

i アプリコンテンツ開発ガイド for Star-2.x

～ 詳細編 ～

第 1.01 版

平成 24 年 8 月 31 日
株式会社 N T T ドコモ



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

Sun、Sun Microsystems、Java、J2ME および J2SE は、米国およびその他の国における米国 Sun Microsystems, Inc. の商標または登録商標です。サンのロゴマークは、米国 Sun Microsystems, Inc. の登録商標です。

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

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 2000-2012 NTT DOCOMO, INC. 2-11-1 Nagata-cho, Chiyoda-ku, Tokyo, 100-6150, Japan and Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California, 94303, U.S.A. All rights reserved.

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

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

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

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

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

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

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

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

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

目 次

本書の目的	8
対象読者	8
本書の構成	8
表記上の規則	9
関連ドキュメント	9
第1章 はじめに	10
1.1 i アプリ実行環境の利点	11
1.1.1 スタンドアロン型およびクライアント/サーバー型の最適化されたサポート	13
1.1.2 永続データのローカル記憶装置サポート	14
1.1.3 ミニアプリ実行環境のサポート	14
1.2 アプリケーションマネージャ	15
1.3 i アプリ対応携帯電話のセキュリティモデル	15
1.4 i アプリのインストールとライフサイクル	16
1.4.1 i アプリのダウンロード	16
1.4.2 i アプリの起動	17
1.4.3 i アプリの更新	18
1.4.4 i アプリの削除	18
1.4.5 携帯電話本体と外部メモリ間の i アプリの移動	18
1.5 DoJa バイナリ互換機能	19
第2章 i アプリ実行環境	21
2.1 i アプリ実行環境	21
2.1.1 ネイティブアプリケーションとオペレーティングシステム	21
2.1.2 アプリケーションマネージャ	22
2.1.3 WidgetView	22
2.1.4 仮想マシン	22
2.1.5 i アプリ API および CLDC で規定されている API	22
2.2 i アプリ実行環境における API の概要	23
2.2.1 ネットワークアーキテクチャ	23
2.2.2 CLDC で規定されている API	24
2.2.3 i アプリ API	28
第3章 設計上の考慮事項	45
3.1 i アプリ対応携帯電話の特性	45
3.2 メモリの問題	46
3.2.1 メモリ使用量の最小化	47
3.2.2 クラスファイルサイズ	49
3.3 使い易さと GUI の設計	49
3.3.1 ターゲットユーザー向けの設計	49
3.3.2 i アプリの操作方法の設計指針	50
3.3.3 スクロール	50
3.3.4 キーおよびファンクションキー	50
3.3.5 データ入力	51
3.3.6 パスワード入力	51
3.3.7 スレッドの使用	51
3.4 セキュリティ	52

3.5	無線ネットワークでの操作	52
3.6	i アプリのライフサイクル	53
3.6.1	i アプリの実行状態遷移	53
3.6.2	ライフサイクルイベント処理	54
3.7	Star アプリケーションイベント	56
3.8	ハードウェアのアクセス	57
3.9	エラー処理	58
3.10	ミニアプリにおける制限	58
3.11	携帯電話の機種判別	58
3.12	FOMA 携帯電話における複数アプリケーション同時起動	59
第 4 章	ユーザーインタフェースの設計	60
4.1	高レベル UI コンポーネントによるプログラミング	63
4.1.1	Panel の使用	63
4.1.2	Dialog の使用	63
4.1.3	コンポーネントの使用	64
4.1.4	コンポーネントの配置	72
4.1.5	フォーカスとスクロール	75
4.1.6	高レベルイベントの処理	75
4.1.7	コンポーネントのフォントサポート	76
4.2	低レベル API によるプログラミング	76
4.2.1	Canvas の使用	76
4.2.2	グラフィックスの描画	77
4.2.3	Image オブジェクトへの描画	79
4.2.4	カラーサポート	80
4.2.5	低レベルイベントの処理	81
4.2.6	Canvas における IME の利用	82
4.2.7	イメージリファレンスによる直接描画	84
4.3	ファンクションキー/決定キー/上下左右キー矢印	85
4.4	イベントリスナー	86
4.4.1	ComponentListener	87
4.4.2	SoftKeyListener	88
4.4.3	KeyListener	89
4.4.4	MediaListener	89
4.4.5	TimerListener	90
4.5	マルチメディアデータの使用	90
4.5.1	i アプリが利用可能なマルチメディアデータ	90
4.5.2	プレゼンタの利用	92
4.5.3	複数の AudioPresenter の同時再生	97
4.5.4	サウンド再生の同期イベント	98
4.5.5	NativeMoviePlayer での i モーションの再生	98
4.5.6	マルチメディアデータの外部メモリへの出力制御	100
4.5.7	メディアデータ利用時のメモリ管理	101
4.6	イメージ処理	102
4.6.1	イメージエンコーダ	102
4.6.2	ピクセル操作	103
4.6.3	イメージ回転・反転・拡大縮小表示	103
4.6.4	イメージマップ	104
4.6.5	スプライト	105

