネイティブの音声合成エンジンが対応する文字エンコーディングは**getEncode()**メソッドで取得することができます。
- 携帯電話に搭載される音声合成エンジンによっては、正しく音声出力できない文字が存在する場合があります。そのような文字を含むテキストを**setText()**メソッドの引数に指定した場合、指定したテキストがどのように音声出力されるかは機種によって異なります。

【Star-1.5】（2011 年春以降発売モデル）

本機能は Star-1.5 プロファイルで新設されました。

第 30 章

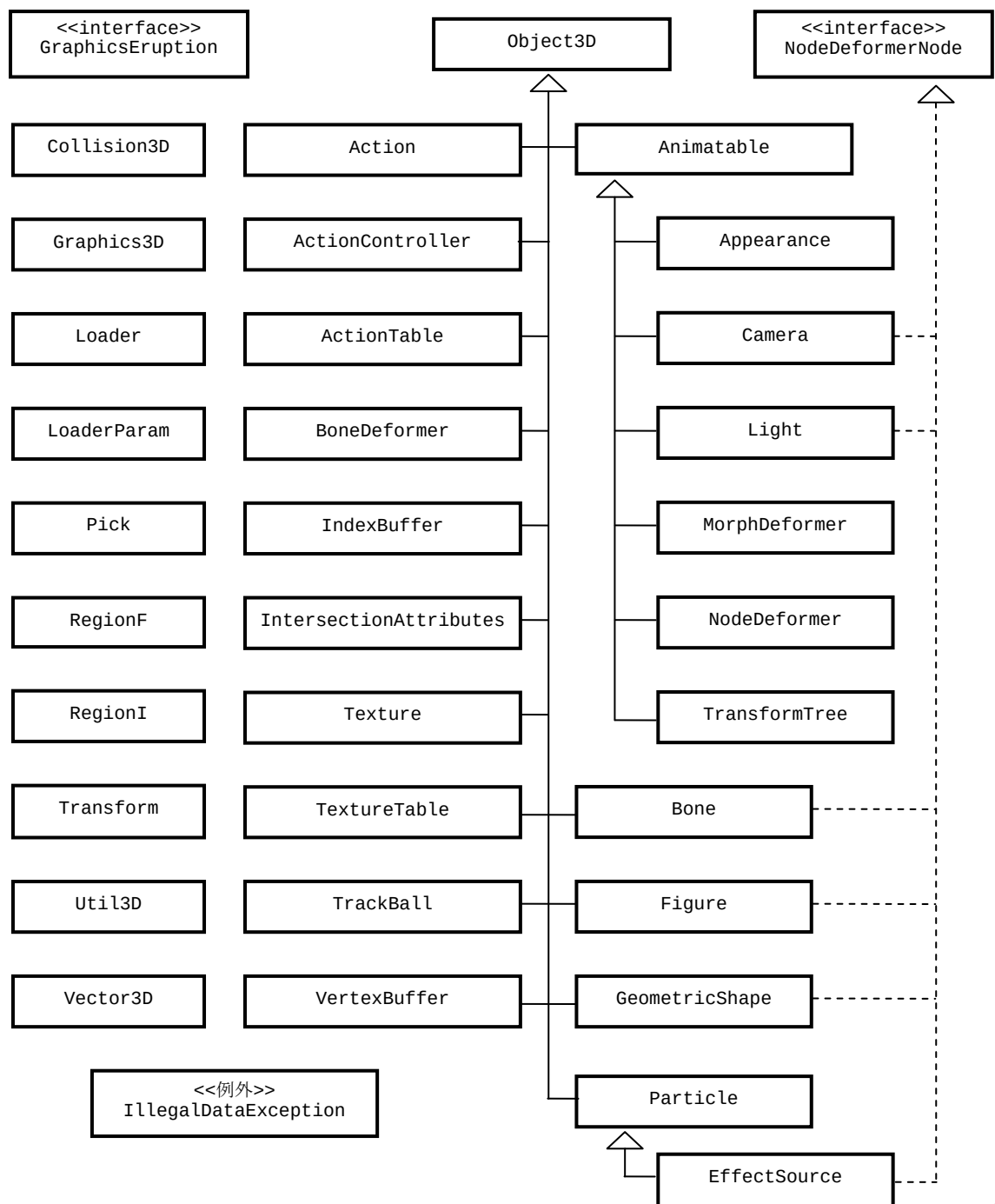
MascotCapsule eruptionST API による 3D グラフィックス描画

Star-1.5 プロファイルに対応する携帯電話の中でも 2011 年春以降に発売する携帯電話には、3D グラフィックスエンジンとして株式会社エイチアイの MascotCapsule eruptionST を搭載したものがああります。このような機種のため、MascotCapsule eruptionST を利用するための API が追加されました。本 API を利用することで、MascotCapsule eruptionST を利用した他プラットフォーム向けのアプリケーションを、従来より容易に i アプリに移植することができます。

MascotCapsule eruptionST を活用した i アプリプログラミングの詳細については、株式会社エイチアイより提供されるドキュメントを参照してください。

