

i アプリコンテンツ開発ガイド for DoJa-5.x/5.x LE
～ i アプリオプション・i アプリ拡張編 ～
第 3.13 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、DoJa-5.x プロファイルに対応する Java 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と主要な搭載機種は以下の通りです。

- DoJa-1.0 プロファイル： 初期の携帯電話モデル
- DoJa-2.0 プロファイル： 現在、このプロファイルバージョンを搭載した携帯電話はありません。
- DoJa-2.1 プロファイル： 2002 年冬以降発売の携帯電話モデル
- DoJa-2.2 プロファイル： 2003 年春以降発売の携帯電話モデル
- DoJa-3.0 プロファイル： 現在、このプロファイルバージョンを搭載した携帯電話はありません。
- DoJa-3.5 プロファイル： 900i シリーズ
- DoJa-4.0 プロファイル： 901i シリーズ
- DoJa-4.1 プロファイル： 902i シリーズ
- DoJa-5.0 プロファイル： 903i シリーズ、904i シリーズ
- DoJa-5.1 プロファイル： 905i シリーズ、906i シリーズ

また DoJa-5.x プロファイルには、エントリー向けモデルのために一部機能を省略した DoJa-5.x LE (Limited Edition) プロファイルが派生仕様として存在しています。

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.....	12
2.1 ズーム座標系	13
2.2 透明色付きイメージ	13
2.3 ラスタオペレーション.....	14
2.4 その他.....	15
第3章 2D アニメーション再生.....	17
第4章 3D グラフィックス描画.....	18
4.1 高レベル 3D グラフィックス描画	18
第5章 サウンド再生	21
5.1 MFi/Phrase	21
5.2 エフェクト.....	22
5.3 i モーションのオーディオトラックの再生	23
第6章 プラットフォームリソース制御.....	24
第7章 サブディスプレイ描画	29
第8章 携帯電話ネイティブの データストレージアクセス	30
第9章 ポインティングデバイス.....	32
第10章 指紋センサー.....	34
10.1 指紋認証機能	34
10.2 ナビゲーション機能	35
第11章 ピクチャーライト操作.....	37
第12章 アプリケーション連携拡張.....	39
12.1 映像データ管理機能の呼び出し	39
12.2 アラーム機能の呼び出し	40
12.3 トルカ管理機能の呼び出し	41
12.3.1 Toruca クラス	41
12.3.2 トルカデータの新規登録.....	42
12.3.3 トルカデータの選択読み込み	43
12.3.4 トルカデータの ID 指定読み込み	43
12.3.5 トルカデータの検索取得.....	44
12.4 マイプロフィール管理機能の呼び出し	45
12.5 デコメールテンプレート管理機能の呼び出し	46
12.5.1 デコメールテンプレートの新規登録.....	47
12.5.2 デコメールテンプレートの選択読み込み	47
12.5.3 デコメールテンプレートの ID 指定読み込み	48
12.6 住所リンク機能からの i アプリの起動	49

12.7	データ BOX フォルダ管理機能の呼び出し.....	50
12.7.1	フォルダの新規登録.....	50
12.7.2	フォルダのエントリ ID の選択取得.....	51
第 13 章	ラジオチューナー制御.....	53
第 14 章	カメラ拡張.....	56
14.1	撮影画像のエンコード形式.....	56
第 15 章	FeliCa チップ.....	57
15.1	i モード携帯電話に搭載される FeliCa チップの概要.....	57
15.2	フリーエリアアクセス API.....	58
15.3	フリーエリア利用のための ADF の設定.....	60
15.4	外部 R/W からのフリーエリアへのアクセス.....	62
15.5	フリー領域のセキュリティに関する注意事項.....	64
15.6	外部 R/W からの指示による携帯電話内の i アプリ起動.....	65
15.7	FeliCa チップによるアドホック通信.....	68
15.7.1	FeliCa アドホック通信 API の概要.....	68
15.7.2	通信リンク制御.....	70
15.7.3	データ送受信.....	72
第 16 章	UART 接続によるシリアル通信.....	76
第 17 章	外部メモリ.....	78
17.1	外部メモリ API の API 構成.....	78
17.2	外部メモリ API の利用.....	81
17.2.1	StorageDevice クラスの利用.....	81
17.2.2	フォルダの操作.....	83
17.2.3	ファイルの操作と入出力.....	84
17.3	外部メモリ API に固有の例外.....	86
17.4	外部メモリを利用するための ADF の設定.....	86
17.5	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.....	88
17.6	メーカー実装に起因する外部メモリ利用上の注意事項.....	88
17.6.1	暗号化機能利用時の注意事項.....	88
17.6.2	ファイル名に関する注意事項.....	89
第 18 章	デジタルテレビ制御.....	90
18.1	DTV 機能の連携起動.....	90
18.2	テレビ番組の視聴・録画予約の登録.....	91
第 19 章	動作認識.....	93
第 20 章	電子コンパス.....	96
第 21 章	加速度センサ.....	98
第 22 章	ユーティリティ API.....	103
22.1	暗号化機能.....	103

第 23 章	ヘルスケア関連機能	105
23.1	歩数計機能.....	105
23.2	脈拍計機能.....	107
第 24 章	読み上げ機能.....	109
第 25 章	赤外線リモコン受信機能	111

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。

- J2ME/CLDC で規定された 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、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、J2ME/CLDC 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。
	透明色付きイメージ	透明色を指定することができるイメージです。任意の色を透明色に指定したり、透明色の使用有無を切り替えたりすることができます。
	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。
2D アニメーション再生	-----	独立した API ではありませんが、VisualPresenter がサポートするメディアデータを拡張し、スムーズな 2D アニメーションの再生を行います。 なお本機能においては、2D アニメーション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。

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。 なお本機能においては、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。 DoJa-4.0 プロファイル以降では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がありません。このため、過去のアプリケーションの互換性の観点から、引き続き DoJa-4.0 プロファイル以降でも i アプリ拡張 API として本機能が規定されています。
サウンド再生	MFi/Phrase	携帯電話の持つ同時発音可能数の範囲で、複数の i メロディ形式データを並列再生する機能を提供します。
	エフェクト	サウンドデータに設定されているパンポットを一定の範囲で変更したり、同一サウンドデータを繰り返して再生する機能を提供します。
プラットフォームリソース制御	音量制御	メロディ、効果音、マイク（イヤホンマイク接続時のみ）の音量を制御します。
	ソフトキー背景色制御	ソフトキーラベルの背景色を制御します。
	追加キー制御	テンキー、上下左右決定キー、ソフトキー以外の特殊なキーをサポートします。
	背面 LCD イルミネーション制御	背面 LCD イルミネーションを備える携帯電話において、その点灯状態を制御します。
	明度・コントラスト制御	ディスプレイやサブディスプレイの明度やコントラストを制御します。
	メモ LED 制御	メモ LED のオンオフを制御します。
	画面可視・不可視状態の参照	メインディスプレイをユーザーが見ることができる状態かどうかを参照します。
	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。
サブディスプレイ描画	-----	携帯電話のサブディスプレイに描画する機能を提供します。
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。

	ナビゲーション	指紋センサー機能を搭載している携帯電話において、その指紋センサーをユーザー操作入力デバイスとして使用するための機能を提供します。この機能を利用することで、指紋センサー部をタッチパッドのような入力デバイスとして使用することができます。
ピクチャーライト操作	----	ピクチャーライトとは、カメラデバイスに付属している撮影補助用のライトです。この機能を利用することで、i アプリからピクチャーライトの操作（消灯・点灯・高輝度発光の切り替え）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。
	アラーム機能の呼び出し	目覚まし時計などのネイティブのアラーム機能に対し、アラーム時刻を登録する機能を提供します。
	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 902i シリーズから導入されたサービスです）。
	マイプロフィール管理機能の呼び出し	携帯電話のマイプロフィール情報（所有者情報）の参照を行う機能を提供します。
	デコメールテンプレート管理機能の呼び出し	i アプリがネイティブのデコメールテンプレート管理機能呼び出し、デコメールテンプレートデータの登録や参照を行う機能を提供します。
	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。
	データ BOX フォルダ管理機能の呼び出し	i アプリからデータ BOX にフォルダを作成したり、データ BOX からユーザーにフォルダを選択させてそのエン트리 ID を取得するための機能を提供します。
ラジオチューナー制御	----	ラジオチューナーを搭載している携帯電話において、i アプリからラジオの電源オンオフや選局、音量などの制御を行う機能を提供します。
カメラ拡張	----	カメラ制御機能（i アプリ基本 API）の拡張機能を提供します。現在のプロファイルでは、カメラ拡張の機能では撮影画像のオプションのエンコード方式を提供しています。
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	----	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を調べる機能を提供します。
加速度センサ	----	加速度センサデバイスを用いて、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。
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。
読み上げ機能	----	String オブジェクトもしくは XString オブジェクトの形で表現されたテキストを音声で読み上げるユーザーインタフェース機能を提供します。

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。

【DoJa-3.0】

DoJa-3.0 プロファイルでは、DoJa-2.0 の i アプリオプション API のうち以下の機能が i アプリ基本 API に移行しています。

- ・ 2D グラフィックス描画 : ピクセル操作
- ・ 2D グラフィックス描画 : イメージ回転・反転・縮小拡大表示
- ・ 2D グラフィックス描画 : 描画クリッピング
- ・ 2D グラフィックス描画 : その他（描画内容のコピー）
- ・ サウンド再生 : 一時停止・再開
- ・ サウンド再生 : エフェクト（音階・テンポ・ボリュームの変更）
- ・ サウンド再生 : 再生時間制御
- ・ サウンド再生 : 再生同期イベント
- ・ プラットフォームリソース制御 : パイプレータ制御
- ・ プラットフォームリソース制御 : 折りたたみ開閉状態取得
- ・ プラットフォームリソース制御 : 効果音発音
- ・ カメラ制御

以前よりこれらの機能を搭載していたメーカーの携帯電話では、古い API を互換性のため残している場合があります。しかしこれら互換 API の利用は DoJa-3.0 プロファイル以降では推奨されません。DoJa-3.0 プロファイル以降の携帯電話向けには、上記の機能は i アプリ基本 API に実装されているものを使用するようにしてください。

また、DoJa-3.0 プロファイルで新規に追加された i アプリオプション API の機能は以下の通りです（従来より規定されている機能にも、DoJa-3.0 プロファイルで機能強化が図られているものがあります）。

- ・ コード認識
- ・ 指紋センサー
- ・ ピクチャーライト操作
- ・ アプリケーション連携拡張 : スケジューラの起動
- ・ 3D 対応液晶による立体表示
- ・ カメラ拡張
- ・ FeliCa チップ
- ・ プラットフォームリソース制御 : 画面可視・不可視状態の参照

【DoJa-3.5】

DoJa-3.5 プロファイルでは、DoJa-3.0 の i アプリオプション API のうち以下の機能が i アプリ基本 API に移行しています。

- ・ 2D グラフィックス描画 : イメージマップ
- ・ 2D グラフィックス描画 : スプライト
- ・ 2D グラフィックス描画 : パレット付きイメージ
- ・ コード認識

以前よりこれらの機能を搭載していたメーカーの携帯電話では、古い API を互換性のため残している場合があります。しかしこれら互換 API の利用は DoJa-3.5 プロファイル以降では推奨されません。DoJa-3.5 プロファイル以降の携帯電話向けには、上記の機能は i アプリ基本 API に実装されているものを使用するようにしてください。

また、DoJa-3.5 プロファイルで新規に追加された i アプリオプション API の機能は以下の通りです（従来より規定されている機能にも、DoJa-3.5 プロファイルで機能強化が図られているものがあります）。

- ・ 2D グラフィックス描画 : アフィン変換つきイメージ描画
- ・ ポインティングデバイス : ジョイスティックモード

【DoJa-4.0】

DoJa-4.0 プロファイルで新規に追加された i アプリオプション API の機能は以下の通りです。

- ・ プラットフォームリソース制御 : アロケート可能な連続メモリ量の参照
- ・ プラットフォームリソース制御 : オーディオ出力のサラウンド制御
- ・ 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

【DoJa-4.1】

DoJa-4.1 プロファイルで新規に追加された i アプリオプション API の機能は以下の通りです。

- ・ 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
- ・ アプリケーション連携拡張 : アラーム機能の呼び出し
- ・ アプリケーション連携拡張 : トルカ管理機能の呼び出し
- ・ UART 接続によるシリアル通信
- ・ 外部メモリ
- ・ デジタルテレビ制御

また、DoJa-4.1 プロファイル以降、以下の i アプリオプション API は削除されます。

- ・ コード認識 (*1)
- ・ カメラ制御 (*1)
- ・ 2D グラフィックス描画 : イメージ回転・反転・縮小拡大表示 (*1)
- ・ 2D グラフィックス描画 : イメージマップ (*1)
- ・ 2D グラフィックス描画 : ピクセル操作 (*1)
- ・ 2D グラフィックス描画 : パレット付きイメージ (*1)
- ・ 3D グラフィックス描画 : 低レベル 3D グラフィックス描画 (*2)
- ・ 携帯電話ネイティブのデータストレージアクセスの一部 (*2)
- ・ プラットフォームリソース制御 : 音量制御の一部 (*2)
- ・ プラットフォームリソース制御 : 背面ライト制御(*2)
- ・ 3D 対応液晶による立体表示 (*2)

(*1) これらについては、i アプリ基本 API に規定されている機能を使用するようにしてください。

(*2) これらについては、長期にわたり採用機種がないため廃止されます。

【DoJa-5.0】

アプリケーション連携拡張のうちトルカ管理機能の呼び出しについて、DoJa-5.0 プロファイルではトルカデータの登録、参照、検索取得の3つの機能が規定されています。トルカデータの参照については DoJa-4.1 プロファイルより API が規定されていますが、DoJa-4.1 プロファイルではこの機能はトラステッド API に属する機能であり、一般 i アプリは使用することはできません（DoJa-5.0 プロファイルでは一般 i アプリも使用することができます）。また、トルカデータの検索取得は、DoJa-5.0 プロファイルにて新設された機能です。

また以下の機能は、DoJa-5.0 プロファイルにて新設されました。

- ・ ラジオチューナー制御
- ・ アプリケーション連携拡張におけるマイプロフィール管理機能の呼び出し、デコメールテンプレート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、データ BOX フォルダ管理機能の呼び出し
- ・ 動作認識
- ・ 電子コンパス
- ・ 加速度センサ

なお、DoJa-5.0 プロファイルでは、DoJa-4.1 の i アプリオプション API のうち i アプリからのスケジューラの連携起動が i アプリ基本 API に移行しています。

【DoJa-5.1】

DoJa-5.1 プロファイルで新規に追加された i アプリオプション API の機能は以下の通りです。

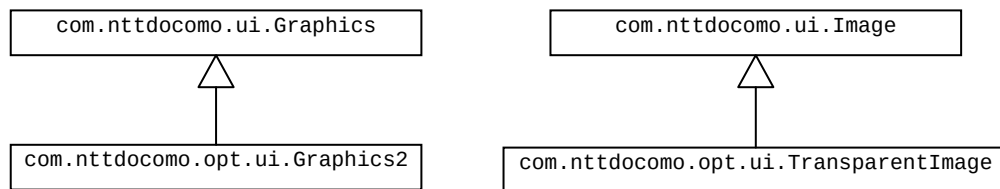
- ・ 2D グラフィックス描画 : その他（表示デバイスの垂直同期間隔の取得）
- ・ 2D グラフィックス描画 : その他（表示デバイスの垂直同期タイミングでのダブルバッファリング終了）
- ・ ユーティリティ API : 暗号化機能
- ・ 歩数計機能
- ・ 読み上げ機能