4.6.6 パレット付きイメージ	105
4.6.7 イメージに対する透過色指定と半透明描画指定	106
第5章 通信の制御	107
5.1 クライアント/サーバープログラミング	108
5.1.1 接続の終了	111
5.2 セッション管理	112
5.3 HTTPS による安全な通信	114
第6章 テキスト処理	115
6.1 テキスト処理	115
6.2 i アプリ対応携帯電話の絵文字	116
第7章 ScratchPad とリソースの使用	120
7.1 ScratchPad に対する読み取りと書き込み	120
7.2 リソースの読み取り	123
7.3 エラー処理	124
第8章 プラットフォームリソースのアクセス	125
第9章 ミニアプリ	128
9.1 ミニアプリの概要	128
9.1.1 ミニアプリの実行環境	128
9.1.2 WidgetView とミニアプリの表示状態	129
9.1.3 ミニアプリのライフサイクル	130
9.1.4 貼り付け設定	133
9.1.5 パレット	133
9.2 ミニアプリの作成	133
第10章 赤外線通信による OBEX 外部接続	138
10.1 OBEX によるデータ送受信	138
10.2 OBEX 外部接続 API	140
10.2.1 OBEX クライアント API	140
10.2.2 OBEX サーバーAPI	143
10.3 外部機器接続のヒント	146
10.4 IrSimple	147
第11章 アプリケーション連携	149
11.1 ブラウザ連携起動	150
11.1.1 ブラウザからの i アプリの起動	150
11.1.2 i アプリからのブラウザ起動	152
11.2 メール連携起動	153
11.2.1 メールからの i アプリの起動	153
11.3 外部機器連携起動	156
11.3.1 外部機器からの i アプリの起動	156
11.4 i アプリ連携起動	159
11.4.1 連携モードによる i アプリ連携起動	159
11.4.2 ランチャーモードによる i アプリ連携起動	160
11.4.3 同一パッケージ内での i アプリ間連携起動	162

11.5	i アプリ更新機能連携起動	163
11.6	通話機能の呼び出し	163
11.7	電話帳管理機能の呼び出し	164
11.7.1	電話帳グループの新規登録	164
11.7.2	電話帳エントリの新規登録	164
11.8	ブックマーク管理機能の呼び出し	165
11.9	スケジュール管理機能の呼び出し	166
11.9.1	スケジュールの新規登録	166
11.9.2	スケジュールの連携起動	169
11.10	画像データ管理機能の呼び出し	169
11.10.1	画像データの新規登録	170
11.10.2	画像データの選択読み込み	171
11.10.3	画像データの ID 指定読み込み	171
11.11	カメラ機能の呼び出し	171
11.11.1	カメラの制御	172
11.11.2	撮影画像の携帯電話間での交換	173
11.11.3	コード認識	174
11.12	映像データ管理機能の呼び出し	178
11.13	アラーム機能の呼び出し	179
11.14	データ BOX フォルダ管理機能の呼び出し	179
11.14.1	フォルダの新規登録	180
11.14.2	フォルダのエントリ ID の選択取得	180
11.15	コンテンツ取得保存機能の呼び出し	181
第 12 章	赤外線リモコン	184
12.1	制御信号の構成	184
12.2	赤外線リモコン API	186
12.2.1	IrRemoteControl	186
12.2.2	IrRemoteControlFrame	187
第 13 章	3D グラフィックス・3D サウンド	189
13.1	3D グラフィックス描画機能	189
13.1.1	3D グラフィックス描画機能	189
13.2	3D サウンド制御機能	191
13.2.1	3D サウンド制御機能のクラス構成	192
13.2.2	Audio3D	194
13.2.3	定位の指定	194
13.2.4	3D サウンド制御未サポート機種における振る舞い	195
第 14 章	内部ストレージ	196
14.1	内部ストレージ機能 API の API 構成	196
14.2	内部ストレージの構造	198
14.3	内部ストレージ機能 API の利用	199
14.3.1	自アプリ固有フォルダの取得	200
14.3.2	フォルダの操作	200
14.3.3	ファイルの操作と入出力	201
第 15 章	ユーティリティ API	203