MascotCapsule eruptionST を利用する API は、`com.docomostar.opt.ui.eruption` パッケージで提供されます。MascotCapsule eruptionST API のクラス構成を以下に示します。

com.docomostar.opt.ui.eruption パッケージ



各クラス、インタフェースの機能概要は以下の通りです。

クラス・インタフェース	機能概要
GraphicsEruption	MascotCapsule eruptionST API をサポートする機種において、Graphics クラスが実装しているインタフェースです。MascotCapsule eruptionST API による 3D 描画機能の使用開始・終了を宣言するメソッドが定義されています。
NodeDeformerNode	シーンノード機能を提供するインタフェースです。Bone、Camera、EffectSource、Figure、GeometricShape、Light の各クラスがこのインタフェースを実装しています。
Action	3D モデルのアクション（アニメーションデータ）を表現するクラスです。Animatable クラスを継承するクラスのオブジェクトのアニメーション情報を保持しています。
ActionController	Action オブジェクトに対して、リピートモードやフレーム値、モーシヨンプレンドの設定を行う際に使用するクラスです。
ActionTable	3D モデルのアクション（アニメーションデータ）を管理するクラスです。
Animatable	アニメーションを行うクラスを表す抽象クラスです。
Appearance	サブメッシュに対するレンダリング属性を扱うクラスです。
Bone	シーングラフ内でアンカーの役割を果たすクラスです。
BoneDeformer	ボーンアニメーションを扱うクラスです。
Camera	描画の投影方法やフォグ効果を与えるためのデータを保持するクラスです。
Collision3D	球体の衝突判定や可視判定を行うクラスです。
EffectSource	エフェクトの発生点を表すクラスです。このクラスのオブジェクトに関連付けられた Particle を使用してエフェクトの設定を行います。
Figure	シーンの構成要素を集約するクラスです。要素として VertexBuffer、IndexBuffer、NodeDeformer、BoneDeformer、MorphDeformer、TransformTree、Camera、GeometricShape、Appearance を持つことができます。
GeometricShape	球、ボックス、円柱、カプセルの幾何形状を表すクラスです。
Graphics3D	MascotCapsule eruptionST API をサポートする機種において、3D 描画に使用するグラフィックスコンテキストを表すクラスです。 ステレオ表示に対応した描画を行うためのカメラパラメータを設定するメソッドも提供します。
IndexBuffer	インデックス情報を保持するクラスです。VertexBuffer と組み合わせて Figure 内のサブメッシュを構成します。
IntersectionAttributes	Figure、VertexBuffer に対してピッキングを行うための API とピッキングの結果を取得するための API を提供するクラスです。
Light	光源データを保持するクラスです。環境光源、平行光源、点光源、スポットライト光源をサポートします。
Loader	3D モデルデータをロードするためのクラスです。
LoaderParam	3D モデルデータをロードする際、デフォルトとは異なる設定値でロードする場合に使用するクラスです。
MorphDeformer	モーフィングアニメーションを行うクラスです。
NodeDeformer	NodeDeformerNode 実装オブジェクトを保持し、それらのノードアニメーション実行単位となるクラスです。NodeDeformerNode 実装オブジェクトと、Figure に関連付けられている TransformTree を関連付けます。
Object3D	3D オブジェクトの基底となるクラスです。
Particle	パーティクルを表すクラスです。
Pick	ピック処理を行うクラスです。
RegionF	矩形領域を表すクラスです。領域を浮動小数点数で指定します。

RegionI	矩形領域を表すクラスです。領域を整数で指定します。
Texture	テクスチャのデータを保持するクラスです。
TextureTable	3D モデルのテクスチャを管理するクラスです。
TrackBall	オブジェクトを覆う仮想球体をドラッグすることでオブジェクトの座標変換を行うトラックボール処理を提供するクラスです。移動と回転処理をサポートします。
Transform	4x4 の変換行列を表すクラスです。
TransformTree	Figure と対になり、シーングラフを構成します。
Util3D	基本的な浮動小数点数演算を行うためのユーティリティクラスです。MascotCapsule eruptionST API を使用する際には、数値演算（三角関数、平方根演算）はこのクラスを使用して行います。
Vector3D	3次元ベクトルを表すクラスです。内積・外積計算、正規化といったユーティリティメソッドを備えています。
VertexBuffer	頂点情報を保持するクラスです。IndexBuffer と組み合わせて Figure 内のサブメッシュを構成します。
IllegalDataException	データフォーマットに異常があることを表す例外クラスです。

注意事項：

- MascotCapsule eruptionST API による 3D グラフィックス描画と通常の (Graphics クラスで定義されている) 2D グラフィックス描画は明示的に分離して行う必要があります。MascotCapsule eruptionST API による 3D グラフィックス描画は、GraphicsEruption.beginEruption() メソッドを呼び出してから GraphicsEruption.endEruption() メソッドを呼び出すまでの区間で行います。この区間では、通常の 2D グラフィックス描画は行わないようにしてください。
- MascotCapsule eruptionST API による 3D グラフィックス描画と、i アプリ基本 API の 3D グラフィックス描画機能 (com.docomostar.ui.graphics3d パッケージ) あるいは i アプリ拡張 API の高レベル 3D グラフィックス描画機能 (com.docomostar.opt.ui.j3d パッケージ) を併用することはできません。これらを混在して利用するとシステム内部で相互に干渉が発生し、予期しない結果を引き起こす場合があります。
- MascotCapsule eruptionST API による 3D グラフィックス描画と Open GL ES API による 3D グラフィックス描画を同時に使用することはできません。
また、Open GL ES API による 3D グラフィックス描画を行った後に MascotCapsule eruptionST API による 3D グラフィックス描画をする場合、com.docomostar.opt.ui.eruption パッケージの Graphics3D.syncContext() メソッドを呼び出して OpenGL ES の状態を初期化する必要があります。

【Star-1.5】 (2011 年春以降発売モデル)