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.nttdocomo.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。

【DoJa-3.0】 【DoJa-4.0】

DoJa-3.0 プロファイルでは、DoJa-2.0 プロファイルで i アプリオプション API に分類されていた以下の機能が i アプリ基本 API に移行しています。

- 2D グラフィックス描画 : ピクセル操作
- 2D グラフィックス描画 : イメージ回転・反転・縮小拡大表示
- 2D グラフィックス描画 : 描画クリッピング
- 2D グラフィックス描画 : その他（描画内容のコピー）

また DoJa-4.0 プロファイルでは、DoJa-3.0 プロファイルで i アプリオプション API に分類されていた以下の機能が i アプリ基本 API に移行しています。

- 2D グラフィックス描画 : イメージマップ
- 2D グラフィックス描画 : スプライト
- 2D グラフィックス描画 : パレット付きイメージ

本書ではこれらの機能については触れません。

【DoJa-4.1】

DoJa-4.1 プロファイル以降、以下の i アプリオプシオン API は削除されます。DoJa-4.1 プロファイル以降では、これらの機能は i アプリ基本 API にて規定されているものを使用するようにしてください。

- ・ 2D グラフィックス描画 : イメージ回転・反転・縮小拡大表示 (*1)
- ・ 2D グラフィックス描画 : イメージマップ (*1)
- ・ 2D グラフィックス描画 : ピクセル操作 (*1)
- ・ 2D グラフィックス描画 : パレット付きイメージ (*1)

2.1 ズーム座標系

`com.nttdocomo.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 透明色付きイメージ

2D グラフィックス描画機能では、アプリケーションによって自由に透明色を指定できる透明色付きイメージ (`com.nttdocomo.opt.ui.TransparentImage` クラス) をサポートしています。透明色付きイメージは static メソッド `TransparentImage.createTransparentImage()` を、透明色付きイメージとして取り扱いたいイメージを引数に指定して呼び出すことにより取得します。

i アプリ基本 API の `Image` クラスでは、元となる GIF データが透過 GIF である場合に限りその GIF データに設定されている透過色が透明色として扱われます。それに対し、透明色付きイメージでは透過 GIF を使用した画像でなくても、またイメージ中で使用されているどの色に対しても、動的に透明色の指定を行うことができます。透明色の指定には、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) を使用します。

透明色の指定を有効にするには、`TransparentImage.setTransparentEnabled()` メソッドを使用します。初期状態では、透明色の指定は無効化されています。

TransparentImage クラスは **i アプリ基本 API** の **Image** クラスを継承しています。ただし **Image** クラスで定義されている **getGraphics()** メソッドは **TransparentImage** オブジェクトに対して呼び出すことはできません。アプリケーションがこのメソッドを呼び出した場合は例外が発生します。

【DoJa-5.0】

DoJa-5.0 プロファイル以降、**i アプリ基本 API** の標準機能として、**Image** クラスで透明色付きイメージがサポートされました (**Image** クラスにおける **setTransparentColor()** / **getTransparentColor()** / **setTransparentEnabled()** の各メソッドのサポート) 。DoJa-5.0 プロファイル以降では、本項で解説した **TransparentImage** クラスの利用は推奨されません。

2.3 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 Si、描画先のピクセル色の RGB 値(D_R, D_G, D_B)の各要素を Di 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。

$$Di = Di * dstRatio + Si * srcRatio$$

- **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。

$$Di = Si * srcRatio$$

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。

$$Di = Di * dstRatio - Si * srcRatio$$

【DoJa-3.0】

DoJa-2.0 プロファイルでは、ラスタオペレーションにおける色演算は、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) に対して行われます。これに対し、DoJa-3.0 プロファイル以降では、色演算はピクセル色を表す RGB 値に対して行われるよう改められています。

2.4 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.drawImage()**)

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.drawString()**)

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.createImage()**)

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.getColorAt()**)

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.nttdocomo.opt.ui.j3d.AffineTransform** 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 Java 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

2D アニメーション再生

i アプリ基本 API では動画再生機能として、**VisualPresenter** によるアニメーション GIF の再生をサポートしています。アニメーション GIF は、その作成手段も広く普及しておりコンテンツの流通量も豊富である一方、容量、再生品質、再生時間の面で最新の 2D アニメーション技術と比較して劣っています。

2D アニメーション再生機能は、最新の 2D アニメーション技術を活用した動画コンテンツを再生する手段を提供します。

2D アニメーション再生のための特別なクラスやメソッドはありません。アニメーション GIF の再生と同様、2D アニメーションの再生には **VisualPresenter** を使用します。

なお、現在のプロファイルでは 2D アニメーション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 2D アニメーション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 2D アニメーション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

3D グラフィックス描画

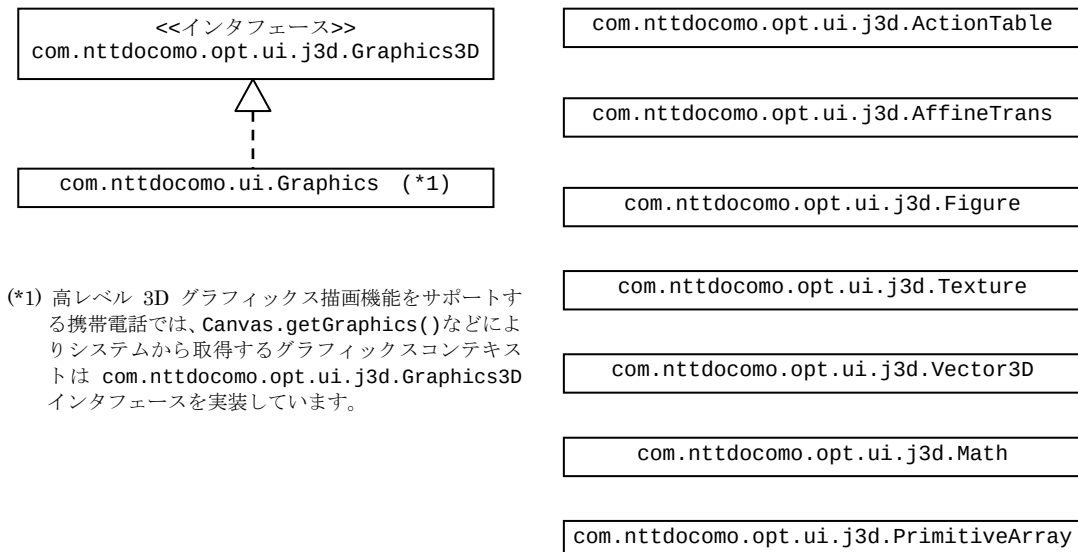
3D グラフィックス描画機能では、外部から取り込んだ 3D モデルデータを操作、表示する機能を提供します。以前のプロファイルでは、3D グラフィックス描画機能は高レベル API と低レベル API の 2 つから構成されており、それぞれで独立した API セットが定義されていましたが、DoJa-4.1 プロファイル以降では低レベル API は廃止されました。

ここでは、高レベル API（高レベル 3D グラフィックス描画）について解説します。

4.1 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.nttdocomo.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され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されるクラス・メソッドでは、CLDC-1.0 で利用可能な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。

【DoJa-3.0】

DoJa-3.0 プロファイルの高レベル 3D グラフィックス描画機能では、DoJa-2.0 プロファイルと比較して以下のような点で機能強化されています。

- ・ 透視投影のサポート
- ・ プリミティブ（点、線、三角形、四角形、ポイントスプライト）描画のサポート
- ・ コマンドリストによる一括処理のサポート
- ・ 頂点アニメーションによる外見状態変化のサポート
- ・ 1 モデルに対する複数テクスチャ設定のサポート

また複数モデル、複数プリミティブの描画をサポートするために、DoJa-3.0 プロファイルではレンダリング演算と描画を個々に独立して行うためのメソッドが用意されています（レンダリングと描画を一括して行う従来のドローメソッド（`Graphics3D.drawFigure()`メソッド）も互換性のため残されています）。

【DoJa-4.0】

DoJa-4.0 プロファイルでは、本機能に相当する機能が i アプリ基本 API として取り入れられています。ただし、DoJa-4.0 プロファイルにおける i アプリ基本 API の 3D グラフィックス描画機能は本 API をベースとはしていません。DoJa-3.x プロファイル以前の機種向けに作成された高レベル 3D グラフィックスアプリケーションを簡易に DoJa-4.0 プロファイル対応携帯電話上で動作させるには、従来通り i アプリ拡張 API の高レベル 3D グラフィックス描画機能を使用するようにしてください。

第 5 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.nttdocomo.ui.AudioPresenter` に定義される拡張メソッド
- `com.nttdocomo.opt.ui.AudioPresenter2` に定義される拡張属性
- `com.nttdocomo.ui.AudioTrackPresenter`

サウンド再生機能で拡張される機能には、以下のようなものがあります。

- MFi/Phrase
- エフェクト
- i モーションのオーディオトラックの再生

以下、各機能の概略を解説します。

【DoJa-3.0】

DoJa-3.0 プロファイルでは、DoJa-2.0 プロファイルで i アプリオプション API に分類されていた以下の機能が i アプリ基本 API に移行しています。本書ではこれらの機能については触れません。

- サウンド再生 : 一時停止・再開
- サウンド再生 : エフェクト（音階・テンポ・ボリュームの変更）
- サウンド再生 : 再生時間制御
- サウンド再生 : 再生同期イベント

【DoJa-5.0】

i モーションのオーディオトラックの再生機能は、DoJa-5.0 プロファイルにて新設されました。

5.1 MFi/Phrase

MFi/Phrase は、携帯電話の持つ音源の最大同時発音数（チャンネル数）を複数の仮想的なトラックに分割し、チャンネル数の制限の範囲内で複数のサウンドデータを同時に再生する機能を提供します。現在のプロファイルでは、使用可能なトラック数は 4 つまでとなります。

MFi/Phrase は以下の手順で使します。

- 1) `AudioPresenter.getAudioPresenter()`メソッドを使用して、複数の `AudioPresenter` オブジェクトを取得します。
- 2) 各々の `AudioPresenter` に対し、`setAttribute()`メソッドでトラック番号を重複しないように割り当てます。`setAttribute()`メソッドには、属性として `AudioPresenter2.TRACK_ASSIGNMENT` を、値としてトラック番号 (0~3) を指定します。
- 3) 各々の `AudioPresenter` に対し、`MediaSound` オブジェクトを設定します。`MediaSound` の元となるサウンドデータに `MFi/Phrase` のための特殊な設定を行う必要はありませんが、同時発音数の合計がチャンネル数を超えることはできません。
- 4) 各々の `AudioPresenter` に対し、`play()`メソッドで再生を指示します。

`MFi/Phrase` のトラック割り当てが行われた `AudioPresenter` の再生開始は、`MFi/Phraes` のトラック割り当てが行われていない全ての `AudioPresenter` が停止した状態でなければ行うことはできません。また、`MFi/Phraes` のトラック割り当てが行われていない `AudioPresenter` の再生は、`MFi/Phraes` のトラック割り当てが行われている全ての `AudioPresenter` が停止した状態でなければ行うことはできません。

5.2 エフェクト

サウンド再生の拡張では、`AudioPresenter2` の属性としてパンポットやループ再生が定義されており、アプリケーションからこれらの属性を制御することができます。この機能を使用することで、再生するサウンドに一定の効果を掛けることができます。

`AudioPresenter2` に定義された属性は、`AudioPresenter.setAttribute()`メソッドに指定して使用します。

以下に `AudioPresenter2` で定義されているエフェクト機能の属性、および指定可能な属性値を示します。

属性	意味・取り得る属性値
LOOP	ループを示す属性です。属性値には、ループ回数を設定します。
PANPOT	パンポット（ステレオ効果における音の位置）を示す属性です。属性値には 0~127 までの数値を指定します。パンポットのデフォルト値は、64（中央）です。パンポット変更機能は、 <code>MFi/Phrase</code> 使用時のみ使用可能です。

PANPOT 属性は、`AudioPresenter` が再生を行っている最中に属性値を変更すると、その時点から再生に効果を与えます。`AudioPresenter` が再生を行っている最中に LOOP 属性を設定しても無視されます。

【DoJa-5.0】

DoJa-5.0 プロファイル以降、`AudioPresenter2.LOOP` と類似の API として、`AudioPresenter` クラスに `LOOP_COUNT` 属性が設けられました。この属性では、`AudioPresenter.setAttribute()`メソッドでループの回数を指定することで任意の回数の繰り返し再生を指示することができます。また、この方法で繰り返し再生が行われる際は、`AudioPresenter` に設定されたリスナーで `AudioPresenter.AUDIO_LOOPED` イベントの通知を受けることもできます。

`LOOP_COUNT` 属性もオプションの API ですが、DoJa-5.0 プロファイル以降では LOOP 属性の代わりに `LOOP_COUNT` 属性を使用することを推奨します。

5.3 i モーションのオーディオトラックの再生

携帯電話は、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。DoJa-5.0 プロファイル以降の携帯電話には、**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。**VisualPresenter** 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 **MediaImage** 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(MediaImage)** 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。

第 6 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.nttdocomo.ui.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
- 電池残量
- 電波強度
- マナーモード設定状況
- 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.nttdocomo.ui.PhoneSystem` クラス、`com.nttdocomo.opt.ui.PhoneSystem2` クラス、`com.nttdocomo.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。

- 音量制御（`PhoneSystem2` クラス）
- ソフトキーの背景色制御（`PhoneSystem2` クラス）
- 追加キー制御（`Display` クラス、`PhoneSystem` クラス、`PhoneSystem2` クラス）
- 背面 LCD イルミネーション制御（`PhoneSystem2` クラス）
- メモ LED 制御（`PhoneSystem2` クラス）
- 明度・コントラスト制御（`PhoneSystem2` クラス）
- 画面可視・不可視状態の参照（`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）

以下、各機能の概要について解説します。

<音量制御>

携帯電話の各種音量を制御する、参照・設定の両方が可能な属性です。以下の属性が `PhoneSystem2` クラスに定義されています。

- `DEV_MELODY_VOLUME` (メロディ再生音量)
- `DEV_SE_VOLUME` (携帯電話組み込みの効果音再生音量)

属性値には、`PhoneSystem2.ATTR_VOLUME_MIN` (最小) から `PhoneSystem2.ATTR_VOLUME_MAX` (最大) までの間の値を設定します。

<ソフトキーの背景色制御>

ソフトキー表示エリアおよびインジケータエリアの背景色を制御するための、参照・設定の両方が可能な属性です。属性 `PhoneSystem2.DEV_INDICATOR_BACKGROUND` に対し、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) を設定することでソフトキー表示エリアおよびインジケータ表示エリアの背景色を変更することができます。

<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、待ち受け i アプリ切り替えキーおよび 2 つのソフト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

対応する定数として、`com.nttdocomo.ui.Display` クラスに `KEY_PAGE_DOWN` および `KEY_PAGE_UP` が定義されています。

- 上下左右キーの 8 方向サポート

対応する定数として、`com.nttdocomo.ui.Display` クラスに `KEY_LOWER_LEFT`、`KEY_LOWER_RIGHT`、`KEY_UPPER_LEFT`、`KEY_UPPER_RIGHT` が定義されています。