15.1	デジタル署名 API	203
15.1.1	デジタル署名 API のクラス構成	204
15.1.2	デジタル署名 API の利用	204
15.2	マナーモード中の音声出力設定	206
第 16 章	i アプリのビルド、テスト、 およびパッケージ化	208
16.1	i アプリ開発環境のインストール	208
16.2	i アプリ開発環境のコンポーネント	209
16.3	開発サイクルの概要	210
16.4	通信先アプリケーションの作成	211
16.5	i アプリのパッケージ化	211
16.5.1	ADF の作成	211
16.5.2	JAR ファイルの作成	221
16.6	i アプリのテスト	223
16.7	i アプリ Bookmark	223
16.7.1	i アプリ Bookmark の構成	224
16.7.2	i アプリ Bookmark の動作	229
16.7.3	コンテンツパッケージからの i アプリ起動	230
第 17 章	i アプリの配布	231
17.1	i アプリ配布のための Web サーバーの構成	231
17.2	サーバー側アプリケーション実行のための Web サーバーの構成	232
17.3	特定メーカーの i アプリ対応携帯電話へのダウンロード	233
付録 A	ミニアプリにおける機能制限	234
A.1	ミニアプリで制限される機能	234
A.2	i アプリ API 利用上の注意点	235
A.2.1	ユーザーインタフェース	235
A.2.2	通信の制御	236
A.2.3	ScratchPad へのアクセス	237
A.2.4	ローカルリソースへのアクセス	237
A.2.5	プラットフォームリソースへのアクセス	237
A.2.6	アプリケーション連携	238
A.2.7	赤外線リモコン	238
付録 B	2in1 機能が i アプリに与える影響	239

本書の目的

『i アプリコンテンツ開発ガイド』は、i モード携帯電話向けにアプリケーションコンテンツサービスを提供する i アプリの設計や開発に必要な情報を、i モード向けコンテンツプロバイダに提供するものです。このガイドの目的は、i アプリの作成や開発に役立つシンプルなモデルを提供することにあります。さらに、このガイドには、この技術に関する説明と、使い易い i アプリを作成する上で開発者が準拠すべきプロセスの詳細が述べられています。この巻頭部分では、さらに次の項目について説明しています。

- 対象読者
- 本書の構成
- 表記上の規則
- 関連ドキュメント

対象読者

『i アプリコンテンツ開発ガイド』には、i アプリを利用したサービスの開発者向けの情報が記載されています。したがって、サービスの設計者や、サーバーおよびクライアントプログラムのプログラマ、サービスの導入担当者は、このガイドを読む必要があります。このガイドでは、次の知識を前提としています。

- Java 言語を使ったネットワークアプリケーションの基礎知識
- 開発した i アプリが実行される i モード携帯電話の製品基本仕様

本書の構成

次の表に、各主題について記載している章を示します。

内容	参照先
i アプリ実行環境の紹介と概要	第 1 章、第 2 章
設計上の考慮事項	第 3 章
仕様の解説	第 4 章～第 15 章
開発プロセス	第 16 章、第 17 章

表記上の規則

このガイドで使用されている表記上の規則を次に示します。

表記	意味
タイプ文字	タイプ文字は、そのテキストがコード例、クラス、インタフェース、メソッド、またはデータ型であることを示します。
ボールドテキスト	ボールドは、そのテキストがメニューやファンクションキーのオプションであることを示します。
[]	書式などの説明において、[] で囲まれた内容はオプションであることを示します。実際の指定では、必要に応じてこれらを設定したり省略することができます。
< >	書式などの説明において、< > で囲まれた内容は、その部分を適切な値に置き換えて設定することを示します。