本機能は Star-1.5 プロファイルで新設されました。

第 31 章

ステレオ表示機能

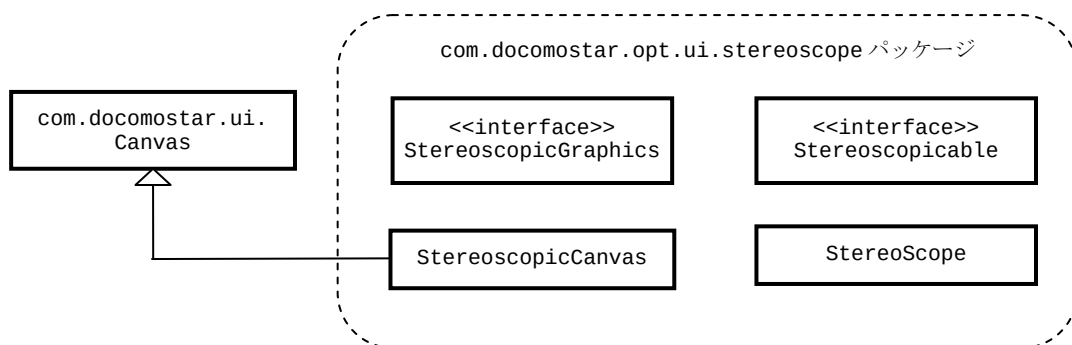
Star-1.5 プロファイルに対応する携帯電話の中でも 2011 年春以降に発売される機種には、立体視表示可能なディスプレイを搭載するものがあります。ステレオ表示機能とは、立体視表示可能なディスプレイ搭載機種において、i アプリの描画内容を立体映像として表示する機能です。ステレオ表示機能の API を利用することで、描画内容を立体映像として表示し、臨場感を得やすい迫力のあるユーザーインターフェースを持つ i アプリを作成することができます。

注意事項：

- 不適切な設定で立体視表示を行うと、ユーザーに違和感を生じさせ、めまいや酔いの症状を生じさせる原因になる場合があります。アプリケーションプログラマは、不適切な設定での立体視表示は避け、ユーザーの安全性を考慮したコンテンツを開発するようにしてください。
参考ドキュメント：3DC 安全ガイドライン (http://www.3dc.gr.jp/jp/scmt_wg_rep/guide_index.html)

31.1 ステレオ表示機能 API の構成

ステレオ表示機能は、`com.docomostar.opt.ui.stereoscope` パッケージで提供されます。以下に、ステレオ表示機能 API のクラス構成を示します。



各クラス、インタフェースの機能概要は以下の通りです。

クラス・インタフェース	機能概要
Stereoscopicable	ステレオ表示に対応する Canvas を識別するためのマーカーインタフェースです。このインタフェースを実装した Canvas は、ステレオ表示が可能な Canvas として i アプリ実行環境に認識されます。 このインタフェースを実装した Canvas がカレントフレームの場合はステレオ表示の状態を有効化したり無効化したりすることができます。
StereoscopicGraphics	ステレオ表示をサポートする機種において、 StereoscopicCanvas に関連付けられる Graphics オブジェクトが実装しているインタフェースです。このインタフェースを実装した Graphics オブジェクトを使用することで、2D グラフィックス描画あるいは 3D グラフィックス描画をステレオ表示することができます。
StereoscopicCanvas	ステレオ表示用のキャンバスを表す抽象クラスです。このクラスは Canvas クラスのサブクラスで、このクラスのオブジェクトがカレントフレームの場合はステレオ表示が常に有効な状態となり、描画内容がステレオ表示されます。 このクラスから取得できる Graphics オブジェクトは StereoscopicGraphics インタフェースを実装しています。
StereoScope	ステレオ表示を制御するためのクラスです。ステレオ表示の有効無効状態の切り替えや視差オフセット、解像度比を取得するなどのメソッドを提供します。

ステレオ表示機能を用いるには、表示面として、**StereoscopicCanvas**、もしくは **Stereoscopicable** インタフェースを実装した **Canvas** サブクラスを使用します。詳細は 31.3 項を参照してください。

31.2 i アプリにおけるステレオ表示機能の概要

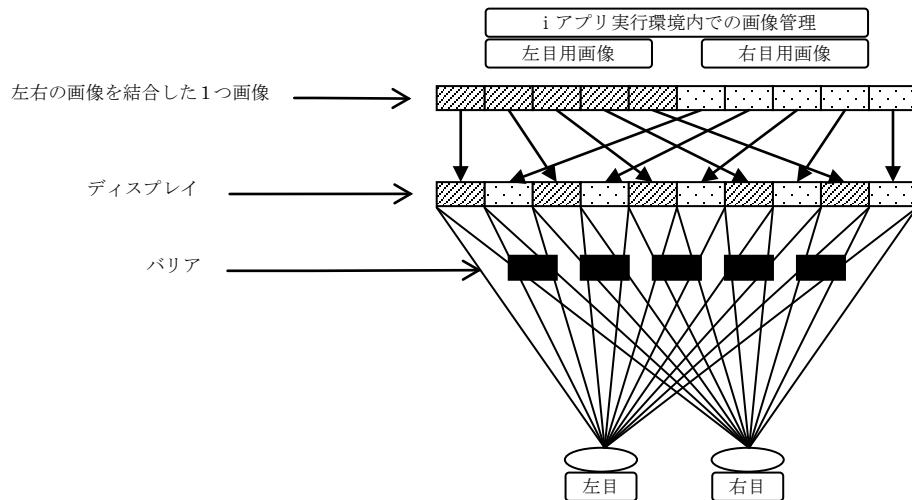
i アプリのステレオ表示機能では、「視差バリア方式」を利用したステレオ表示をサポートしています。視差バリア方式とは、右目用の画像と左目用の画像が水平方向に交互に描画された物理ディスプレイを「視差バリア」と呼ばれる遮蔽物を通して、右目は右目用画像のみを、左目は左目用画像のみを見せることによって、裸眼のままで描画内容を立体映像として視認させる方式です。

視差バリア方式を利用した立体視表示では、左目用の画像と右目用の画像を必要とします。i アプリ実行環境では、これらの画像から内部的に 1 枚の画像を生成し、ディスプレイへの表示の際に左右の画像を交互にディスプレイに表示します。