なお、8 方向サポートを有効にするには、属性 `PhoneSystem2.DEV_KEY_SLANT` に対して属性値 `PhoneSystem2.ATTR_KEY_SLANT_ON` を設定します。

また DoJa-3.0 プロファイル以降、携帯電話に上記以外に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追加されています。

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。

<背面 LCD イルミネーション制御>

背面 LCD（サブディスプレイ）のバックライトの点灯色を制御するための、設定のみ可能な属性です。属性 `PhoneSystem2.DEV_ILLUMINATION` に対し、背面 LCD イルミネーションの点灯色を表す属性値（`PhoneSystem2` クラスで定義されている、`ATTR_ILLUMINATION` で始まる名前の属性値）を設定します。なお、属性値として、点灯色と `PhoneSystem2.ATTR_ILLUMINATION_GRADUALLY` の論理和を取ったものを設定すると、その点灯色で徐々に点灯します。

<メモ LED 制御>

メモ LED の点灯色を制御するための、参照・設定の両方が可能な属性です。属性 `PhoneSystem2.DEV_MEMO_LED` に対し、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）を設定することでメモ LED の点灯色を選択できます。

なお、属性値として `PhoneSystem2.ATTR_MEMO_LED_ON` を設定した場合はデフォルトの点灯色で点灯します。また、`PhoneSystem2.ATTR_MEMO_LED_OFF` を設定した場合はメモ LED を消灯します。

<明度・コントラスト制御>

ディスプレイおよびサブディスプレイの明度およびコントラストを制御するための、参照・設定の両方が可能な属性です。以下の属性が `PhoneSystem2` クラスに定義されています。

- `DEV_DISPLAY_BRIGHTNESS`（ディスプレイの明度）
- `DEV_DISPLAY_CONTRAST`（ディスプレイのコントラスト）
- `DEV_SUBDISPLAY_BRIGHTNESS`（サブディスプレイの明度）
- `DEV_SUBDISPLAY_CONTRAST`（サブディスプレイのコントラスト）

明度の属性値には、`PhoneSystem2.ATTR_BRIGHTNESS_MIN`（最小）から `PhoneSystem2.ATTR_BRIGHTNESS_MAX`（最大）までの間の値を設定します。またコントラストの属性値には、`PhoneSystem2.ATTR_CONTRAST_MIN`（最小）から `PhoneSystem2.ATTR_CONTRAST_MAX`（最大）までの間の値を設定します。

<画面可視・不可視状態の参照>

メインディスプレイをユーザーが見ることができる状態かどうかを示す、参照のみ可能な属性です。ストレート型携帯電話ではメインディスプレイは常に露出しているため、ユーザーは常にメインディスプレイを見ることができる状態です。また、通常の折りたたみ型携帯電話では、機体が開いた状態でのみユーザーはメインディスプレイを見ることができます。これに対し、回転型の機体を持つ携帯電話では、機体が閉じた状態であってもユーザーがメインディスプレイを見ることができる状態が存在する場合があります。

この属性をサポートする機種では、機体の折りたたみ状態（`PhoneSystem.DEV_FOLDING`）とは独立に、現在ユーザーがメインディスプレイを見ることができる状態か否かを参照することができます。

画面可視・不可視状態を参照するには、属性 `PhoneSystem.DEV_SCREEN_VISIBLE` を使用します。`getAttribute()` メソッドが `ATTR_SCREEN_VISIBLE` を返した場合は可視状態、`ATTR_SCREEN_INVISIBLE` を返した場合は不可視状態であることを示します。

<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。

KVM 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。

【DoJa-4.x】 【DoJa-5.0】

DoJa-4.x プロファイルでは、i アプリがサラウンド機能のオン、オフを制御するためにはネイティブのユーザー設定によってサラウンド機能が有効化されていなければなりません。ネイティブのユーザー設定の条件も含めて i アプリからサラウンド機能のオン、オフを制御可能かどうかは、`PhoneSystem.isAvailable()` メソッドにより調べることができます。これに対し、DoJa-5.0 プロファイル以降ではネイティブのサラウンド設定には関係なく i アプリからサラウンド機能のオン、オフを制御することができます。

<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.nttdocomo.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。

【DoJa-5.1】

キャンバス上の座標系の原点は、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。

DoJa-5.1 プロファイルでは、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の `Canvas2` クラスが、i アプリオプション API として `com.nttdocomo.opt.ui` パッケージに追加されています。`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パスの座標系を指示（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パ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。

【DoJa-3.0】

DoJa-3.0 プロファイルでは、DoJa-2.0 プロファイルで i アプリオプション API に分類されていた以下の機能が i アプリ基本 API に移行しています。

- ・ プラットフォームリソース制御 : パイプレータ制御
- ・ プラットフォームリソース制御 : 折りたたみ開閉状態取得
- ・ プラットフォームリソース制御 : 効果音発音

また、キーグループの指定による追加キー制御、および画面可視・不可視状態の参照は、DoJa-3.0 プロファイルにて新たに追加されました。

【DoJa-4.0】

アロケート可能な連続メモリ量の参照およびオーディオ出力のサラウンド制御は、DoJa-4.0 プロファイルにて新たに追加されました。

【DoJa-5.0】 【DoJa-5.1】

ディスプレイスタイル状態の参照は、DoJa-5.0 プロファイルにて新たに追加されました。なお、属性値 **ATTR_DISPLAY_STYLE_REVERSE** は DoJa-5.1 プロファイルで追加されており、DoJa-5.0 プロファイルには存在していません。

第 7 章

サブディスプレイ描画

サブディスプレイ描画機能では、サブディスプレイ（携帯電話の背面などに取り付けられている小さな液晶ディスプレイ）に対し描画する機能を提供します。サブディスプレイへの描画には、`com.nttdocomo.opt.ui.SubDisplay` クラスを使用します。

`SubDisplay` クラスは、サブディスプレイのデバイス情報を取得するメソッドとサブディスプレイへの描画メソッドを備えるシンプルなユーティリティクラスです。

<サブディスプレイのデバイス情報取得>

`SubDisplay` クラスには、サブディスプレイのデバイス情報を取得するためのメソッドとして以下のものが用意されています。

- `getHeight()` : スクリーンの高さを取得します。
- `getWidth()` : スクリーンの幅を取得します。
- `isColor()` : スクリーンがカラー表示をサポートしているかどうかを調べます。
- `numColors()` : スクリーンで使用できる色数を取得します。

i アプリ基本 API の `Display` クラスにも同名のメソッドが用意されていますが、それらのメソッドと上記メソッドの使用方法は共通です。

<サブディスプレイへの描画>

サブディスプレイへの描画は、ディスプレイへの描画と異なり直接 `SubDisplay` クラスに描画指示を行います。サブディスプレイに描画できるものはイメージ（`Image` オブジェクト）のみです。

サブディスプレイへのイメージの描画には、`SubDisplay.setImage()` メソッドを使用します。

第 8 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 キャラクターデータなど、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.nttdocomo.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。

【DoJa-4.1】

DoJa-4.1 プロファイルにて、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ためのメソッドが `PhoneResource.getInformationDisplayData()` として追加されました。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 9 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.nttdocomo.opt.ui.PointingDevice` クラスで提供されます。なお、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。

<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はできないため、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()**

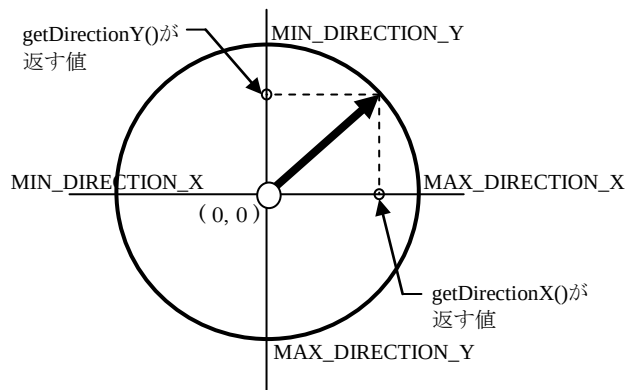
デバイスの現在の操作方向の、X 軸（横方向）の成分を取得します。この値の取り得る範囲は、**PointingDevice** クラスに定義されている定数 **MIN_DIRECTION_X** から **MAX_DIRECTION_X** までとなります。

- **getDirectionY()**

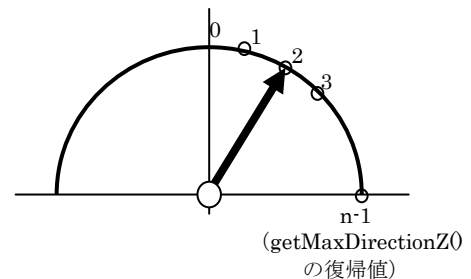
デバイスの現在の操作方向の、Y 軸（縦方向）の成分を取得します。この値の取り得る範囲は、**PointingDevice** クラスに定義されている定数 **MIN_DIRECTION_Y** から **MAX_DIRECTION_Y** までとなります。

- **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

第 10 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 として以下の 2 つの機能を規定しています。

- ・ 指紋認証機能（デバイスを使用して入力された指紋データと携帯電話内に登録されている指紋データとを照合します）
- ・ ナビゲーション機能（デバイス上に触れられた指の動きを i アプリに伝達します）

以下、それぞれの機能の概要を解説します。

10.1 指紋認証機能

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.nttdocomo.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트리 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트리 ID を指定します。そのエン트리 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트리 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て、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。

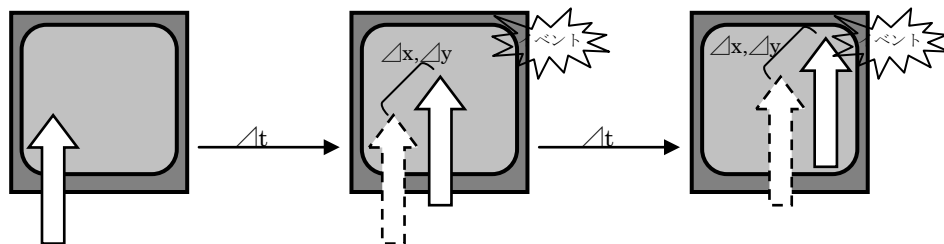
注意事項：

- 非活性化状態の待ち受けアプリケーションからは、`FingerprintAuthenticator.select()` メソッドおよび `FingerprintAuthenticator.authenticate()` メソッドを呼び出すことはできません。
- 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。

10.2 ナビゲーション機能

ナビゲーション機能では、指紋センサー上に触れられたユーザーの指の動きを i アプリに伝える機能を提供します。ナビゲーション機能をサポートする指紋センサーでは、一定間隔周期でデバイス上の指の位置をセンシングし、その変化の量（1 周期間隔における指の移動量）をイベントとして i アプリに通知します。開発者はこの情報を利用して、アプリケーションプログラムにタッチパッドのような入力デバイス機能を実装することが可能となります。



周期 Δt ごとにデバイス上の指の位置の移動量 ($\Delta x, \Delta y$) を求め、定期的に発生するイベントとして i アプリに通知します。

ナビゲーション機能は `com.nttdocomo.opt.device.FingerSensor` クラスで提供されます。

ナビゲーション機能を使用するには、以下の手順を踏みます。

- 1) static メソッド `FingerSensor.getFingerSensor()` を呼び出して、`FingerSensor` オブジェクトを取得します。
- 2) `FingerSensor` オブジェクトの `setAttribute()` メソッドを使用して、センサーのセットアップを行います。セットアップ項目には以下の 2 種類があります。

- ・ センサーが指の動きをセンスする間隔（センス周期 DEV_SENSE_INTERVAL）
- ・ 指の動きがない場合におけるイベント通知有無（DEV_NULL_EVENT_NOTIFICATION）

3) **FingerSensor** オブジェクトの **setEnabled()** メソッドを使用して、ナビゲーションモードの有効、無効を切り替えます。

ナビゲーション機能のセンス周期イベントは、低レベルイベント

Display.FINGER_MOVED_EVENT としてアプリケーションプログラム（**Canvas** 継承オブジェクトの **processEvent()** メソッド）に通知されます。従って、ナビゲーション機能はカレントフレームが **Canvas** である場合にのみ利用することができます。

センス周期イベントが発生したとき、**FingerSensor** オブジェクトはそのセンス処理で取得した指の移動量を保持しています。この情報は、**getSenseData()** メソッドを使用して取り出すことができます。

注意事項：

- ナビゲーション機能はカレントフレームに **Canvas** である場合にのみ使用できますが、ナビゲーションモードの有効、無効の状態はカレントフレームの種類に関係なく保持されています。つまり、ナビゲーションモードが有効な状態でカレントフレームを **Panel** に設定すると低レベルイベントが通知されなくなるためナビゲーション機能は使用不能となりますが、その後カレントフレームを **Canvas** に設定すると再度センス周期イベントが発生するようになります。
- ナビゲーション機能は、非活性化状態および休眠状態の待ち受けアプリケーションから使用することはできません。ナビゲーションモードが有効な状態で待ち受けアプリケーションが非活性化状態に遷移すると、自動的にナビゲーションモードは無効化されます。活性化状態への復帰で自動的に有効化されることはありません。
- ナビゲーションモードが有効な状態で i アプリがサスペンド、レジュームした場合、自動的にナビゲーションモードは無効化されます。レジューム後に自動的に有効化されることはありません。

第 11 章

ピクチャーライト操作

ピクチャーライトとは、カメラデバイスに付属している撮影補助用のライトです。ピクチャーライト操作機能を使用することで、i アプリからピクチャーライトをカメラデバイスと切り離して制御（消灯・点灯（通常発光）・高輝度発光）することができます。ピクチャーライト操作機能は、`com.nttdocomo.opt.device.PictureLight` クラスで提供されます。

ピクチャーライトを使用するには、まず `PictureLight` クラスの static メソッド `getPictureLight()` を呼び出して `PictureLight` オブジェクトを取得します。この `PictureLight` オブジェクトに対して、発光制御や各種状態設定・参照を行います。

発光制御

発光制御系のメソッドには、`light()` メソッドおよび `flash()` メソッドがあります。

● `light()` メソッド

`light()` メソッドでは、ピクチャーライトの点灯・消灯を制御します。ピクチャーライトを点灯するにはメソッドの引数に `true` を、消灯するには `false` を指定します。消灯時に消灯指示を行ったり、点灯時または高輝度発行時に点灯指示を行っても何も行われません。

● `flash()` メソッド

`flash()` メソッドではピクチャーライトの高輝度発光を制御します。`flash()` メソッドを呼び出すと、一定時間ピクチャーライトが高輝度発光します。

電力消費の観点から、高輝度発光する時間をアプリケーションプログラムが自由に決定することはできません。ただし以下に解説する `setTime()` メソッドを使用することで、高輝度発光時間を長・中・短の 3 段階から選択することは可能です。また、一度高輝度発光を行ってから次の高輝度発光を行うまでには一定の間隔が必要です。現在高輝度発光を行えるかどうかは、以下に解説する `isFlashBusy()` メソッドで調べることができます。

状態設定・参照

状態設定・参照系のメソッドには以下のものがあります。

● `setColor()` メソッド・`getColor()` メソッド

これらのメソッドを使用することで、ピクチャーライトの点灯時または高輝度発行時の色を設定、参照することができます。色情報には、各機種に固有の色を表す整数値 (`Graphics.getColorOfRGB()` メソッドなどで返される値) を使用します。指定した色がピクチャーライトでサポートされていない場合、最も近い色に補正されます。