関連ドキュメント

このガイドに関連するドキュメントを次に示します。

- 『i アプリコンテンツ開発ガイド for Star-2.x API リファレンス編』
- 『i アプリコンテンツ開発ガイド for Star-2.x i アプリオプション・i アプリ拡張編』
- 『i アプリコンテンツ開発ガイド for Star-2.x オプション・拡張 API リファレンス編』
- 『Star-2.0 API i α ppli Development Kit ユーザーズガイド』
- 『Java 2 Platform, Micro Edition (J2ME) Connected Limited Device Configuration (CLDC) specification』 (<http://java.sun.com/products/cldc/>)

Star プロファイルに対応する i アプリ対応携帯電話には、CLDC1.1 仕様で規定されている API を利用可能なアプリケーション実行環境が搭載されます。

なお、上記に含まれる URL は予告なく変更される場合があります。

第 1 章 はじめに



i アプリ対応携帯電話では、電話やインターネットアクセスの他に、i モードコンテンツプロバイダの Web サイトからアプリケーションコンテンツ（i アプリ）をダウンロード、実行する機能がサポートされています。このガイドは、DoJa プロファイルの後継となる i アプリプラットフォーム - Star プロファイルに対応したアプリケーションコンテンツを開発するためのプログラミングガイドです。Star プロファイルでは、DoJa プロファイルの持つ多くの基本機能を引き継ぎつつ、ウィジェットアプリ（iWidget）など新しいアプリケーションサービスのための機能も提供しています。

本書では、Star プロファイルに対応したアプリケーション実行環境の全般的な特徴と機能について解説します。

注意事項：

- Star プロファイルと DoJa プロファイルでは、ソースコードレベルで互換性はありません。DoJa 向けに開発されたアプリケーションを Star アプリケーション実行環境で動作させる場合には、Star プロファイルの仕様に合わせてアプリケーションを移植する作業が必要となります。また、DoJa に存在している機能の中には、Star では廃止されるものもあります（待ち受けアプリケーション機能やダウンロード即起動 i アプリなど）。この互換性の問題に対応するため、今後当面の期間において、Star 対応携帯電話にも DoJa アプリケーション実行環境が搭載されます。アプリケーション開発者は早いうちに、Star 対応携帯電話には Star アプリケーションを提供することを検討してください。DoJa アプリケーション実行環境は、将来的には Star 対応携帯電話に搭載されなくなる可能性があります。また今後は、新規機能の追加などの対応は基本的に Star アプリケーション実行環境のみに対して行われます。
- Star アプリケーションは、DoJa プロファイルの API を使用することはできません。また、DoJa アプリケーション（Star 対応携帯電話上で動作させる DoJa アプリケーション）は、Star プロファイルの API を使用することはできません。両プロファイルの API を混在して使用しないようにしてください。
- 本書では、Star プロファイルをターゲットとして解説を行います。DoJa プロファイルについては、DoJa プロファイル向けの i アプリコンテンツ開発ガイドを参照してください。
- Star-1.3 プロファイル以降の i アプリ対応携帯電話では、搭載される i アプリ実行環境の内部仕様が一部変更となり、これに伴い i アプリビルド時の事前検証プロセスが不要になります（事前検証が施された i アプリも実行は可能です）。NTT ドコモが提供する i アプリ開発環境は、Star-1.3 プロファイル向け以降のものはビルド時に事前検証プロセスを行わないようになっています。このため、Star-1.3 プロファイル向け以降の i アプリ開発環境でビルドした i アプリは Star-1.2 プロファイル対応の i アプリ対応携帯電話では動作しないため注意してください（詳細については第 16 章も参照してください）。

1.1 i アプリ実行環境の利点

i アプリサービスにおいて、コンテンツプロバイダは i アプリを Java 言語を使って作成します。i アプリ実行環境では、携帯電話のユーザーインタフェース、通信、テキスト変換、グラフィックス、マルチメディア、携帯電話上へのデータの保管といった処理を制御する API（i アプリ API）が提供されます。i アプリ API は Star プロファイルでも DoJa プロファイルと同様に以下の 3 つのカテゴリの API 群から構成されます。