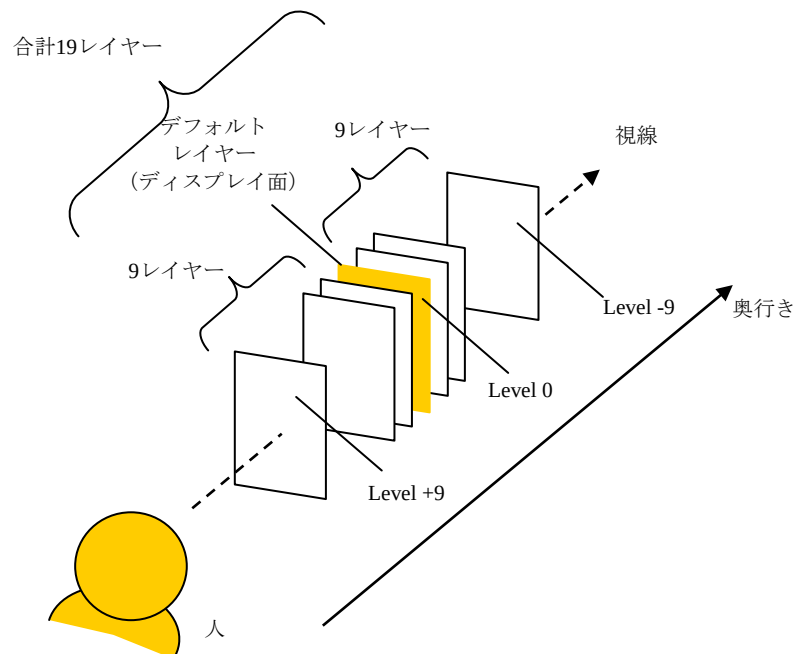
視差バリア方式を使用したステレオ表示では、左目用の画像と右目用の画像を 1 つのディスプレイ上に表示します。そのため、端末が採用している方式によっては、アプリケーション側が用意する右目用画像、左目用画像それぞれの横方向の解像度を、実際の表示領域より小さくする（横幅を狭くする）必要がある場合があります。通常表示時のサイズとステレオ表示時のサイズの比を解像度比といい、解像度比は **StereoScope.getResolutionRate()** メソッドで取得することができます。ステレオ表示する際の左右それぞれの画像の描画に使用できる水平方向の描画領域サイズは、通常表示時の水平方向の描画領域サイズを解像度比で除算することで求めることができます。現在、携帯電話向けのステレオ表示機能では、解像度比が 2 の方式が主流となっています。

アプリケーションプログラマは、解像度比に応じたサイズで左右それぞれの画像を描画し、i アプリ実行環境が、左右それぞれの画像を結合してステレオ表示用の 1 つの画像（サイドバイサイド形式）を生成します。サイドバイサイドに配置されたそれぞれの画像は、システムによ

ってディスプレイ上に交互に並べられ、それを視差バリアを通して見ることで立体映像となります。

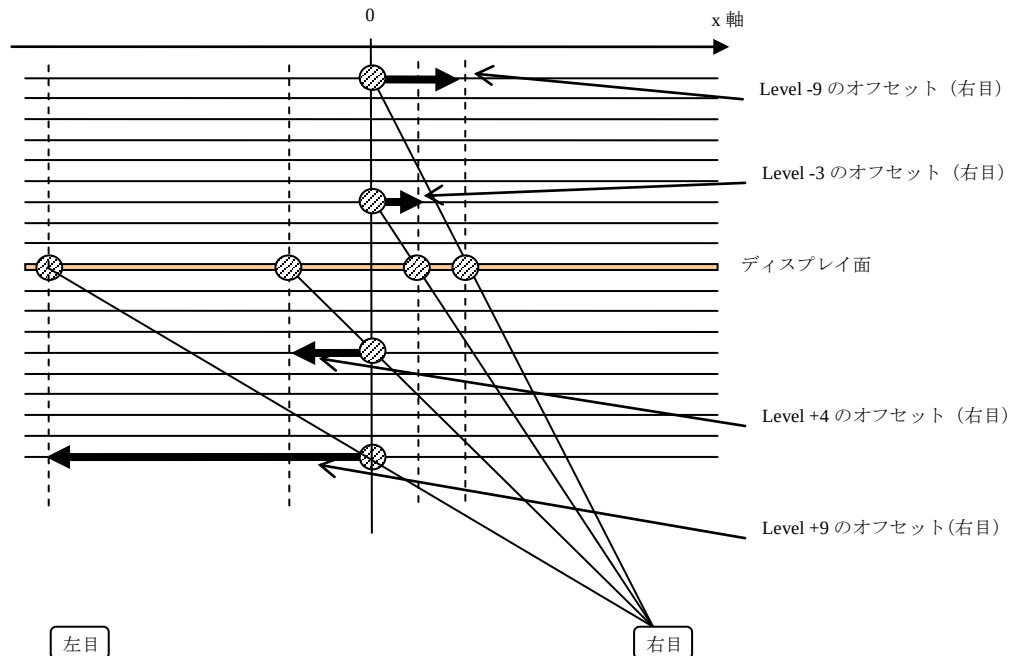


ステレオ表示機能を利用した描画を行うには、ディスプレイに表示された際に立体的に見えるように奥行きを考慮して、左右それぞれの画像を用意する必要があります。i アプリ実行環境では、奥行き表現を平易にするため、視線方向に向かって垂直に並んだ論理的な仮想平面（レイヤー）を定義し、各レイヤーの位置関係に従って描画内容を重ね合わせることで、奥行きを考慮した画像を生成できるようになっています。仮想平面（レイヤー）は、ディスプレイ面を基準にして視線方向に対して手前方向に9つ、奥方向に9つの合計19段階設けられています。アプリケーションプログラマは、レイヤーを考慮しながら描画を行います。



奥行きを考慮する場合、レイヤーが奥になるに従い、描画内容を見る左右それぞれの視線の方向は外側に広がります。この広がり具合を視差オフセットといいます。ディスプレイ面（デフォルトレイヤー）での視差オフセットを0とし、その他のレイヤーにおける視差オフセットを

取得するには、`StereoScope.getParallaxOffset()`メソッドを使用します。このメソッドの引数に、どのレイヤーの視差オフセットを取得するかを奥行きを表すレベル値で指定します。それぞれのレイヤーの描画を行う際、取得した視差オフセット値を元に描画位置を決定することで、自然な奥行き表現を得ることができます。