消灯状態、点灯状態で `setColor()` メソッドを呼び出した場合は即座にデバイス設定に反映されますが、高輝度状態で `setColor()` メソッドを呼び出した場合はメソッド呼び出しが無視されます。

- setTime()メソッド・getTime()メソッド

これらのメソッドを使用することで、flash()メソッド呼び出し時の高輝度発光時間を設定、参照することができます。高輝度発光時間は高 (PictureLight.FLASH_LONG)、中 (PictureLight.FLASH_MIDDLE)、短 (PictureLight.FLASH_SHORT) のいずれかで表されます。

高輝度発光中にこのメソッドを呼び出しても何も行われません。

- isLightOn()メソッド

ピクチャーライトの点灯状態を返します。点灯状態および高輝度発光状態では true が、消灯状態では false が返されます。

- isFlashBusy()メソッド

高輝度発光がビジー状態かどうかを返します。高輝度発光を行った直後で、次の高輝度発光を行えない場合は true が、そうでなければ false が返されます。

このメソッドが true を返している間は、flash()メソッドを呼び出しても何も行われません。

注意事項：

- i アプリによるピクチャーライトの利用とカメラデバイスの利用が競合した場合、ピクチャーライトはカメラデバイスと連動した動作を行います。
- i アプリがピクチャーライトを利用している状態でサスペンドが発生したり i アプリが終了したりすると、ピクチャーライトは自動的に消灯します。

第 12 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、映像データ管理機能の呼び出し、アラーム機能の呼び出し、トルカ管理機能の呼び出し、マイプロフィール管理機能の呼び出し、デコメールテンプレート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、およびデータ BOX フォルダ管理機能の呼び出しが規定されています。

【DoJa-5.0】

DoJa-4.1 プロファイルで i アプリオプション API として規定されていた i アプリからのスケジューラの連携起動は、DoJa-5.0 プロファイル以降は i アプリ基本 API の位置づけとなっています。

12.1 映像データ管理機能の呼び出し

DoJa-4.1 プロファイル以降、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が i アプリ基本 API として使用可能となりました（DoJa-4.0 プロファイルでは、映像データの新規登録はトラステッド i アプリのみ使用可能です）。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.nttdocomo.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 `MediaImage` 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。

注意事項：

- ダウンロード即起動 i アプリ、および非活性化状態の待ち受けアプリケーションからは映像データの選択読み込みを行うことはできません。

(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 `MediaImage` オブジェクトを取得することができます。

注意事項：

- ダウンロード即起動 i アプリからは映像データの ID 指定読み込みを行うことはできません。また、この機能はユーザーの操作を必要としないため、非活性化状態の待ち受けアプリケーションも使用することができます。

12.2 アラーム機能の呼び出し

アラーム機能とは、目覚まし時計などの名称で実装されているネイティブアプリケーションです。本項の機能を利用することで、i アプリからネイティブのアラーム機能にアクセスし、アラーム時刻の新規登録を行うことができます。

なお、アラーム機能を使用する 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。

アラーム時刻を新規登録するには、`com.nttdocomo.system.Alarm` 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は、登録したいアラーム時刻を設定した `ScheduleDate` オブジェクトを指定します。`ScheduleDate` クラスの使用方法は、スケジュール管理機能の呼び出し（11.9 項を参照）における `ScheduleDate` クラスの使用方法と同様です。

アラーム時刻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 **Alarm.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) アラーム時刻の登録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。この画面では、ユーザーは i アプリからのアラーム時刻の新規登録を許可、キャンセルすることができます。
- (3) ユーザーがアラーム時刻の新規登録を許可するとアラーム時刻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。

ユーザーがアラーム時刻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。

注意事項：

- ダウンロード即起動 i アプリ、および非活性化状態の待ち受けアプリケーションからはアラーム時刻の新規登録を行うことはできません。
- アラーム時刻の登録において、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。機種ごとにどのような日時指定タイプが使用できるかは、**Alarm.getSupportedTypes()** メソッドを使用して調べることができます。

12.3 トルカ管理機能の呼び出し

トルカ機能は、902i シリーズより導入されました。DoJa-4.1 プロファイル以降では、i アプリがネイティブのトルカ管理機能呼び出し、トルカデータを新規登録するための API が i アプリオブジェクト API として規定されています。また DoJa-5.0 プロファイル以降では、i アプリから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したりできるよう機能が拡張されています。

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。

【DoJa-5.0】

トルカデータ読み込み機能は DoJa-4.1 プロファイルより規定されていますが、DoJa-4.1 プロファイルではこの機能はトラステッド API に属しており、一般 i アプリでは使用することはできません。DoJa-5.0 プロファイル以降では、一般 i アプリもトルカデータの読み込みを行うことができます。

また、トルカデータの検索取得機能は、DoJa-5.0 プロファイルにて新設されました。

12.3.1 Toruca クラス

i アプリでは、トルカデータは **com.nttdocomo.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。

【DoJa-5.0】

DoJa-5.0 プロファイル以降では、**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。

また DoJa-5.0 プロファイルでは、**Toruca** クラスについてトルカデータのコンテンツフォーマットのバージョンアップに伴うメソッドの新設（**getColorID()**/ **getExpirationDate()**/ **getIPID()**/ **getRedistributionID()**/ **getSortID()**）が行われています。

12.3.2 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.nttdocomo.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 に対応するエントリ 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。

注意事項：

- ダウンロード即起動 i アプリ、および非活性化状態の待ち受けアプリケーションからはトルカデータの新規登録を行うことはできません。

12.3.3 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：

- ダウンロード即起動 i アプリ、および非活性化状態の待ち受けアプリケーションからは、トルカデータの選択読み込みを使用することはできません。
- 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。

12.3.4 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

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：

- トルカデータの ID 指定読み込みはユーザーの操作を必要としないため、非活性化状態の待ち受けアプリケーションも使用することができます。ダウンロード即起動 i アプリからは、トルカデータの ID 指定読み込みを使用することはできません。
- 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。

12.3.5 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 12.3.3 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の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：

- ダウンロード即起動 i アプリ、および非活性化状態の待ち受けアプリケーションからは、トルカデータの検索取得を使用することはできません。
- 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。

12.4 マイプロフィール管理機能の呼び出し

マイプロフィールとは、携帯電話の所有者情報のことです。多くの機種において、ネイティブ操作ではマイプロフィール情報は、トップメニューを開いて 0 キー操作を行うことで呼び出すことができます。

i アプリでは、`com.nttdocomo.system.OwnerProfile` クラスを使用してマイプロフィール情報を参照することができます。この機能はトラステッド i アプリ、一般 i アプリのいずれでも使用することができますが、機能を使用する際には携帯電話本体の暗証番号による本人確認が行われた上で、そのユーザーによってマイプロフィール参照が許可されなければなりません。

マイプロフィール情報を参照するには、`OwnerProfile` クラスの `static` メソッド `getProfileData()` を呼び出します。このメソッドの引数には、ユーザーから参照の許可を得たいデータ項目（`OwnerProfile` クラスで定数定義されているデータ項目）のリストを渡しますが、このリストは省略することもできます。

`OwnerProfile.getProfileData()` メソッドを呼び出すと、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 **OwnerProfile.getProfileData()** 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) 続いて、携帯電話本体のパスワードを求めるダイアログが携帯電話システムにより表示されます。このダイアログに正しいパスワードを入力し、マイプロフィール管理機能に移ることをユーザーが許可すると、マイプロフィールの項目選択画面が表示されます。
- (3) マイプロフィールの項目選択画面では、マイプロフィールの全データ項目について、ユーザーが i アプリからの参照を許可するか許可しないかを個別に設定できるようになっています。i アプリが参照の許可を得たデータ項目のリストを **getProfileData()** メソッドの引数に渡した場合は、それらの項目については参照許可の状態で選択画面が開きます。ただし、ユーザーは選択画面の操作によって個別の許可を取り消すこともできます（逆に、参照不許可の項目に対し、ユーザー操作によって許可を与えることもできます）。
- (4) ユーザー操作によってマイプロフィールの項目選択が確定されると、i アプリはレジューム復帰して **getProfileData()** メソッドは **OwnerProfile** オブジェクトを返します。ユーザーがマイプロフィールの項目選択をキャンセルしても特に例外は発生せず、**getProfileData()** メソッドは **null** を返します。

上記の手順で入手した **OwnerProfile** オブジェクトに対して、**getData()** メソッドに目的のデータ項目を指定して呼び出すことで、そのデータ項目に対応する値を取得することができます。

なお、上記の処理の流れに記載した通り、最終的に **OwnerProfile** オブジェクトにどのようなデータ項目が格納されるかはユーザーの選択操作によって決定付けられます。このため、**OwnerProfile** オブジェクトに対して **getSelectedItems()** メソッドを呼び出すことで、最終的にユーザーがどのデータ項目を選択したかを示すリストを取得することができます。

なお、マイプロフィール情報を参照する 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。

注意事項：

- ダウンロード即起動 i アプリ、および非活性化状態の待ち受けアプリケーションからは、マイプロフィール情報の参照を行うことはできません。

12.5 デコメールテンプレート管理機能の呼び出し

デコメールとは、携帯電話上で簡易な HTML メール作成・送受信を行うサービスです。デコメールは 900i シリーズにてサポートされ、その後 901i シリーズにて、デコメールテンプレート（携帯電話上でユーザーがデコメールを作成する際の雛形として利用できるデータ）をコンテンツプロバイダが i モードコンテンツとして配布するための仕組みが取り入れられています。

デコメールテンプレート管理機能の呼び出しは、DoJa-5.0 プロファイルにて規定されました。この機能を使用することで、i アプリからネイティブの保存領域にデコメールテンプレートを保存したり、ネイティブに保存されているデコメールテンプレートのコンテンツデータを i アプリから参照したりすることができます。

なお、本機能を利用する 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。

12.5.1 デコメールテンプレートの新規登録

デコメールテンプレートをネイティブ保存領域に新規登録するには、`com.nttdocomo.system.DecomailTemplateStore` 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、デコメールテンプレートのデータフォーマットに従ったテキストデータ (`String` オブジェクト) を指定することで、そのデコメールテンプレートが新規登録されます。なお、デコメールテンプレートのデータフォーマットは別途 NTT ドコモより公開されます。

デコメールテンプレート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 `DecomailTemplateStore.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) デコメールテンプレートの登録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。この画面では、ユーザーは i アプリからのデコメールテンプレートの新規登録を許可、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デコメールテンプレート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) ユーザーがデコメールテンプレートの新規登録を許可するとデコメールテンプレート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、新規登録されたデコメールテンプレート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。

ユーザーがデコメールテンプレート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。

注意事項：

- ダウンロード即起動 i アプリ、および非活性化状態の待ち受けアプリケーションからはデコメールテンプレートの新規登録を行うことはできません。
- デコメールテンプレートのフォーマット規定に従っていれば、インライン画像を含むものも新規登録することができます。
- i アプリから新規登録されるデコメールテンプレートは、システム内部で整形されてから保存される可能性があります。従って、本機能で新規登録したデコメールテンプレートを以降に解説する方法で読み込んだ場合に、登録前のデータと読み込み後のデータは厳密に一致しない場合があります。

12.5.2 デコメールテンプレートの選択読み込み

デコメールテンプレート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デコメールテンプレート情報要求 (メソッド呼び出し) に対し、その都度画面に表示されるデコメールテンプレート一覧からユーザーが選択操作したデコメールテンプレートの情報を与える機能です。デコメールテンプレートの選択読み込みを行うには、`DecomailTemplateStore` クラスの static メソッド `selectEntry()` を使用します。

デコメールテンプレート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 `DecomailTemplateStore.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) デコメールテンプレートの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デコメールテンプレートを選択します。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デコメールテンプレートを決定すると、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、選択されたデコメールテンプレートに対応する `DecomailTemplateStore` オブジェクトを返します。

ユーザーがデコメールテンプレート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 `DecomailTemplateStore` オブジェクトからは、選択されたデコメールテンプレートの内容をテキストデータ (`StringBuffer` オブジェクト) として取得することができます。

また、この `DecomailTemplateStore` オブジェクトからはデコメールテンプレートのエントリ ID を取得することもでき、後述するデコメールテンプレートの ID 指定読み込みにおいて、ユーザー操作なしに再度そのデコメールテンプレートを読み込むために使用することができます。

注意事項：

- ダウンロード即起動 i アプリ、および非活性化状態の待ち受けアプリケーションからは、デコメールテンプレートの選択読み込みを使用することはできません。
- i アプリから選択読み込みを行うことができるデコメールテンプレートは、その i アプリから登録された (12.5.1 項に記載した方法で新規登録された) デコメールテンプレートのみです。

12.5.3 デコメールテンプレートの ID 指定読み込み

デコメールテンプレートのエントリ ID は、デコメールテンプレート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、そのデコメールテンプレート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デコメールテンプレートのエントリ ID を取得することができます。

- ・ デコメールテンプレート新規登録 API (`DecomailTemplateStore.addEntry()` メソッド) 呼び出しにおける返り値
- ・ デコメールテンプレート選択読み込み API (`DecomailTemplateStore.selectEntry()` メソッド) で取得した `DecomailTemplateStore` 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

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デコメールテンプレートにアクセスすることができます。デコメールテンプレートの ID 指定読み込みを行うには、`DecomailTemplateStore` クラスの static メソッド `getEntry()` を使用します。

`getEntry()` メソッドの引数にデコメールテンプレートのエントリ ID を指定して呼び出すと、そのエントリ ID に対応付けられたデコメールテンプレートを示す `DecomailTemplateStore` オブジェクトが返されます。

注意事項：

- デコメールテンプレートの ID 指定読み込みはユーザーの操作を必要としないため、非活性化状態の待ち受けアプリケーションも使用することができます。ダウンロード即起動 i アプリからは、デコメールテンプレートの ID 指定読み込みを使用することはできません。
- i アプリから ID 指定読み込みを行うことができるデコメールテンプレートは、その i アプリから登録された (12.5.1 項に記載した方法で新規登録された) デコメールテンプレートのみです。

12.6 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、903i シリーズ以降の携帯電話上で、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、`IApplication.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。pos スキーム URL の lat パラメータ(*1)に記述された値を得ることができます。 例： N12.34.56.789 （北緯 12 度 34 分 56.789 秒）
longitude	経度	経度を表すテキスト情報。pos スキーム URL の lon パラメータ(*1)に記述された値を得ることができます。 例： E12.34.56.789 （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。pos スキーム URL の geo 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： "0" 東京測地系の場合： "1"
address	住所	住所を表すテキスト情報。pos スキーム URL の address 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 `IApplication.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 (`IApplication.getLaunchType()` メソッドで取得する起動種別) は、`IApplication.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 (`IApplication.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。

12.7 データ BOX フォルダ管理機能の呼び出し

データ BOX とは、画像やメロディ、i モーション、その他のコンテンツを携帯電話内に保存するためのネイティブメモリ領域の総称です。i アプリ基本 API の機能である画像データ管理機能や映像データ管理機能は、いずれも広い意味ではデータ BOX にデータを新規登録したり、データ BOX からデータを読み込んでいることになります。

DoJa-5.0 プロファイルにて、データ BOX 内の各データ保存領域にフォルダを新規登録したり、ユーザーに特定のフォルダを選択させてそのフォルダの情報を i アプリから取得するための API がオプション API として新設されました。