- i アプリ基本 API

共通仕様として API および動作が規定されており、全ての機種に共通に搭載される API です。

- i アプリオプション API

共通仕様として API および動作が規定されていますが、搭載有無の判断はメーカーに委ねられる API です。このカテゴリの API の使用にあたっては、API が搭載されていない機種があることを前提にしなければなりません。

- i アプリ拡張 API

同一の機能を実現するものであっても、API および動作がメーカー毎に異なる可能性のある API です。このカテゴリの API の使用にあたっては、API が搭載されていない機種があることと、機種により API の使用方法などが異なる（同機能を持つ機種間であってもソースコードレベルでの互換性が保持されない）ことを前提にしなければなりません。

なお、本書「i アプリコンテンツ開発ガイド 詳細編」では主に i アプリ基本 API を解説の対象とします。i アプリオプション API および i アプリ拡張 API の詳細については、「i アプリコンテンツ開発ガイド i アプリオプション・i アプリ拡張編」および「i アプリコンテンツ開発ガイド オプション・拡張 API リファレンス編」を参照してください。

i アプリ実行環境には、この他に次の機能があります。

- i アプリの動的な配布
- アプリケーションマネージャ
- シンプルなサンドボックスモデルなどのセキュリティ機能。

i アプリ実行環境では、上記に記載した i アプリ API に加え、CLDC 仕様で規定されている API を利用することができます。i アプリ実行環境は次の 4 つの API レイヤから構成されています。

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

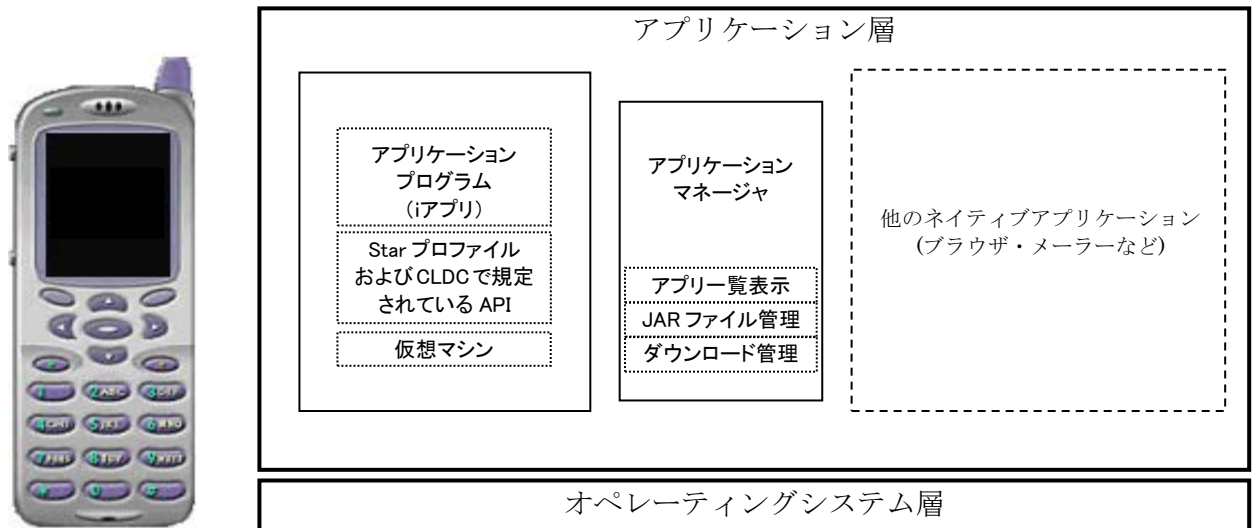


図 1 i アプリ対応携帯電話における i アプリ実行環境

注意事項：

- 多くの i アプリでは、CLDC で規定されている API と i アプリ基本 API だけを使用します。しかし特殊な i アプリの中には、ユーザーへの表現力を増すために i アプリオプション API や i アプリ拡張 API を使用するものがあります。これらの API を使用する i アプリは、メーカーにより動作しない可能性があるため注意が必要です。
- i アプリオプション API、i アプリ拡張 API は、将来のプロファイルにおいて i アプリ基本 API に取り入れられる可能性のあるものが含まれています。また逆に、将来のプロファイルで削除される可能性のあるものも含まれています。

i アプリ対応携帯電話など、資源に制約がある小型ネットワーク接続デバイスには次のような特徴があります。

- メモリの全体量が少ない。
- 消費電力が限られている。一般には電池で動作する。
- 何らかのネットワークに接続される。途切れやすい無線ネットワークに接続されていることも多く、帯域幅も限られている。
- デバイスの提供するサービスに最適化されたユーザーインターフェースが必要となる。

i アプリ実行環境は、上記のような特徴を持つ携帯電話に最適化されたアプリケーションプラットフォームです。i アプリ実行環境には、次のような機能を持つ API が含まれています。