ステレオ表示機能 API では、i アプリ実行中に立体視強度を変更することができます。立体視強度を変更するには、`StereoScope.setStereoscopicIntensity()`メソッドを使用します。このメソッドの引数には、0 から `StereoScope.getMaxIntensityLevel()`メソッドで取得できる立体視強度の段階の最大値の範囲内の値を指定します。現在の立体視強度の設定を取得するには、`StereoScope.getStereoscopicIntensity()`メソッドを使用します。

注意事項：

- 携帯電話ユーザーは、携帯電話の設定により立体視強度を変更することができます。i アプリ起動直後の立体視強度の設定値は、携帯電話で設定された立体視強度です。
`StereoScope.setStereoscopicIntensity()`メソッドの引数に指定する立体視強度の0は携帯電話の設定の「強度なし」に、`getMaxIntensityLevel()`で取得できる値は携帯電話の設定の「標準」に相当します。
`setStereoscopicIntensity()`メソッドで設定した立体視強度は、そのi アプリが実行中の間のみ有効で、また携帯電話で設定されている立体視強度に影響を与えません。
- `StereoScope.setStereoscopicIntensity()`メソッドで立体視強度を変更すると、`StereoScope.getParallaxOffset()`メソッドで取得できる視差オフセット値も変わります。そのため、描画の途中で立体視強度を変更した場合には、視差オフセットを再度取得し、視差オフセットに適した画像を再描画する必要があります。

31.3 ステレオ表示用キャンバス

ステレオ表示機能 API では、ステレオ表示可能なフレームとして以下の 2 種類を規定しています。

- **StereoscopicCanvas**
- **Stereoscopicable** インタフェースを実装した **Canvas**

StereoscopicCanvas

StereoscopicCanvas は **Canvas** クラスのサブクラスです。アプリケーションプログラマは、**StereoscopicCanvas** のサブクラスを定義し、**paint()** メソッドをオーバーライドしてステレオ表示に対応した描画を行うことができます。

このクラスは、通常表示時の描画領域サイズ（ADF の **DrawArea** キーで指定する描画領域サイズの横幅）で描画された画像を、携帯電話の解像度比に応じたサイズに自動的に変換する機能を提供します。この機能を利用すると、アプリケーションプログラマは、解像度比を考慮した画像サイズを意識する必要がなく、**DrawArea** キーで指定したサイズで画面を描画することができます。i アプリ実行環境によって解像度比に応じた画像サイズの自動変換を行うには、**boolean** を引数に持つコンストラクタに **true** を指定して **StereoscopicCanvas** オブジェクトを生成します。**false** を指定した場合、i アプリ実行環境で自動変換は行われないため、アプリケーションプログラマが携帯電話の解像度比に応じたサイズで描画を行う必要があります。

StereoscopicCanvas クラスに関連付けられている **Graphics** オブジェクト (**init()**、**paint()** の各メソッドの引数で渡される **Graphics** オブジェクトおよび **getGraphics()** メソッドで取得する **Graphics** オブジェクト) は、**StereoscopicGraphics** インタフェースを実装しています。このインタフェースは、ステレオ表示用の描画を行うメソッドが定義されています。アプリケーションプログラムでは、このインタフェースのメソッドを利用してステレオ表示に対応した 2D グラフィックス描画あるいは 3D グラフィックス描画を行います。

また、**StereoscopicGraphics** インタフェースを実装した **Graphics** オブジェクト（以降、**StereoscopicGraphics** オブジェクト）は、左目用画像を描画する **Graphics** オブジェクトと右目用画像を描画する **Graphics** オブジェクトを内包しています。アプリケーションプログラマは、**StereoscopicGraphics** オブジェクトに対して描画メソッドを呼び出して描画することも、左右それぞれの **Graphics** オブジェクトに対して直接描画メソッドを呼び出して描画することもできます。

StereoscopicCanvas オブジェクトがカレントフレームに設定されている状態では、常にステレオ表示が有効な状態となり、左目用画像と右目用画像を利用した両眼視差によるステレオ表示が行われます（**StereoscopicCanvas** オブジェクトをカレントフレームに設定した状態でステレオ表示を無効化することはできません）。

Stereoscopicable インタフェースを実装した Canvas

Stereoscopicable インタフェースは、ステレオ表示に対応する **Canvas** であることを i アプリ実行環境に識別させるためのマーカーインタフェースです。このインタフェースが実装された **Canvas** サブクラスのオブジェクトがカレントフレームに設定されている場合、ステレオ表示の有効無効状態を切り替えたり、ステレオ表示スタイル（視差バリアの方向）を **Canvas** に対して縦方向とするか横方向にするかを設定することができます。

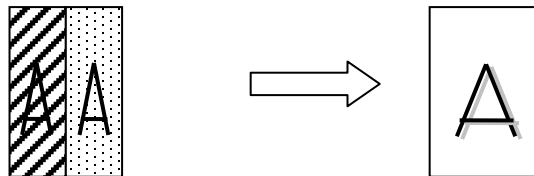
ステレオ表示の有効無効状態を切り替えるには、**StereoScope.setEnabled()** メソッドを使用します。**StereoScope.isEnabled()** メソッドにより、現在ステレオ表示が有効であるか否かを確認することもできます。

ステレオ表示スタイルの切り替えは **StereoScope.setStereoscopicStyle()** メソッドを使用して行います。このメソッドの引数に **StereoScope.STEREOGRAPHIC_STYLE_VERTICAL** を指定すると **Canvas** に対して縦方向に、**StereoScope.STEREOGRAPHIC_STYLE_HORIZONTAL** を指定すると **Canvas** に対して横方向に視差バリアが設けられます。なお、設定可能なステレオ表示スタイルは機種によって異なります。携帯電話がどの方向のステレオ表示スタイルをサポートしているかは、**StereoScope.getAvailableStyle()** メソッドにより調べることができます。また、