なお、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。

12.7.1 フォルダの新規登録

データ BOX 内のデータ保存領域にフォルダを新規登録するには、`com.nttdocomo.system.DataBoxFolder` 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は、フォルダ作成対象としたいデータ保存領域の種別と、フォルダに付与する名称を指定します。現在のプロファイルでは、フォルダを作成可能なデータ保存領域種別として

- ・ マイピクチャフォルダ (`DataBoxFolder.FOLDER_MY_PICTURE`)
- ・ i モーションフォルダ (`DataBoxFolder.FOLDER_I_MOTION`)

の 2 つが規定されています。ただし、マイピクチャフォルダのみサポートし、i モーションフォルダをサポートしないメーカーもあります。

フォルダ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 `DataBoxFolder.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) 指定されたデータ保存領域にフォルダの登録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。この画面では、ユーザーは i アプリからのフォルダの新規登録を許可、キャンセルすることができます。

- (3) ユーザーがフォルダの新規登録を許可するとフォルダ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、新規登録されたフォルダにその携帯電話内で 1 対 1 に対応するエントリ ID を返します。

ユーザーがフォルダ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。

注意事項：

- ダウンロード即起動 i アプリ、および非活性化状態の待ち受けアプリケーションからはフォルダの新規登録を行うことはできません。
- 現在のプロファイルでは、指定したデータ保存領域の直下へのフォルダの新規登録のみサポートしています。フォルダの下にさらにサブフォルダを作成するといった操作はサポートされません。
- **addEntry()** メソッドが返すフォルダのエントリ ID を、DoJa-5.0 プロファイルで新設された **ImageStore.getEntryIds()** メソッドや **MovieStore.getEntryIds()** の引数に指定することで、そのフォルダ内に存在する参照可能な全てのデータ（画像もしくは映像）のエントリ ID を一括して取得することができます。

なお本プロファイルでは、画像や映像のデータを保存する際には、アプリケーションプログラムから保存先フォルダを指定することはできません。画像や映像の保存では、デフォルトの保存位置を変更するには必ずユーザーが登録のためのユーザーインタフェース画面を操作してフォルダを変更する必要があります。

12.7.2 フォルダのエントリ ID の選択取得

フォルダのエントリ ID の選択取得とは、アプリケーションプログラムからの要求に対し、その都度画面に表示されるフォルダ一覧（指定されたデータ保存領域に登録されているフォルダの一覧）からユーザーが選択操作したフォルダのエントリ ID を与える機能です。フォルダのエントリ ID の選択取得を行うには、**DataBoxFolder** クラスの static メソッド **selectEntryId()** を使用します。このメソッドの引数には、前項で解説したデータ保存領域種別を指定します。フォルダの新規登録と同様に、マイピクチャフォルダのみサポートし、i モーションフォルダをサポートしないメーカーもあります。

フォルダのエントリ ID の選択取得を行うと、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 **DataBoxFolder.selectEntryId()** 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) フォルダの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 アプリにエントリ ID を取得させるフォルダを選択します。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フォルダを決定すると、i アプリはレジュームして **selectEntryId()** 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、選択されたフォルダに対応するエントリ ID を返します。

ユーザーがフォルダ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-1 を返します。

注意事項：

- ダウンロード即起動 i アプリ、および非活性化状態の待ち受けアプリケーションからはフォルダのエントリ ID の選択取得を行うことはできません。
- 現在のプロファイルでは、本機能は指定したデータ保存領域の直下のフォルダのみサポートしています。それより深い階層のサブフォルダを選択取得するといった操作はサポートされません。

- **selectEntryId()**メソッドが返すフォルダのエントリ ID を、DoJa-5.0 プロファイルで新設された **ImageStore.getEntryIds()**メソッドや **MovieStore.getEntryIds()**の引数に指定することで、そのフォルダ内に存在する参照可能な全てのデータ（画像もしくは映像）のエントリ ID を一括して取得することができます。

なお本プロファイルでは、画像や映像のデータを保存する際には、アプリケーションプログラムから保存先フォルダを指定することはできません。画像や映像の保存では、デフォルトの保存位置を変更するには必ずユーザーが登録のためのユーザーインタフェース画面を操作してフォルダを変更する必要があります。

第 13 章

ラジオチューナー制御

メーカーによっては、i アプリから制御可能なラジオチューナーを搭載している場合があります。i アプリからのラジオチューナー制御をサポートする携帯電話では、i アプリからラジオの電源オンオフや選局、音量などの制御を行うことができます。

本項では、ラジオチューナーを制御するための API について解説します。

ラジオチューナー機能は、`com.nttdocomo.opt.device.RadioTuner` クラスで提供されます。この機能を使用する i アプリは、まず最初に `RadioTuner` クラスの static メソッド `getRadioTuner()` を使用して `RadioTuner` オブジェクトを取得します。

`RadioTuner` オブジェクトには以下のような項目の設定を行うことができます。

- ・ チューナータイプ (AM、FM などのバンド切り替え)
- ・ 周波数
- ・ ボリューム
- ・ ステレオモード
- ・ 音の出力先 (スピーカーもしくはイヤホン)

上記の項目はいずれも、`RadioTuner` オブジェクトを取得した時点で何らかの状態に初期設定されています。しかし各項目の初期設定がどのような状態であるかはメーカーにより異なります。

上記項目の設定を適切に行った後に `RadioTuner.turnOn()` メソッドを呼び出すことで、チューナー電源をオンにしてラジオ音鳴動状態にすることができます。また、`RadioTuner.turnOff()` メソッドでチューナー電源をオフ (ラジオ音停止状態) にすることができます。現在の電源状態は `RadioTuner.isTurnedOn()` メソッドで調べることができます。

以下に、`RadioTuner` オブジェクトの各設定項目について解説します。なお、各項目の `set` 系メソッドに対しては、現在の設定値を取得するための `set` 系メソッドも用意されています。

- ・ チューナータイプ
`setTunerType()` メソッドを使用して、AM、FM などのバンド切り替えを行います。設定値として、現在のプロファイルでは定数 `TUNERTYPE_AM` (AM)、`TUNERTYPE_FM` (FM) の 2 つが用意されています。
その端末でサポートされるチューナータイプは、`getSupportedTunerTypes()` で取得することができます。
- ・ 周波数
`setFrequency()` メソッドを使用して、各チューナータイプに対応する周波数の設定を行います。周波数は kHz 単位の整数値で指定します (FM で 80.2MHz を設定する場合は 80200 を指定します)。

各チューナータイプで指定可能な周波数の範囲は、`getMinFrequency()`、`getMaxFrequency()`の各メソッドを、チューナータイプを指定して呼び出すことで調べることができます。

なお、チューナータイプが FM の場合は、FM 波として指定可能な範囲の周波数の他にテレビ 1～3ch の各音声波を示す定数 `FREQUENCY_TV1`、`FREQUENCY_TV2`、`FREQUENCY_TV3` を指定することができます。

- ・ ボリューム
`setVolume()`メソッドを使用して、ラジオの音量のボリュームを設定します。指定可能な値は 0（無音）から 100（最大音量）までの範囲となります。
- ・ ステレオモード
`setStereoMode()`メソッドを使用して、ステレオモードの設定を行います。設定値として定数 `MODE_MONAURAL`（モノラル）と `MODE_AUTO`（自動切換え）の 2 つが用意されています。
- ・ 音の出力先
`setOutput()`メソッドを使用して、音の出力先の設定を行います。設定値として以下の定数が用意されています。
 - `OUT_SPEAKER` : イヤホン装着有無に関係なく、スピーカーから音出力されます。
 - `OUT_EARPHONE` : イヤホン装着時のみ、イヤホンから音出力されます。
 - `OUT_AUTO` : イヤホン装着時はイヤホンから、非装着時はスピーカーから音出力されます。

なお周波数については、特定の周波数をチューナーに設定するだけでなく、その場所で受信可能な周波数を現在の周波数の位置からシーク（サーチ）することもできます。

周波数のシークを行うには、チューナー電源をオンにした上で `RadioTuner.seekFrequency()` メソッドを使用します。このメソッドは、以下のいずれかの結果を得るまでブロックします。

- a) シークに成功した（受信可能な周波数を見つけることができた）
見つけた周波数がチューナーに設定され、戻り値としてその周波数が返されます。
- b) シークに失敗した（バンド全体をシークして受信可能な周波数を見つけられなかった）
戻り値として -1 が返されます。チューナーにどのような周波数が設定されるかはメーカーにより異なります。
- c) シークが中断された
例外が発生します。チューナーにどのような周波数が設定されるかはメーカーにより異なります。
- d) その他の何らかの理由によりシークが異常終了した
例外が発生します。チューナーにどのような周波数が設定されるかはメーカーにより異なります。

`seekFrequency()`メソッドによるシークは、他のスレッドから `RadioTuner` オブジェクトに対して `interruptSeeking()`メソッド、もしくは `turnOff()`メソッドを呼び出すことで中断することができます（この場合、上記 c のケースに該当します）。

なお、テレビ 1～3ch はシークの対象とはなりません。

注意事項：

- ラジオチューナーの電源をオンにした状態で i アプリがサスペンドしても、ラジオ音は停止せず鳴り続けます。ラジオチューナーも i アプリも電池消費の多い機能であるため、ラジオ利用中に i アプリをサスペンドさせて電池消費を少なくするといったことが実現できるよう、`RadioTuner` クラスには意図的にサスペンドを発生させる `savePower()`メソッドが用意されています。

また、ラジオチューナーの電源をオンにした状態で待ち受けアプリケーションが非活性化状態や休眠状態に入った場合にも、同様にラジオ音は停止せず鳴り続けます。

- シーク実行中にチューナータイプの変更（`setTunerType()`）や周波数の設定（`setFrequency()`）を行うことはできません。また、シーク実行中にこれら以外の設定項目を設定できるかどうかはメーカーにより異なります。
- 待ち受け実行されている i アプリでは、シークは活性化状態でのみ実行することができます。
- `seekFrequency()` メソッドは、現在チューナーに設定されている周波数の位置から引数に指定された方向にシークを行いますが、シークの結果バンドの上端（または下端）まで到達するとバンドの下端（または上端）に戻ってサイクリックにシークを継続します。シーク開始地点からバンド上端（または下端）を経由して再度シーク開始地点まで到達しても受信可能な周波数が見つからなかった場合（バンド全域に渡り受信可能な周波数なかった場合）にシーク失敗と判断されます。
- 携帯電話上でサウンドや音声を取り扱う機能が使用されている状態（通話中の場合や i アプリからサウンド等の再生が行われている場合など）では、ラジオ音を鳴らすことはできません。また、ラジオ音を鳴らしている際にこれらの機能が使用された場合は、ラジオ音は停止しラジオチューナーは電源オフの状態になります。

【DoJa-5.0】

ラジオチューナー制御 API は、DoJa-5.0 プロファイルにて追加されました。

第 14 章

カメラ拡張

DoJa-3.0 プロファイル以降、カメラ制御機能 (`com.nttdocomo.device.Camera` クラス) が i アプリ基本 API としてサポートされるようになりました。i アプリオプション API のカメラ拡張では、カメラ制御機能に関するオプションの属性を規定しています。カメラ拡張は、`com.nttdocomo.opt.device.Camera2` クラスで提供されます。

なお、現在のプロファイルのカメラ拡張では、規定されている属性は撮影画像のエンコード形式のみです。

14.1 撮影画像のエンコード形式

カメラ制御機能で撮影する画像のエンコード形式を指定します。画像のエンコード形式を変更するには、`com.nttdocomo.device.Camera` クラスの `setAttribute()` メソッドを使用して、属性 `Camera2.DEV_IMAGE_ENCODER` に以下の属性値を設定します。

Camera2.ATTR_DEFAULT_IMAGE_ENCODER

デフォルトのエンコード形式です。一度変更したエンコード形式を元に戻す際に使用します。

Camera2.ATTR_JPEG_IMAGE_ENCODER

JPEG (JFIF・Baseline エンコーディング) を示すエンコード形式です。一般に、`ATTR_DEFAULT_IMAGE_ENCODING` を指定した場合と同じ動作となります。

Camera2.ATTR_RAW_IMAGE_ENCODER

内部形式のままエンコードしないことを示すエンコード形式です。JPEG のような圧縮処理が行われないため、ブロックノイズが入らず高品質な画像を取得することができます。このエンコード形式を指定した場合、`Camera.getInputStream()` メソッドを使用して画像データをバイトイメージ形式で取り出すことはできません。このエンコード形式を指定した場合の画像の取り出しは、`Camera.getImage()` メソッドを使用するのみ可能です。

注意事項：

- デフォルト以外のエンコード形式は、撮影画像サイズや連写モードの設定により使用できない場合があります。`Camera.setAttribute()` メソッドでエンコード形式を設定した後で `Camera.getAttribute()` メソッドでエンコード形式を取得したときに、セットした属性値が反映されていればそのエンコード形式は利用可能です。セットした属性値が反映されておらず、`Camera2.ATTR_DEFAULT_IMAGE_ENCODER` が返された場合は、そのエンコード形式は利用できません。
- この機能を、DoJa-2.0 プロファイルとの互換のため残されている `com.nttdocomo.opt.ui.Camera` クラスに対して使用することはできません。

第 15 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）が行えると同時に、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、DoJa-2.0 プロファイルで採用された OBEX 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。

OBEX 外部接続：

OBEX 外部接続は、i アプリから通信路を開設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。

15.1 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

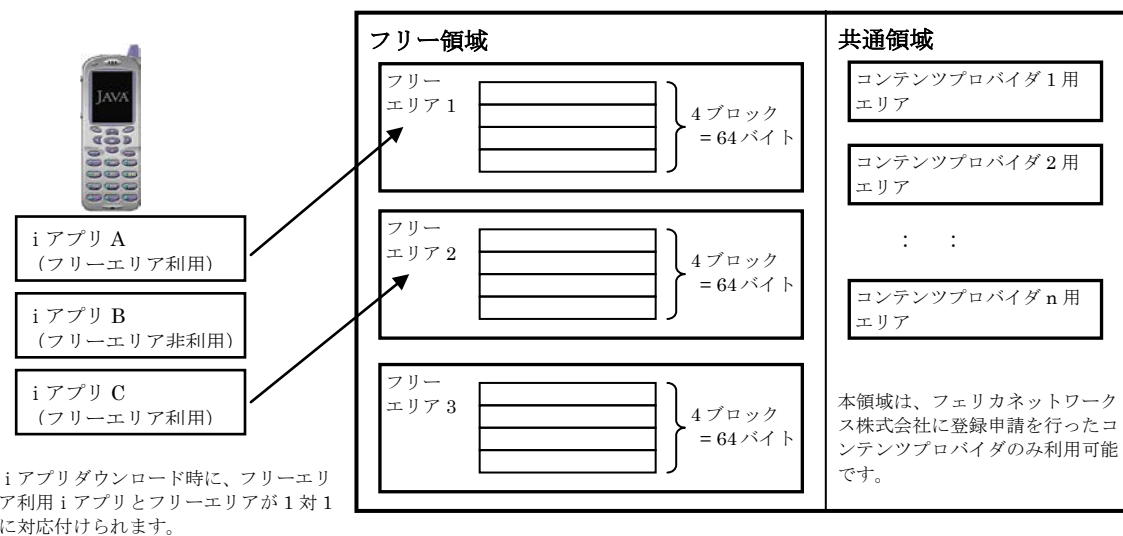
フェリカネットワークス株式会社からエリアの割り当てを受け、i モードサーバーに登録された i アプリにのみ許可されます。許可を受けていない i アプリが共通領域を使用することはできません（FeliCa チップ上の共通領域へのエリア登録は i アプリからの API 呼び出しにより行われますが、許可を受けていない i アプリはこの API を呼び出すことはできません）。