- HTTP や HTTPS によるネットワーク接続
- ユーザーインターフェースを定義するコンポーネント
- 低レベルのグラフィックス制御
- データの保存に使用する ScratchPad
- i アプリと他のアプリケーション（ブラウザ、メーラ、i アプリなど）との連携動作
- カメラなど、携帯電話搭載デバイスの利用
- 赤外線ポートを利用したの外部接続（IrOBEX、赤外線リモコン）

1.1.1 スタンドアロン型およびクライアント/サーバー型の最適化されたサポート

i アプリサービスでは、スタンドアロン型とクライアント/サーバー型の両方の形態のプログラミングをサポートしています。

スタンドアロン型 i アプリでは、アプリケーションファイルとデータの両方が携帯電話上に存在し、サーバーと対話動作することはありません。ゲームや計算機、ユーティリティの多くはスタンドアロン型 i アプリとして作成することができます。

スタンドアロン型 i アプリでは、その i アプリの本来の目的を達するためにサーバーに接続することはありませんが、i アプリを携帯電話にダウンロードする際には、サーバーとの接続が必要です。i アプリを、インターネットを介して携帯電話にダウンロードできるようにするための方法などについては、第 17 章を参照してください。

スタンドアロン型 i アプリにも価値があるとはいえ、魅力ある i アプリの多くはクライアント/サーバー型の形態を取るでしょう。クライアント/サーバー型 i アプリでは、携帯電話の制約された処理能力が、サーバーのはるかに大きな処理能力や強力なデータ検索能力によって拡張されます。i アプリ実行環境には、i アプリとコンテンツプロバイダのサーバーをインターネットを介して接続するための、HTTP および HTTPS に基づくネットワーク接続を行う API が含まれています。また、赤外線ポートを介して近距離にある 2 台の携帯電話をクライアントおよびサーバーとして動作させるための IrOBEX API も含まれています。

HTTP(S)接続

HTTP(S)接続を利用する i アプリでは、クライアントコンピュータ（この場合は携帯電話）の処理能力に、インターネット上にあるリモートサーバーの処理能力やデータアクセス機能を結合することができます。携帯電話は処理能力や記憶装置容量が限られていますが、強力な記憶装置とプロセッサを持つサーバーのフロントエンド・プロセッサとして機能することができます。携帯電話にはデータ入力や表示サイズに制限があるものの、適切な設計やプログラミングに基づいて i アプリを作成することにより、i アプリ対応携帯電話のユーザーは、快適に情報にアクセスしたり、電子商取引のようなリモートサービスを受けることができますようになります。

クライアント/サーバー型 i アプリにおけるサーバーの代表的な役割は、要求されたデータをデータベースから検索し、それを要求元のクライアントに返すというものです。しかし場合によっては、クライアントの限られた処理能力とメモリ容量では不適切または不可能な計算処理をサーバーで代行することもあります。

HTTP(S)接続を利用したクライアント/サーバー型 i アプリの作成については、第 5 章を参照してください。

IrOBEX 外部接続

IrOBEX 外部接続は、携帯電話に装備された赤外線ポートを使用して、近距離の外部機器との間でデータ送受信を行うためのシンプルなクライアント/サーバーモデルを実現します。i アプリ実行環境は IrOBEX 外部接続のクライアント API とサーバー API の両方を持っており、携帯電話間でのデータの送受信もサポートしています。この機能を利用することで、名刺交換アプリケーションやスケジュール交換アプリケーションのような、携帯電話間で小さなデータをやりとりする i アプリを実現することができます。