`StereoScope.getStereoscopicStyle()` メソッドにより、現在のステレオ表示スタイルの設定を確認することもできます。

`setEnabled()` メソッドによるステレオ表示状態の設定や、`setStereoscopicStyle()` メソッドによるステレオ表示スタイルの設定は、`StereoscopicCanvas` インタフェースを実装した `Canvas` に対してのみ有効で、`StereoScopicCanvas` オブジェクトでのステレオ表示には影響を与えません。

`StereoscopicCanvas` インタフェースを実装した `Canvas` はあくまでもマーカーインタフェースが実装されただけの通常の `Canvas` であり、`StereoscopicCanvas` クラスのように解像度比に応じて描画領域サイズを自動変換する機能はありません。`StereoscopicCanvas` インタフェースを実装した `Canvas` を使用する場合、アプリケーションプログラマは、`StereoScope` クラスの `getResolutionRate()` メソッドで取得できる解像度比に応じたサイズの左右それぞれの画像が描画された 1 つの画像を描画する必要があります。ステレオ表示が有効な状態では、その画像がシステム内部で加工され、視差バリアを通じて立体映像として表示されます。



アプリケーションプログラムでの
描画内容

立体映像

注意事項：

- `StereoscopicCanvas` クラスの画像の自動変換を利用した場合、システムが機械的に縦方向の画素の間引きを行うなどの理由により、アプリケーションプログラマが意図しない画質劣化や描画速度の低下が発生する場合があります。
- 解像度比が 2 である携帯電話では、ステレオ表示に使用するキャンバスにあわせて、ADF の `DrawArea` キーの値には以下を設定するようにしてください。
 - `StereoscopicCanvas` を使用する場合、`DrawArea` キーで指定する横幅には偶数の値を指定します。
 - `StereoscopicCanvas` インタフェースを実装した `Canvas` を縦方向のステレオ表示スタイルで使用する場合、`DrawArea` キーで指定する横幅には偶数の値を指定します。
 - `StereoscopicCanvas` インタフェースを実装した `Canvas` を横方向のステレオ表示スタイルで使用する場合、`DrawArea` キーで指定する縦幅には偶数の値を指定します。

視差バリアに対して横方向の描画サイズを奇数に設定した場合、携帯電話によっては正しく立体視表示が行えない場合があります。

- `StereoscopicCanvas` オブジェクト、あるいは `StereoscopicCanvas` インタフェースを実装した `Canvas` 以外のオブジェクトがカレントフレームに設定されている場合はステレオ表示されません。また、ステレオ表示されている状態であっても、i アプリがサスペンドした場合や、i アプリから `Dialog.show()` メソッドを呼び出した場合にはステレオ表示は無効な状態になります。
- ステレオ表示が有効な状態で同時に以下の機能を利用しないようにしてください。ステレオ表示機能と同時にこれらの機能を利用した場合の振る舞いは機種によって異なります。
 - タッチパネルデバイス機能
 - ポインティングデバイス機能
 - カメラファインダ機能
- `StereoscopicCanvas` インタフェースは、`StereoscopicCanvas` クラスと `Canvas2` クラス以外の、`Canvas` クラスのサブクラスで実装されている場合のみ、ステレオ表示対応と認識されます。`Panel` クラスなどに `StereoscopicCanvas` インタフェースを実装してもステレオ表示することはできません。

31.4 ステレオ表示用グラフィックスの描画

ステレオ表示に対応したグラフィックスを描画する方法として、ステレオ表示機能 API では以下の 2 つの方式を提供しています。

- 簡単描画方式
左右それぞれの画像を意識せずに従来の方法で描画する方式です。
この方式を利用すると、最小限の描画処理部分の変更や追加で、既存のアプリケーションをステレオ表示に対応させることができます。
簡単描画方式は、フレームとして **StereoscopicCanvas** を使う場合のみ利用可能な描画方式です。
- 左右描画方式
アプリケーションプログラマが左右それぞれの画像を描画する方式です。
この方式を利用すると、アプリケーションプログラムでステレオ表示を細かく制御できるため、動きのある物体などの立体的な動作を緻密に表現することができます。
左右描画方式は、フレームとして **StereoscopicCanvas** を使う場合にも、**StereoscopicCanvas** インタフェースを実装した **Canvas** を使う場合にも利用可能な描画方式です。

なお、**StereoscopicCanvas** を使う場合においては、これらの方式は混在させて使用することができます。たとえば、ゲームアプリにおいて動きのあるキャラクターの 3D 描画は緻密な動作を表現できる左右描画方式で、メッセージウィンドウは 2D で簡単描画方式で描画する、などといった利用方法が可能です。

簡単描画方式を利用した描画

簡単描画方式は、立体視用の設定としては奥行き（31.2 項に図示したレイヤーセットのうちどのレイヤーの位置に描画するか）のみを指定し、あとは通常の描画と同様に表示したいものをそのままのサイズで描画する方式です。簡単描画方式では左目用・右目用といった画像の区別や解像度比といった概念を意識する必要のない描画手段を提供します。簡単描画方式では、2D グラフィックス描画および 3D グラフィックス描画をステレオ表示することができます。

簡単描画方式では **StereoscopicCanvas**、および **StereoscopicCanvas** オブジェクトから取り出した **StereoscopicGraphics** オブジェクトを使用します。2D グラフィックス描画をするには、まず最初に **StereoscopicGraphics.setDepth()** を使用して奥行き（レイヤー）を指定し、その後 **StereoscopicGraphics** オブジェクトが持つ（**Graphics** クラスとしての）描画メソッドを使用して、従来通りの手法で描画を行います。3D グラフィックス描画では、利用する 3D グラフィックス描画エンジンによって奥行きの指定の方法が異なります。それぞれの描画の際に、ステレオ表示に対応した描画する方法の詳細は API リファレンスを参照してください。

i アプリ実行環境は、アプリケーションプログラマが指定した奥行き情報と描画内容を用いて自動的に左目用画像・右目用画像を生成し、**StereoscopicGraphics** オブジェクトが内包する左目用、右目用それぞれの **Graphics** オブジェクトにそれぞれ描画を行います。