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、15.2 項に記載する 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。



15.2 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.nttdocomo.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。
	static void open()	FeliCa チップをオープンする。
	static FreeArea getFreeArea()	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。
	static void close()	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。
	byte[] read()	全てのブロックのデータを読み込む。
	byte[] read(int[] index)	指定したブロックのデータを読み込む。
	void write(byte[] data)	全てのブロックにデータを書き込む。
	void write(int[] index, byte[] data)	指定したブロックにデータを書き込む。
	byte[] getIDm()	フリー領域の IDm を取得する。(*1)
	void setReset(boolean flag)	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。(*1)
	boolean isReset()	フリーエリアアクセス後に、FeliCa チップをリセットする設定となっているかどうか調べる。(*1)
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。
	int getID()	エラー種別を返す。
	int getType()	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。 (*1) この例外クラスには、独自のメソッドはありません。

(*1) これらのクラス、メソッドは、DoJa-5.0 プロファイルにて追加されました。

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 からの Felica チップアクセスと i アプリからの Felica チップアクセスは排他的にのみ行うことができます。i アプリからの Felica チップアクセス時 (**Felica.open()** メソッド実行から **Felica.close()** メソッド実行までの区間) には、外部 R/W から Felica チップにアクセスしても反応が得られません。また外部 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。
- ダウンロード即起動 i アプリ、および非活性化状態の待ち受けアプリケーションからは Felica チップへアクセスすることはできません。
- 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 15.3 項を参照してください。
- 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。

【DoJa-5.0】

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。DoJa-5.0 プロファイルでは、この PIN が解除されている区間を極力短くすることができるように、**FreeArea** クラスに **setReset()** メソッドが追加されています。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。

また、903i シリーズ以降の携帯電話で採用されている新しいバージョンのモバイル Felica IC チップ (モバイル Felica OS Version 2.0 搭載の Felica チップ) では、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るようになっています。この変更に伴い、外部 R/W から Felica チップにアクセスしているタイミングで i アプリがフリーエリアにアクセスした際に発生する例外 **PortBusyException** が DoJa-5.0 プロファイルにて新設されました。

15.3 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、15.4 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（15.4 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：

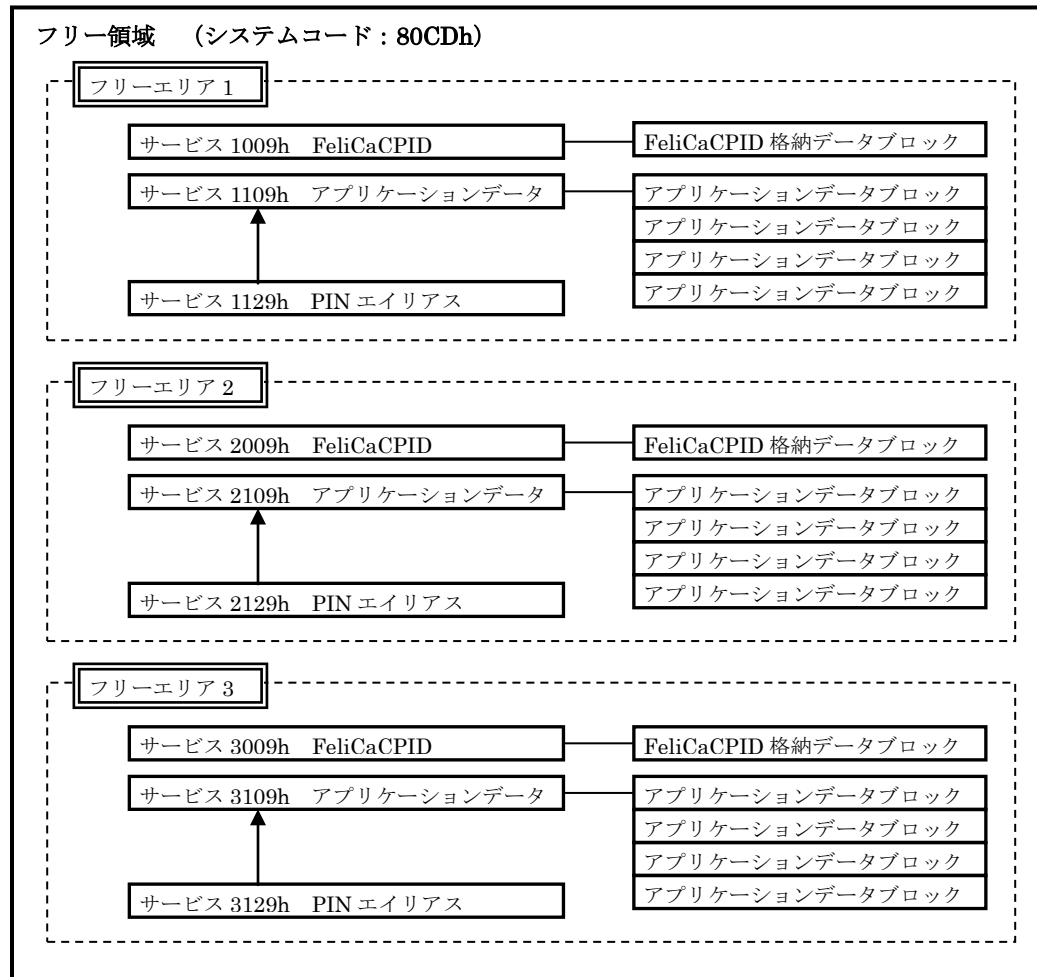
- これらのキーによりフリーエリアを利用するとして携帯電話にダウンロードされた i アプリを、キーを削除してフリーエリアを利用しない i アプリにバージョンアップすることはできません。この場合は i アプリのバージョンアップではなく、いったん i アプリを削除して再度ダウンロードし直す必要があります。なお、FeliCaCPID キーおよび FeliCaPIN キーのいずれも、バージョンアップ時に値を変更することは可能です。
- これらのキーが指定されている i アプリをバージョンアップする際、それまでフリーエリアに格納されていたアプリケーションデータはそのまま引き継がれます。また、これらのキーが指定されている 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。

15.4 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 사용됩니다。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。

【DoJa-5.0】

DoJa-5.0 プロファイルの携帯電話に搭載される FeliCa チップ（モバイル FeliCa OS Version 2.0 搭載の FeliCa チップ）では、以前のバージョンの FeliCa チップと比較して PIN に関する領域上の構造が変更されています。しかし、外部 R/W が PIN を解除するために指定するサービスコードの値については従来のものから変更はありません。

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）することが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。

15.5 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。

【DoJa-5.0】

DoJa-5.0 プロファイル対応の 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）。

注：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。

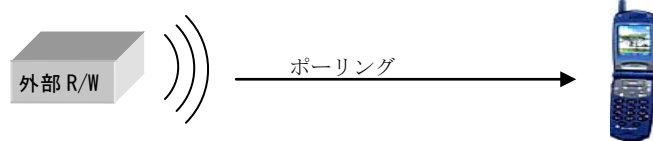
保護機構の動作契機は、UIM カードを抜いたり差し換えた後の最初の電源投入時、もしくは携帯電話の FeliCa ロック機能をオフにするユーザ操作が行われた時となります。

15.6 外部 R/W からの指示による携帯電話内の i アプリ起動

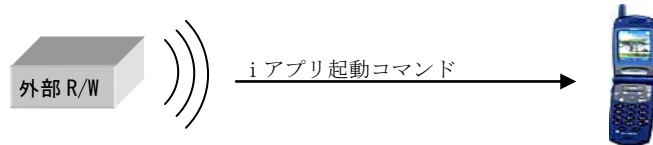
FeliCa チップ搭載 i モード携帯電話の中にはオプション機能として、外部 R/W から携帯電話上の FeliCa チップに所定の信号（コマンド）を送信することにより、携帯電話内にダウンロードされている特定の i アプリを起動する機能を持つものがあります。

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／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に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、IApplication.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、OBEX 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。
- 携帯電話上で待ち受けアプリケーションが非活性化状態または休眠状態で実行されている場合に外部 R/W から i アプリの起動が指示された場合、システムは待ち受けアプリケーションを自動的に終了して起動対象 i アプリを起動します。携帯電話上で待ち受けアプリケーションが活性化状態で実行されている場合や i アプリが通常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 (**IApplication.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と、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 とするようになしてください。
- フリーエリアを利用する i アプリを起動する際、および DoJa-5.0 プロファイル以降の携帯電話において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。

15.7 FeliCa チップによるアドホック通信

903i シリーズ以降の携帯電話で採用されている新しいバージョンのモバイル 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ますようになります。DoJa-5.0 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。

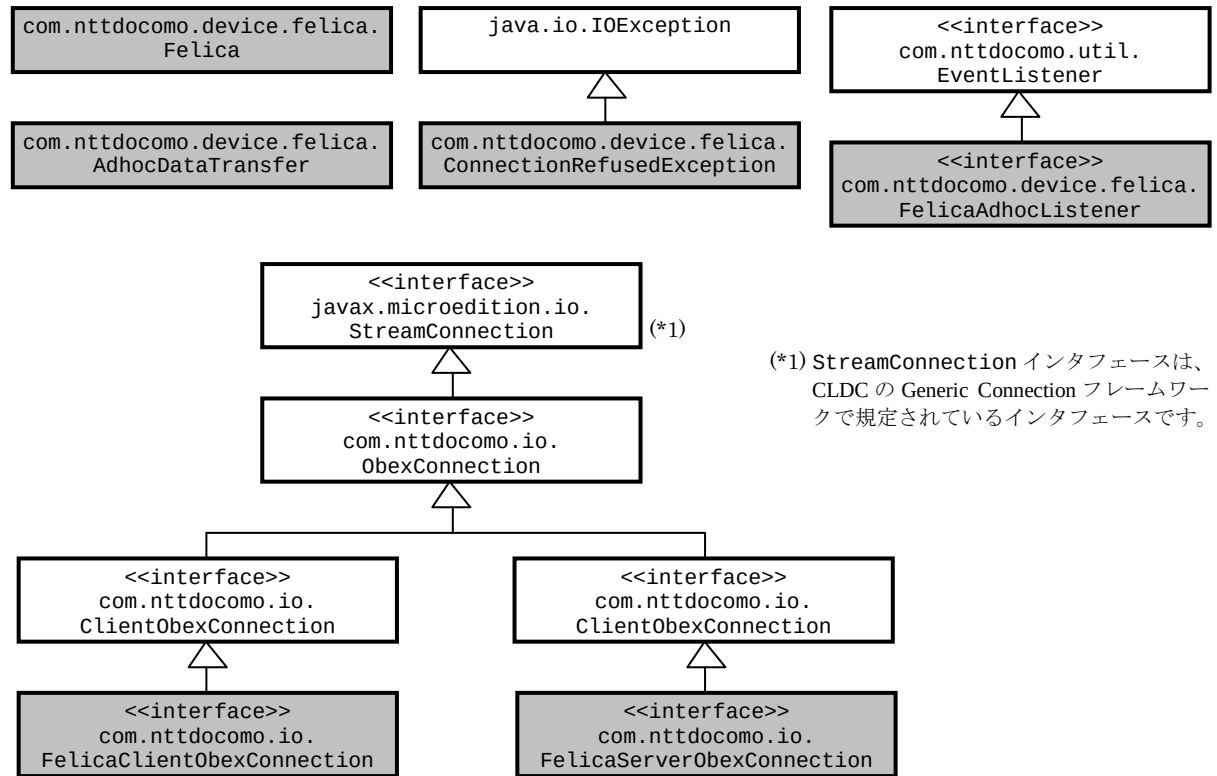
注意事項：

- 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。

15.7.1 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・インタフェー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 アプリで使用するべきクラス・インタフェースが異なります。

com.nttdocomo.device.felica パッケージ：

クラス・インタフェース	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。
FelicaAdhocListener	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の通知を受けるために実装すべきインタフェースです。サーバー側 i アプリでは、i アプリメインクラス (IApplication 継承クラス) にてこのインタフェースを実装する必要があります。

com.nttdocomo.io パッケージ:

クラス・インタフェース	機能概要
FelicaClientObex Connection	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に使用するインタフェースです。このインタフェースの実装オブジェクトは、Generic Connection フレームワークの Connector クラスを使用して取得します。
FelicaServerObex Connection	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に使用するインタフェースです。このインタフェースの実装オブジェクトは、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、データ送受信用の 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。

15.7.2 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た上でリンク開設要求があった旨の通知が行われます。

- ・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 アドホック通信のサーバー側 i アプリでは、i アプリメインクラス（`IApplication` 継承クラス）にて `FelicaAdhocListener` インタフェースを実装しなければなりません。i アプリ実行環境はこのインタフェースに定義されたリスナーメソッド `requestReceived()` を通じて、クライアント側 i アプリから通信リンク接続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。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い場合、i アプリ実行環境はその i アプリを起動した上で、リスナーメソッドへの通知を行います。

`requestReceived()` のパラメータとして渡される `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。

このリスナーメソッドは `boolean` 型であり、アプリケーションプログラムが値 `true` を返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値 `false` を返した場合や、キャッチされない例外によってこのメソッドが終了してしまった場合には、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。
- 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。
- 非活性化状態の待ち受けアプリケーションは FeliCa チップをオープンできないため、通信リンクの確立を行うことはできません。また、活性化状態の待ち受けアプリケーションが通信リンクの確立を行っている際に非活性化状態に遷移しようとする、FeliCa アドホック通信は失敗します。
- 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。
- 通信リンク確立時には、サーバー側 i アプリの `requestReceived()` メソッドが戻り値 `true` で復帰したことを受けて、クライアント i アプリ側の `AdhocDataTransfer.setup()` メソッドが復帰することに注意してください。特に、サーバー側 i アプリの `requestReceived()` メソッド内に次項で解説するデータ送受信処理を実装した場合、サーバー側 i アプリのデータ送受信開始待ちとクライアント側 i アプリの `setup()` メソッド復帰待ちがデッドロックを引き起こします。サーバー側 i アプリでのデータ送受信処理は `requestReceived()` メソッド内には直接実装せず、別スレッドとして実装したものを起動するなどにより `requestReceived()` メソッド復帰と並列に動作できるようにしてください。
- サーバー側 i アプリが起動されていない状態で通信リンク確立要求を受けた場合、システムは該当のサーバー側 i アプリを起動した後にリスナーメソッドを呼び出します。その際のリスナーメソッドの呼び出しは、i アプリメインクラスのインスタンス化後（コンストラクタの処理完了後）に行われることは保証されていますが、i アプリ開始メソッド（i アプリメインクラスの `start()` メソッド）の呼び出しとの前後関係は保証されません。また、このようにして起動された i アプリの起動種別（`IApplication.getLaunchType()` メソッドで取得する起動種別）は、`IApplication.LAUNCHED_FROM_FELICA_ADHOC` となります。
- 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、15.6 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。

15.7.3 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` インタフェースの実装オブジェクト) を取得します。`FelicaClientObexConnection` インタフェースは `ClientObexConnection` インタフェースのサブインタフェースですが、追加のメソッドやフィールドはありません。このコネクションオブジェクトの使用方法は、一部の例外規定にデバイス起因の差異がある点を除き、原則的に `ClientObexConnection` 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設します。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など例外が発生します。