IrOBEX 外部接続では、接続先の機器のハードウェア条件や周囲の環境などによっては通信を行えない場合があります。

なお、以降は特に注記のない場合クライアント、サーバーはそれぞれ HTTP(S)通信を行う場合のクライアント、サーバーを指すものとします。IrOBEX においてはそれぞれ IrOBEX クライアント、IrOBEX サーバーと記載します。

1.1.2 永続データのローカル記憶装置サポート

i アプリ実行環境では、ScratchPad と呼ぶローカル記憶装置のメカニズムがサポートされます。このメカニズムでは、永続性を持つデータ（i アプリの実行が終了しても保持され続けるデータ 永続データとも言います）の保管領域が携帯電話内の記憶装置に確保されます。永続データを格納する必要があるスタンドアロン型 i アプリは ScratchPad にデータを格納しますが、クライアント/サーバー型 i アプリは、通常は一部の情報を ScratchPad に格納するだけで、残りのデータはサーバー側に格納します。

サーバー側の永続データは、ほとんどの場合、データベースに格納されます。クライアントデータは HTTP リクエストでサーバーに送信され、HTTP リクエストがサーブレットや CGI スクリプトと対話します。そして、サーブレットや CGI スクリプトが HTTP リクエストとデータベースアクセスプロトコルとの仲介として働きます。このため、サーバー側には常に HTTP サーバーが必要です。

永続データの格納場所を決定するには、データの性質を考慮する必要があります。永続データを失っても影響が小さい場合は、ScratchPad に格納し、そうでない場合は、サーバー側に格納します。ScratchPad に保存されたデータは、携帯電話のハードウェア的な故障などの原因によって失われてしまう可能性があるためです。

i アプリ対応携帯電話のセキュリティ機能の一つに、アプリケーションマネージャ(1.2 項参照)によって個々の i アプリに個別に割り当てられた ScratchPad メモリの領域は、他の i アプリからはアクセスできないという点があります。また、複数の i アプリがアプリケーション実行メモリ上でデータを共有することもできません。複数の i アプリ間でデータを共有する場合は、サーバーを各 i アプリ共通のデータ格納領域として使用する必要があります。

1.1.3 ミニアプリ実行環境のサポート

i アプリ実行環境では、小さな i アプリをディスプレイ上で同時に複数実行可能な WidgetView と呼ばれる実行環境がサポートされています。WidgetView には、ユーザーのお気に入りの i アプリをいくつか貼り付けておくことができます。WidgetView のサポートにより、PC の世界のウィジェットのように、ユーザーは待ち受け状態からワンボタン操作で WidgetView を呼び出して、i アプリの提示する情報をいつでも簡単に見ることができます。この機能は、i ウィジェットというサービス名称で呼ばれることもあります。

WidgetView では複数の i アプリを同時に実行可能ですが、その i アプリのサイズや利用可能な機能に制限があります。WidgetView 上で動作する i アプリはミニアプリ（もしくはウィジェットアプリ）と呼ばれ、通常の i アプリと区別されています。通常の i アプリは、ミニアプリに対して、フルアプリと呼ばれることがあります。

また、フルアプリとミニアプリを 1 つの JAR ファイルにパッケージすることで、フルアプリとして携帯電話の全ての機能（ミニアプリからの利用が制限されている機能）を利用したり、ミニアプリとして情報をユーザーに簡易表示したりを容易に切替えることも可能です。1 つの JAR ファイルにパッケージされた i アプリはフル+ミニアプリと呼ばれることがあります。

1 つの JAR ファイルにフルアプリのみがパッケージされた i アプリを「フルのみアプリ」、1 つの JAR ファイルにミニアプリのみがパッケージされた i アプリを「ミニのみアプリ」と呼ぶ場合があります。

以降、本書では断りがない限り、「i アプリ」は「フルアプリ」あるいは「フル+ミニアプリ」を指すものとします。

1.2 アプリケーションマネージャ

i アプリ実行環境が搭載されている携帯電話には、携帯電話にインストールされている i アプリを管理するコンポーネントが含まれています。このコンポーネントはアプリケーションマネージャと呼ばれ、次のような機能を持っています。

- 携帯電話に格納されている i アプリをリスト表示する。
- i アプリの実行管理（起動や強制終了、i アプリ実行環境と他のアプリケーションの間の仲介など）を行う。
- i アプリをインストールまたは更新（バージョンアップ）する。
- 携帯電話に格納されている i アプリを削除する。いったん携帯電話上に保存された i アプリの削除は、ユーザーの明示的な指示に基づいてのみ行われます。