左右描画方式を利用した描画

左右描画方式は、アプリケーションプログラマが奥行きに対する焦点を考慮しながら、左右それぞれの画像を描画する方式です。この方式では、**StereoscopicCanvas** オブジェクトから得られる

StereoscopicGraphics オブジェクトに対して描画を行うのではなく、**StereoscopicGraphics** オブジェクトが内包する左目用、右目用それぞれの **Graphics** オブジェクトに対して描画を行います。

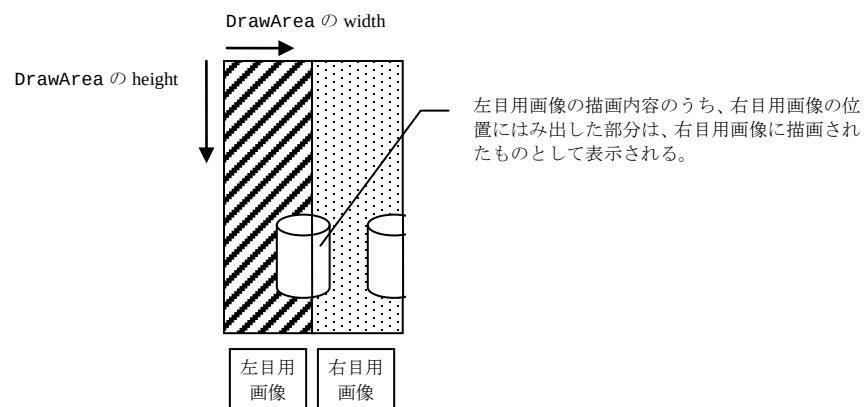
StereoscopicCanvas を用いて左右描画方式で描画を行うには、**StereoscopicGraphics** オブジェクトが内包する左目用、右目用それぞれの **Graphics** オブジェクトに直接描画を行います。

StereoscopicGraphics オブジェクトが内包する **Graphics** オブジェクトを取得するには、**StereoscopicGraphics.getRawGraphics()** メソッドを使用します。このメソッドの引数には、左目用、右目用いずれの **Graphics** オブジェクトを取得するかを指定します。このようにして取得したそれぞれの **Graphics** オブジェクトに対して、アプリケーションプログラムでは、描画する奥行きに対する視差オフセットを考慮して描画位置や視点位置を計算し、2D グラフィックス描画あるいは 3D グラフィックス描画を行います。描画方法の詳細は API リファレンスを参照してください。

Stereoscopicable インタフェースを実装した **Canvas** クラスを使用してステレオ表示を行う場合も左右描画方式を用います。しかし、**Stereoscopicable** インタフェースを実装した **Canvas** オブジェクトから取得できる **Graphics** オブジェクトは、**StereoscopicGraphics** インタフェースを実装しておらず、左目用や右目用の描画を行う専用の **Graphics** オブジェクトも提供されません。そのためアプリケーションプログラムは、**Stereoscopicable** インタフェースを実装した **Canvas** にステレオ表示を行う場合は、**Canvas** から取得できる **Graphics** オブジェクトに対して、左右それぞれの画像をサイドバイサイド形式で描画する必要があります。このとき左右それぞれの画像の描画領域サイズは、**StereoScope.getResolutionRate()** で取得する携帯電話の解像度比を使用して計算します。

Stereoscopicable インタフェースを実装した **Canvas** オブジェクトから取得できる **Graphics** オブジェクトに対してサイドバイサイド形式で描画する場合、一方の画像の描画内容を他方の画像の描画位置にはみ出して描画すると、そのはみ出した画像は他方の画像の一部として利用されます

(**StereoscopicCanvas** では、左右の画像が別々に管理されるため、はみ出すことはありません)。



また、サイドバイサイド形式で描画する場合の、左目用画像と右目用画像の位置関係は、携帯電話の描画方向やステレオ表示スタイル、解像度比によって異なります。解像度比 2 の場合と 1 の場合の、左目用画像と右目用画像の位置関係を以下に示します。

	表示スタイル	解像度比 2 の場合	解像度比 1 の場合
縦ワイドアプリの場合 例) DrawArea=480x854	STEREOSCOPIC_STYLE_VERTICAL		
	STEREOSCOPIC_STYLE_HORIZONTAL		
横ワイドアプリの場合 例) DrawArea=854x480	STEREOSCOPIC_STYLE_HORIZONTAL		
	STEREOSCOPIC_STYLE_VERTICAL		

○ 描画原点

 左目用画像
 右目用画像

 DrawArea の width
 DrawArea の height

注意事項：

- 簡単描画方式で以下のパッケージを利用した 3D グラフィックス描画を行う場合、i アプリ実行環境内部では、左右それぞれの画像を描画するために各 3D 描画機能が 2 回呼び出されます。そのため、通常よりも描画速度が低下する可能性があります。
 - com.docomostar.ui.opengl パッケージ
 - com.docomostar.ui.graphics3d パッケージ
 - com.docomostar.opt.ui.j3d パッケージ
- 左右描画方式で StereoscopicCanvas を使用してステレオ表示を行う場合、com.docomostar.opt.ui.eruption パッケージを使用することはできません。左右描画方式で Stereoscopicable インタフェースを実装した Canvas を使用してステレオ表示を行う場合、com.docomostar.opt.ui.eruption パッケージを使用することはできませんが、このパッケージの立体視描画機能を利用することはできません。
- StereoscopicCanvas への描画において、以下のパッケージを利用した 3D グラフィックス描画と 2D グラフィックス描画を併用することはできません。
 - com.docomostar.ui.opengl パッケージ
 - com.docomostar.opt.ui.eruption パッケージ
- Stereoscopicable インタフェースを実装した Canvas がカレントフレームに設定されている場合、サイドバイサイド形式の画像はステレオ表示が有効な状態では立体映像として表示されますが、その状態で StereoScope.setEnabled() メソッドでステレオ表示状態を無効化すると、サイドバイサイド形式の画像がそのままディスプレイに表示されます。ステレオ表示状態を切り替える際には、切り替え後のステレオ表示状態で適切な表示内容となるように画像を再描画する必要があります。