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` インタフェースの実装オブジェクト) を取得します。`FelicaServerObexConnection` インタフェースは `ServerObexConnection` インタフェースのサブインタフェースですが、追加のメソッドやフィールドはありません。このコネクションオブジェクトの使用方法は、一部の例外規定にデバイス起因の差異がある点を除き、原則的に `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 `getContentLength()` 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例が発生します。

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 16 章

UART 接続によるシリアル通信

本章では、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、J2ME CLDC の 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
baudrate	ボーレートを baudrate=<値> 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。
databitlen	データビット長を databitlen=<値> 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。
parity	パリティの設定方法を parity=<識別子> 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。
stopbitlen	ストップビット長を stopbitlen=<値> 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。
flowctrl	フローコントロールの指定を flowctrl=<識別子> 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したり待ち受け休眠状態に遷移したりすると、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。

【DoJa-4.1】

DoJa-4.1 プロファイルにおける **ConnectionException** クラスでは、UART を使用して通信を行っている際に物理的に接続が絶たれた（接続ケーブルが外れるなど）ことを示す例外ステータス **UART_DISCONNECTED** が追加されています。

第 17 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：

- ダウンロード即起動 i アプリは、本章で解説される機能を使用することはできません。また、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。

17.1 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.nttdocomo.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.nttdocomo.device` パッケージで提供されます。

com.nttdocomo.fs パッケージ

`com.nttdocomo.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。

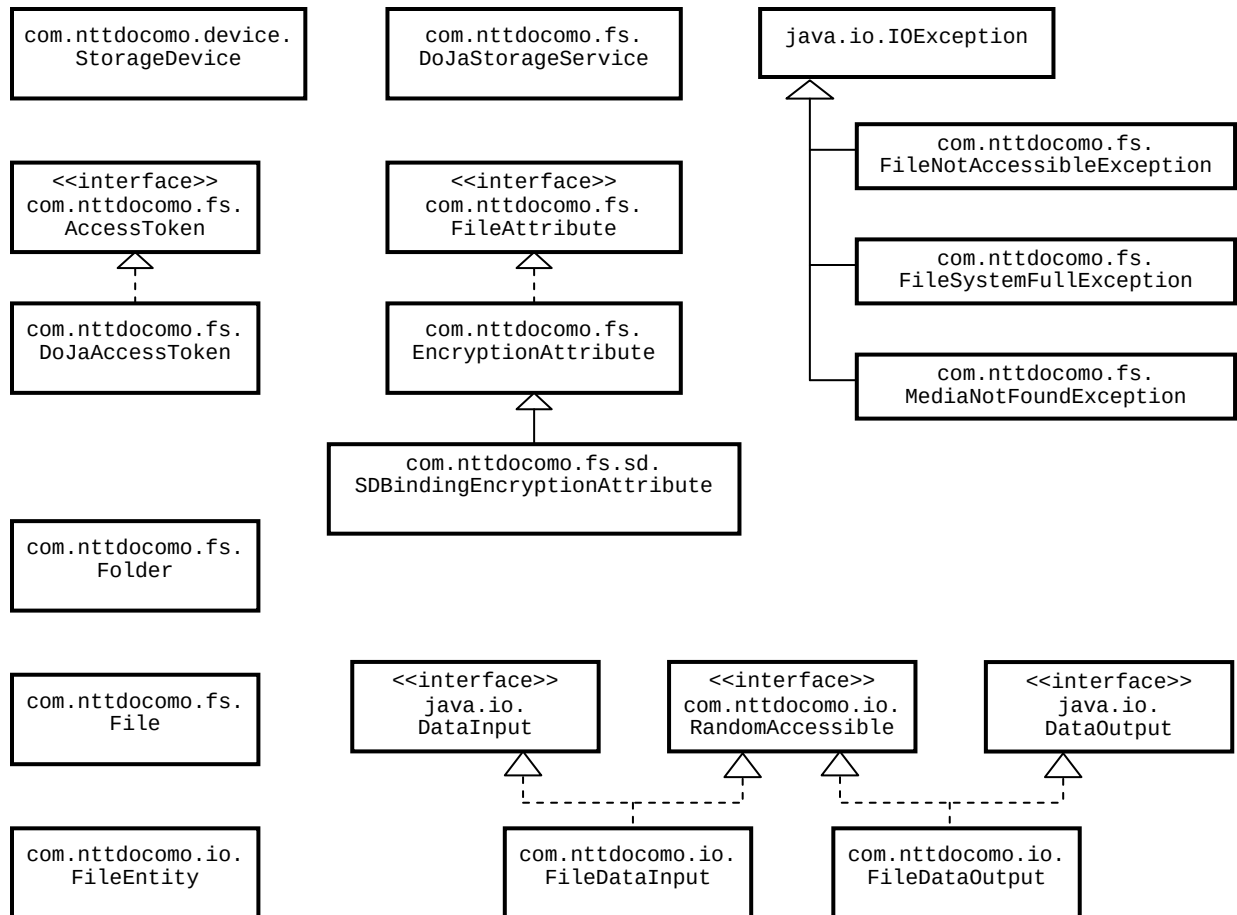
com.nttdocomo.fs 配下サブパッケージ

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.nttdocomo.fs.sd` パッケージが規定されています。

com.nttdocomo.io パッケージ

com.nttdocomo.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.nttdocomo.device パッケージ

クラス、インタフェース	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.nttdocomo.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。
DoJaAccessToken	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 DoJaStorageService 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。
DoJaStorageService	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 DoJaAccessToken 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.nttdocomo.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.nttdocomo.fs.sd パッケージ

クラス、インタフェース	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.nttdocomo.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.nttdocomo.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。

17.2 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 **DoJaStorageService** 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 (**DoJaAccessToken** 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。

17.2.1 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 **DoJaAccessToken** クラスが規定されています。

DoJaAccessToken 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、**DoJaStorageService** 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、**DoJaAccessToken** 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。
- **ACCESS_SERIES**： データ保存時と同じシリーズの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。

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 `DoJaStorageService` 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 `CPName` キーを設定する必要があります。`CPName` キーについては 17.4 項を参照してください。なお、この規定は DoJa-4.1 プロファイルの中でも 902iS シリーズ以降に規定されました。`SHARE_CONTENTS_PROVIDER` を使用する i アプリは、902i シリーズの機種では ADF に `CPName` キーが設定されていなくても正常に動作しますが、902iS シリーズ以降の機種では正常に動作しないため注意が必要です。

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 = DoJaAccessToken.ACCESS_UIM | DoJaAccessToken.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 = DoJaStorageService.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
    DoJaAccessToken dat = DoJaStorageService.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( dat );
    ...    // 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である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。
```

17.2.2 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。

17.2.3 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。CLDC 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)。

17.3 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。

`FileNotFoundException`

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、例えば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。

17.4 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（`DoJaAccessToken` 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 902i シリーズには存在しないキーですが、902iS シリーズ以降では **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。902iS シリーズ以降では、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（902i シリーズでは、このキーがなくても **SHARE_CONTENTS_PROVIDER** を使用することができます）。

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

17.5 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

DoJa-5.0 プロファイルにて、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 が追加されました。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.nttdocomo.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 **MediaImage** 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。

17.6 メーカー実装に起因する外部メモリ利用上の注意事項

17.6.1 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

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。

17.6.2 ファイル名に関する注意事項

外部メモリで使用す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太文字・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太文字・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、それらには太文字・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、常に全て太文字で指定する、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、ファイル名の太文字・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太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 18 章

デジタルテレビ制御

DoJa-4.0 プロファイルにおいて、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追加されました。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

【DoJa-5.0】

テレビ番組の録画予約の登録機能は、DoJa-5.0 プロファイルにて追加で規定されました。ただし、録画予約登録機能は、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。

18.1 DTV 機能の連携起動

本機能に対応した携帯電話では、`IApplication.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 `IApplication.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：

- `IApplication.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
```

- **IApplication.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。

【DoJa-5.0】

DTV 機能の連携起動の際に指定する、系列識別番号を表す引数は、DoJa-5.0 プロファイルにて追加で規定されました。DoJa-4.x 以前のプロファイルでは、DTV 機能を連携起動する際の引数には物理チャンネル番号とサービス識別番号のみ指定することができます。

18.2 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.nttdocomo.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.nttdocomo.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・録画の開始時刻、および終了時刻を表す **Calendar** オブジェクト（視聴予約の場合は終了時刻は設定不要であり、設定しても無視されます）
- ・ 視聴・録画の予約について繰り返しの有無やサイクルを指定する日時指定タイプ（日時指定タイプには、**com.nttdocomo.util.ScheduleDate** クラスに定義されている定数を指定します。省略時は繰り返しのない1回限りの予約となります）

- （*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（**FREQUENCY_NONE** と **AFFILIATION_ID_NONE** は、DoJa-5.0 プロファイルで新設された定数です）。

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：

- ダウンロード即起動 i アプリ、および非活性化状態の待ち受けアプリケーションからは視聴・録画予約の新規登録を行うことはできません。
- 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。
- 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の振舞いは、メーカーにより異なります。

【DoJa-5.0】

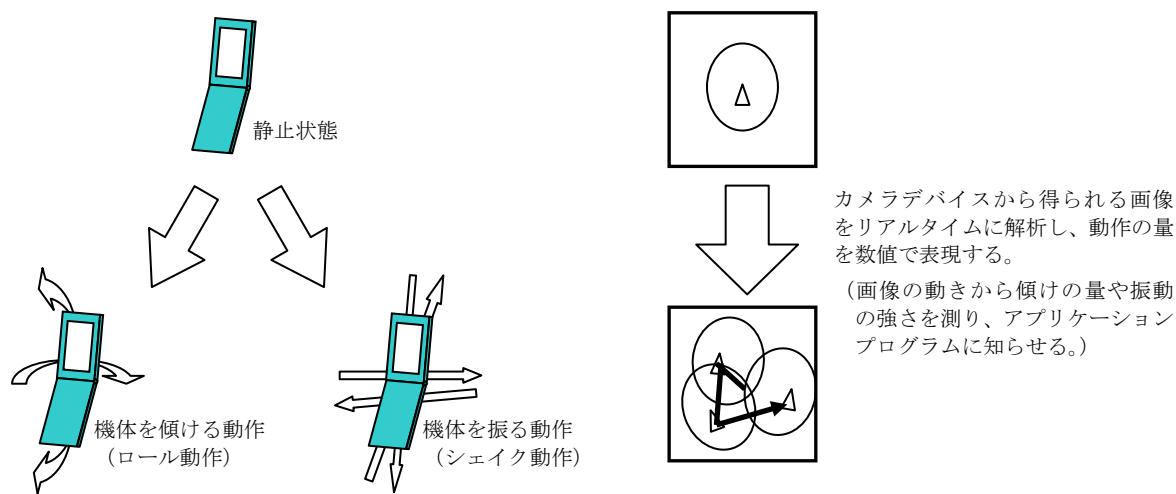
予約タイプ（`DTVSchedule` クラスの `TYPE` 定数）、予約タイプを引数に使用する API、および `DTVScheduleParameter` クラスは、録画予約登録のサポートに伴い DoJa-5.0 プロファイルにて新設されました。DoJa-4.x 以前のプロファイルでは、これらの API は使用することはできません。

なお、予約タイプを指定しない従来の API も、互換性保持のため DoJa-5.0 以降も残されています（従来同様、それら互換 API では視聴予約に対する操作が行われます）。しかしそれら互換 API の使用は、DoJa-5.0 プロファイル以降では推奨されません。

第 19 章

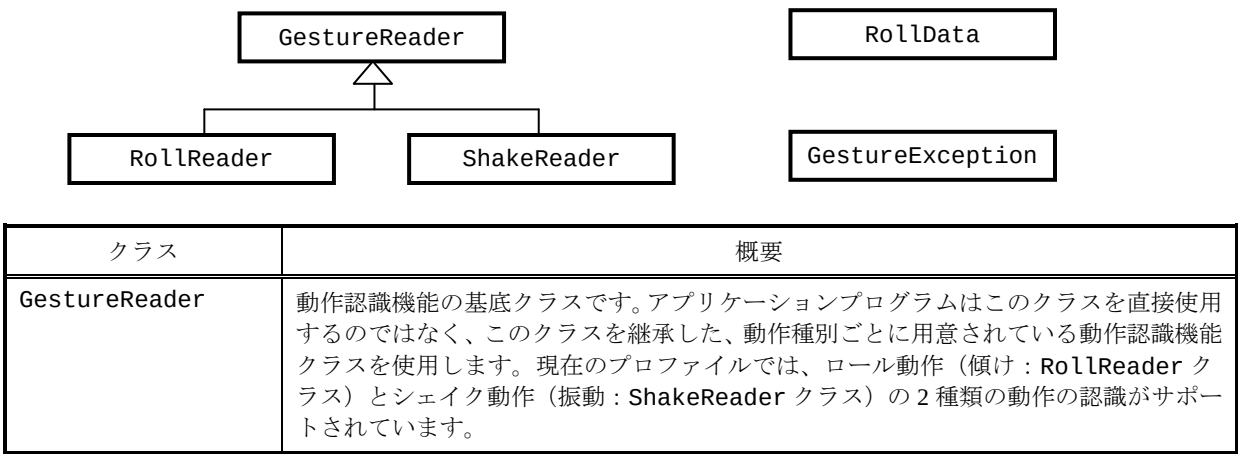
動作認識

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.nttdocomo.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.nttdocomo.device.gesturereader` パッケージ



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、例外の大まか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いきます。

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や待ち受け休眠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。

【DoJa-5.0】

動作認識機能は、DoJa-5.0 プロファイルにて追加されました。

第 20 章

電子コンパス

DoJa-5.0 プロファイル以降の携帯電話では、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、DoJa-5.0 プロファイル以降では電子コンパスを扱うための `com.nttdocomo.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.nttdocomo.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での角度を表す `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、`long` 型による表現と `double` 型による表現のどちらかを選ぶことができます。

`long` 型による固定小数点形式では、角度を、`long` 型 64 ビットのうち下位 31 ビットを小数部とする整数値として表します。従ってこの形式では、`long` 値と実際の角度の関係は以下の通りとなります。

DEGREE 単位の <code>long</code> 値	<code>long</code> 値の表現する角度
1	1/(2 ³¹)度
1<<31	1 度
90<<31	90 度

`double` 型による表現では、その値がそのまま単位が°である角度を表します。

`long` 型の表現値を取得するには `getFixedPointNumber()` メソッドを、`double` 型の表現値を取得するには `getFloatingPointNumber()` 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。

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することを推奨します。

【DoJa-5.0】

電子コンパス機能は、DoJa-5.0 プロファイルにて追加されました。

第 21 章

加速度センサ

加速度センサ機能は、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

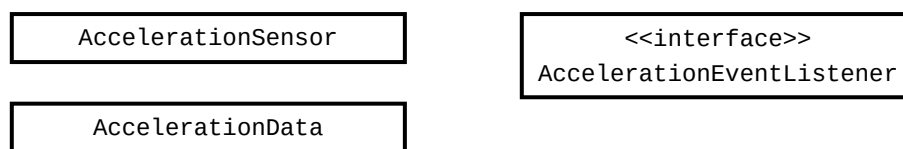
加速度センサ機能をサポートする機種においても、上記オプション機能をサポートするかどうかはメーカーの設計に委ねられています。