【Star-1.5】（2011 年春以降発売モデル）

本機能は Star-1.5 プロファイルで新設されました。

付録 A

ミニアプリにおける機能制限

ミニアプリ実行環境は複数のミニアプリを同時に実行することができますが、通常の i アプリに比べ利用可能な機能に制限が設けられています。ここでは、ミニアプリ実行環境が i アプリに与える影響のうち、i アプリオプション API および i アプリ拡張 API に関するものについて解説します。ミニアプリ実行環境やミニアプリの詳細は、「i アプリコンテンツ開発ガイド 詳細編」を参照してください。

なお、本項で解説する制限によって、全機能がミニアプリ実行環境では使用出来なくなるクラスやインタフェースについては、クラス、インタフェースそのものがミニアプリ実行環境から削除されています。ミニアプリ実行環境においてクラス定義が存在しないクラスについては API リファレンスのトップページからリンクされている注意事項を参照してください。

A.1 ミニアプリで制限される機能

以下のオプション機能、拡張機能は、ミニアプリ実行中はミニアプリの表示状態に関わらず、利用することができません。ミニアプリからこれらの機能を利用しようとした場合、例外が発生します。

- 2D グラフィックス描画
ズーム座標系機能、ラスタオペレーション機能、アニメーション GIF 内の特定フレームのイメージ描画、数値の描画、グラフィックスコンテキストからのイメージ生成、2 色の間値の取得、アフィン変換付きイメージ描画、表示デバイスの垂直同期間隔の取得、表示デバイスの垂直同期タイミングでのダブルバッファリング終了を利用することはできません。
- 3D グラフィックス描画
高レベル 3D グラフィックス描画機能を利用することはできません。
- サウンド再生
i モーションのオーディオトラック再生機能を利用することはできません。
- プラットフォームリソース制御
PhoneSystem2 クラスで定義されている、アロケート可能な連続メモリ量の参照機能、オーディオ出力のサラウンド制御機能、ディスプレイスタイル状態の参照機能、方位デバイスの基準方向の制御機能を利用することはできません。
- 携帯電話ネイティブのデータストレージアクセス
インフォメーションディスプレイ表示パターン保存領域へのアクセス機能を利用することはできません。
- ポインティングデバイス
ポインティングデバイス機能を利用することはできません。

- 指紋センサー
指紋認証機能を利用することはできません。
- 映像データ管理機能の呼び出し
映像データの選択読み込み機能、映像データの ID 指定読み込み機能を利用することはできません。
- トルカ管理機能の呼び出し
トルカデータの新規登録機能、トルカデータの選択読み込み機能、トルカデータの ID 指定読み込み機能、トルカデータの検索取得機能を利用することはできません。
- 住所リンク機能からの i アプリの起動
住所リンク機能からミニのみアプリを起動することはできません。
- スケジューラからの i アプリの起動
スケジューラからミニのみアプリを起動することはできません。
- データ BOX 保存画像からの i アプリの起動
データ BOX 保存画像からミニのみアプリを起動することはできません。
- FeliCa アドホック通信
FeliCa アドホック通信機能を利用することはできません。
- UART 接続によるシリアル通信
UART 接続によるシリアル通信機能を利用することはできません。
- 外部メモリ
外部メモリ API を利用することはできません。
- 動作認識
動作認識機能を利用することはできません。
- 電子コンパス
電子コンパス機能を利用することはできません。
- 加速度センサー
加速度センサー機能を利用することはできません。
- 脈拍計
脈拍計機能を利用することはできません。
- フォーム分析
フォーム分析機能を利用することはできません。
- 暗号化機能
暗号化機能を利用することはできません。
- タッチパネルデバイス
タッチパネルデバイス機能を利用することはできません。
- シンセサイザー
シンセサイザー機能を利用することはできません。
- i アプリ-Flash 連携
i アプリ-Flash 連携機能を利用することはできません。
- ジャイロセンサー
ジャイロセンサー機能を利用することはできません。
- フルキーボード
フルキーボード機能を利用することはできません。

- 赤外線リモコン受信
赤外線リモコン受信機能を利用することはできません。
- 歩行方位取得機能
歩行方位取得機能を利用することはできません。
- 音楽プレイヤー機能
音楽プレイヤー機能を利用することはできません。
- カメラファインダ機能
カメラファインダ機能を利用することはできません。
- 音声合成機能
音声合成機能を利用することはできません。
- MascotCapsule eruptionST API による 3D グラフィックス描画
MascotCapsule eruptionST API による 3D グラフィックス描画を利用することはできません。
- ステレオ表示機能
ステレオ表示機能を利用することはできません。

A.2 i アプリ API 利用上の注意点

A.2.1 デジタルテレビ制御

(1) DTV 機能の連携起動

表示状態が Selected 状態以外のミニアプリが、DTV 機能を連携起動しようとした場合、例外が発生します。

(2) テレビ番組の視聴・録画予約の登録

表示状態が Selected 状態以外のミニアプリが、テレビ番組の視聴・録画予約を新規登録しようとした場合、例外が発生します。

A.2.2 ヘルスケア関連機能

(1) 歩数計機能

表示状態が Selected 状態以外のミニアプリが、携帯電話ネイティブの歩数計機能のオン/オフの状態を制御しようとした場合、例外が発生します。

(2) 身体活動量計機能

表示状態が Selected 状態以外のミニアプリが、携帯電話ネイティブの身体活動量計機能のオン/オフの状態を制御しようとした場合、例外が発生します。