なお加速度センサ 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 API ではその機種における加速度センサ機能の機能・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。

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

`com.nttdocomo.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ー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。以下に、加速度センサ API の利用方法について解説します。

(1) 加速度センサ機能の機能・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。
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。
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。

`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）

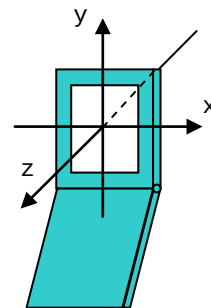
機体のピッチ角：`getPitch()`
単位 度

機体のロール角：`getRoll()`
単位 度

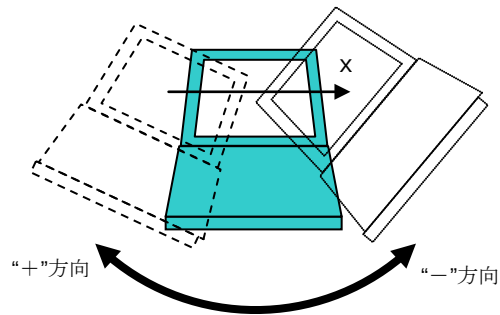
メインスクリーンの向き：`getScreenOrientation()`
単位 度（ただし返却値は 0、90、180、270 のいずれかに丸められます）

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。

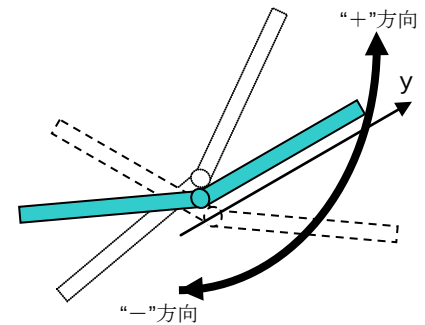
x, y, z の各方向：
 x ：メインスクリーン上を左から右に向かう方向
 y ：メインスクリーン上を下から上に向かう方向
 z ：メインスクリーンの奥から手前に向かう方向



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の能力により、-180 度～180 度の範囲を計測できる機種と -90 度～90 度の範囲を計測できる機種があります。

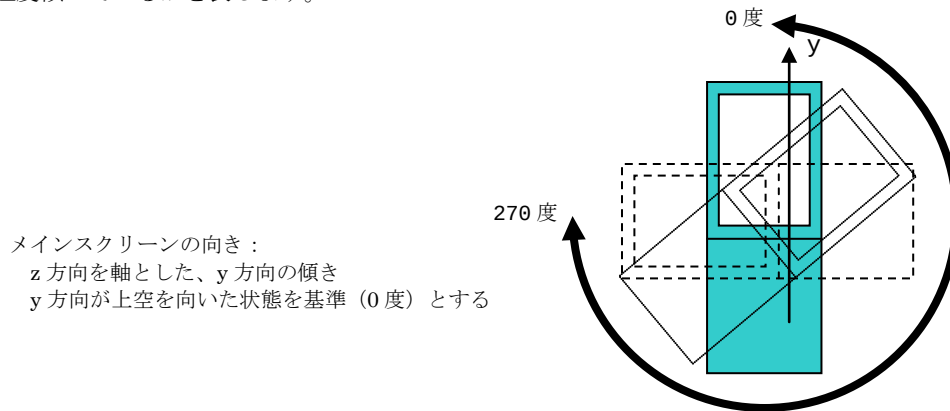


ロール角：
 地面に対する x 方向の傾き
 x 方向と地面が平行な状態を基準（0 度）とする



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、加速度の変化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`) 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、加速度の変化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。

またダブルタップイベントでは、ユーザ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。

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、`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や待ち受け休眠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。
- 周期計測を行っている状況では、ワンショット計測を行ったり、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検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ったり周期計測を新たに開始することは可能です）。

なお、905iシリーズの世代まではワンショット計測、周期計測、加速度変化のイベント検知の各機能を同時に利用することはできません。つまり、周期計測が開始されている状態や加速度変化のイベント検知が開始されている状態では、他の加速度計測機能を使用することはできません。

- `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。

【DoJa-5.0】 【DoJa-5.1】

加速度センサ機能は、DoJa-5.0 プロファイルにて追加されました。なお DoJa-5.0 プロファイルでは、加速度の変化に関するイベントとしてはメインスクリーンの向きの変化のみ規定されています。ダブルタップイベントは、DoJa-5.1 プロファイル世代のうち 906i シリーズにて追加されました。

第 22 章

ユーティリティ API

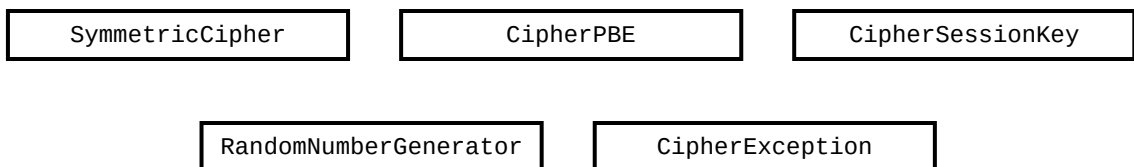
この章では、これまでに解説した機能カテゴリに含まれない、オプションのユーティリティ機能について解説します。

22.1 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.nttdocomo.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。

【DoJa-5.1】

暗号化機能は、DoJa-5.1 プロファイルにて追加されました。

第 23 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）や脈拍計などのヘルスケア関連機能を搭載しているものがあります。DoJa-5.1 プロファイルでは、i アプリからこれら機能にアクセスするためのオプション API が規定されました。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が保持している情報を参照して利用したり、i アプリから脈拍計機能を呼び出して脈拍の測定を行ったりすることができます。

23.1 歩数計機能

歩数計機能の API は、`com.nttdocomo.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。

(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.getDate()` 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。

【DoJa-5.1】

歩数計機能をサポートする機種の中には、携帯電話ネイティブの歩数計のユーザー設定項目を i アプリから設定するための、`PedometerSettings` クラスをサポートするものがあります。i アプリから設定可能なユーザー設定項目には、身長と体重があります。携帯電話ネイティブの歩数計は、ユーザー設定項目の内容と歩数データを使用して、消費カロリーや燃焼脂肪量などの参考情報を推定します。

i アプリからユーザー設定項目を設定するには、`PedometerSettings` クラスの `setBodyHeight()` メソッド(身長:単位 mm)および `setBodyWeight()` メソッド(体重:単位 g)を使用します。また、`getAttribute()` メソッドを使用して、身長や体重に設定可能な最大値、最小値を調べることもできます。なお、携帯電話ネイティブの歩数計に設定されているユーザー設定項目の内容を、i アプリから読み出すことはできません。

`PedometerSettings` クラスを使用するには、その i アプリの ADF で `AccessUserInfo` キーが設定されている必要があります。

23.2 脈拍計機能

脈拍計機能の API は、`com.nttdocomo.opt.device.Pulsemeter` クラスで提供されます。この機能では、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 `Pulsemeter` クラスの static メソッド `getPulsemeter()` を使用して、`Pulsemeter` 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 `Pulsemeter` 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。`Pulsemeter` 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 {
    Pulsemeter pm = Pulsemeter.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==Pulsemeter.STATUS_NONE) {              // 正常に測定中
        } else if (state==Pulsemeter.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==Pulsemeter.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、非活性化状態遷移、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：

- 非活性化状態の待ち受けアプリケーションは、脈拍計機能を使用することはできません。また、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もしくは非活性化状態に遷移すると自動的に停止します。

第 24 章

読み上げ機能

読み上げ機能は、**String** オブジェクトもしくは **XString** オブジェクトの形で表現されたテキストを音声で読み上げるユーザーインタフェース機能を提供します。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。

読み上げ機能は **com.nttdocomo.opt.device** パッケージで提供されます。読み上げ機能の API 構成は以下の通りです。

- **SpeechSynthesizer** : 読み上げ機能を提供するクラス
- **SpeechListener** : 読み上げイベント（読み上げ終了または中断）の通知を受けるためのリスナーインタフェース

読み上げ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。**SpeechSynthesizer** オブジェクトを使用して、読み上げ機能に関する以下の操作を行うことができます。

- ・ 読み上げ機能に関する属性の設定および参照
- ・ 読み上げイベントのリスナーオブジェクトの登録
- ・ テキストの読み上げ開始指示、および中断指示

(1) 読み上げ機能に関する属性の設定および参照

読み上げ機能の属性は、**SpeechSynthesizer** クラスの **setAttribute()** メソッドおよび **getAttribute()** メソッドを使用して設定、参照を行います。現在のプロファイルでは、読み上げ機能の属性として音声の種類と読み上げ速度の 2 種類が規定されています。属性に関する定数は、全て **SpeechSynthesizer** クラスで定義されています。

- ・ 音声の種類 (**SPEECH_VOICE**)

携帯電話にプリセットされている音声（声色）のうち、どれを使用するかを指定するための属性です。現在のプロファイルでは女声 (**FEMALE_VOICE**)、男声 (**MALE_VOICE**) およびネイティブ設定への連動 (**DEFAULT_VOICE**) の 3 種類の属性値が規定されています。アプリケーションプログラムから音声の設定が行われていない状況では、ネイティブ設定に連動した音声で読み上げが行われます。

- ・ 読み上げ速度 (**SPEECH_SPEED**)

読み上げの速度を制御するための属性です。読み上げ速度には、**MIN_SPEED** から **MAX_SPEED** までの範囲の整数値を設定することができます。アプリケーションプログラムから読み上げ速度の設定が行われていない状況では、ネイティブ設定に連動した読み上げ速度で読み上げが行われます。

(2) 読み上げ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、読み上げ機能ではアプリケーションプログラムからテキストの読み上げ開始が指示された後、実際の発声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。このため読み上げ機能では、iアプリが発声の終了や中断をイベントとして検出するための機構が備えられています。

読み上げ開始指示に対するイベントの通知を受けるには、アプリケーションプログラム側で **SpeechListener** 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.setSpeechListener()** メソッドを使用して登録します。手動での読み上げや、ネイティブの読み上げの完了や中断に対してイベントを受け取ることはありません。読み上げイベントが発生すると、リスナーオブジェクトの **speechAction()** メソッドに通知されます。

speechAction() メソッドの引数に渡されるイベントタイプ定数は **SpeechListener** インタフェースに定義されており、読み上げ終了を表す **SPEECH_COMPLETE** と、読み上げ中断を表す **SPEECH_CANCEL** の2つが規定されています。

(3) テキストの読み上げ開始指示、および中断指示

テキストの読み上げ開始指示は、**SpeechSynthesizer.speak()** メソッドを使用して行います。このメソッドには、読み上げたいテキスト (**String** オブジェクトまたは **XString** オブジェクト) を引数に指定します。

speak() メソッドは、テキストの読み上げを開始したらすぐにアプリケーションプログラムに復帰し、以降の発声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。

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、システムが実行している読み上げの中断を指示することができます。

指定されたテキストが最後まで読み上げられたら、読み上げ終了イベントがイベントリスナーに通知されます。読み上げがアプリからの指示、もしくは他の要因で中断された場合には、読み上げ中断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.speak()** メソッドに指定可能な文字数には上限（1000 文字）があります。長い文を読み上げる場合には、複数回に区切って読み上げ処理を行うようにしてください。
- 本機能は、携帯電話ネイティブの読み上げ機能の設定の影響を受けます。本機能を使用して読み上げを行うには、携帯電話ネイティブの読み上げ機能の設定を「自動」としておく必要があります。「自動」以外（「手動」もしくは「読み上げしない」）に設定されている場合には、読み上げは行われません。
- この機能は、**AudioPresenter** によるサウンド再生など、音の鳴動を伴う他の機能を使用している状況では読み上げは行われません。読み上げが行われている状況で再度読み上げ開始を指示した場合には、先発の読み上げが中断され、後発の読み上げが行われます。
- 折りたたみ形状の携帯電話では、本機能は機体が開いている状態でのみ読み上げを行うことができます。
- 上記、読み上げが行われない状況で **SpeechSynthesizer.speak()** メソッドを呼び出した際の読み上げイベント発生に関する詳細な規定は、API リファレンスを参照してください。

第 25 章

赤外線リモコン受信機能

i アプリ基本 API では、`com.nttdocomo.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。

赤外線リモコン受信 API は、`com.nttdocomo.opt.device.IrReceiver` 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。

- ・ リモコン信号パラメータの設定
- ・ 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設定

リモコン信号パラメータの設定は、`IrReceiver.setAttribute()` メソッド、および `IrReceiver` クラスに定義された各種属性定数を使用して行います。設定可能なリモコン信号パラメータには、以下のものがあります。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。

- ・ データフレームスタート部の High 区間長 (`START_HIGH` 定数 単位 μ 秒)
データフレームのスタート部における、High 区間の長さを設定します。この項目は、`IrRemoteControlFrame.setStartHighDuration()` メソッドによる設定項目に対応します。
- ・ データフレームスタート部の Low 区間長 (`START_LOW` 定数 単位 μ 秒)
データフレームのスタート部における、Low 区間の長さを設定します。この項目は、`IrRemoteControlFrame.setStartLowDuration()` メソッドによる設定項目に対応します。
- ・ データフレームストップ部の High 区間長 (`STOP_HIGH` 定数 単位 μ 秒)
データフレームのストップ部における、High 区間の長さを設定します。この項目は、`IrRemoteControlFrame.setStopHighDuration()` メソッドによる設定項目に対応します。

- データフレームストップ部の Low 区間長 (STOP_LOW 定数 単位 μ 秒)

データフレームのストップ部における、Low 区間の長さを設定します。この項目は、**IrRemoteControlFrame** クラスの設定項目には存在しません(**IrRemoteControlFrame** クラスでは、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。

- 論理 0 の High 区間長 (DATA0_HIGH 定数 単位 μ 秒)

データフレームに格納されているビットデータのうち、論理 0 に対応する信号の High 区間の長さを設定します。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。

- 論理 0 の Low 区間長 (DATA0_LOW 定数 単位 μ 秒)

データフレームに格納されているビットデータのうち、論理 0 に対応する信号の Low 区間の長さを設定します。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。

- 論理 1 の High 区間長 (DATA1_HIGH 定数 単位 μ 秒)

データフレームに格納されているビットデータのうち、論理 1 に対応する信号の High 区間の長さを設定します。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。

- 論理 1 の Low 区間長 (DATA1_LOW 定数 単位 μ 秒)

データフレームに格納されているビットデータのうち、論理 1 に対応する信号の Low 区間の長さを設定します。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。

- 論理 0 波形、論理 1 波形の基準長 (TTIME 定数 単位 μ 秒)

論理 0 波形、論理 1 波形の基準長を設定します。このパラメータは、基本 API の赤外線リモコン機能の設定項目には存在しません。

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、論理 0 波形と論理 1 波形は概ね以下のように設計されます。

論理 0 High 区間長 = 論理 0 Low 区間長 = 論理 1 High 区間長 (この長さを T とします)

論理 1 Low 区間長 = 3T

このパラメータでは、上記の基準となる時間 T の長さを設定します。

i アプリは上記の各リモコンパラメータについて、最小値 (MIN 定数) と最大値 (MAX 定数) 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。i アプリはリモコン信号を受信する前に、**IrReceiver** クラスの上記全ての属性定数に対して値を設定する必要があります。

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。受信されたリモコン信号は (a) 項で解説したリモコン信号パラメータの設定に従って解釈され、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` クラスの独自の形式で、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従って変換した結果) 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。