アプリケーションマネージャは i アプリおよび仮想マシンから独立したネイティブコンポーネントであり、これを i アプリのプログラムロジック（以降、単にアプリケーションプログラムと呼びます）から制御することはできません。

フルアプリを終了させようとしても終了しない場合は、携帯電話の終話キーを押すことによって、アプリケーションマネージャにフルアプリの強制終了を指示することができます。また、ミニアプリの強制終了はファンクションキーに割り当てられます。

1.3 i アプリ対応携帯電話のセキュリティモデル

アプリケーションマネージャでは i アプリ実行環境特有の機能として、次のセキュリティ機能が提供されます。

- 携帯電話上では、フルアプリは同時に1つしか動作することはできず、また複数の i アプリが同時に動作可能なミニアプリ実行環境においても、各々の i アプリはそれぞれが独立した空間で動作します。そのため、ある i アプリが別の i アプリに干渉したり、別の i アプリのデータにアクセスしたりすることはできません。
- アプリケーションマネージャは i アプリおよび仮想マシンから独立したネイティブコンポーネントであり、i アプリからこれを制御することはできません。i アプリのダウンロードや管理はアプリケーションマネージャが行います。そのためユーザーにとっては、ダウンロードした i アプリの不正な行為やバグによる誤動作などから携帯電話が保護されることになります。
- i アプリは、携帯電話の記憶装置（ScratchPad）に、制限されたアクセスしかできません。ある i アプリに割り当てられた ScratchPad の領域に、他の i アプリからアクセスすることは許可されません。したがってダウンロードされた i アプリが、他の i アプリのために ScratchPad に保存された個人情報などをユーザーの知らないうちに読み込んで別のサーバーに送信したり書き換えたりするなどの不正な行為を行うことはできません。
- i アプリが電話帳などの個人情報を含んだネイティブデータを参照することはできません。また、ブラウザやダイヤラーなど携帯電話上の他のアプリケーションにアクセスする機能では、ユーザーの意思に反して Web ブラウジングや通話発信を行うことのないよう、API 呼び出し時に必ずユーザーに動作への同意を求めます。そのためユーザーは、i アプリを安心してダウンロードおよび実行することができます。
- i アプリ実行環境の通信機能では、i アプリ対応携帯電話でサポートされる HTTP 通信および HTTPS 通信に対する簡単で使いやすいインタフェースが提供されます。i アプリは、その i アプリ自身のダウンロード元であるサーバーとしか通信できません。このネットワークセキュリティ機能により、i アプリが、ユーザーの知らない別のサーバーに対して情報を送信するようなことはありません。

i アプリ実行環境には、認定された i アプリに対してのみこれらのセキュリティ機能を緩和するトラステッド i アプリ（サービス名"i アプリ DX"）の機構が取り入れられています。トラステッド i アプリは、電話帳などのネイティブデータへのアクセスや i アプリダウンロード元以外のサーバーとの通信など、緩和されたセキュリティ条件下で動作することができます。ただしトラステッド i アプリといえども、電話帳などの特定個人に結びつくデータについては、携帯電話の外に流出することを防止するため一定の制約の元でのみアクセスが可能となります。

これらの機能（トラステッド API）は、i モードサーバーに登録された、i モードメニュー上のサービスでのみ使用することができます。それ以外の i アプリでは、トラステッド API の機能を使用することはできません。

本書では、トラステッド i アプリに関する詳細は記載しません。

1.4 i アプリのインストールとライフサイクル

携帯電話上で i アプリを実行するためには、コンテンツプロバイダの i モード対応 Web サイトから i アプリをダウンロード（インストール）しなければなりません。i アプリがダウンロードされると、i アプリの保存のために必要な各メモリをアプリケーションマネージャが割り当てます。i アプリの JAR ファイルと ScratchPad は、どちらも携帯電話の不揮発性メモリに格納されます。

次の各項では、i アプリのダウンロード、実行、削除といったライフサイクルについて説明します。

1.4.1 i アプリのダウンロード

i アプリのダウンロードは一般的に以下のような流れで行われます。なお、携帯電話の具体的な動作の詳細はメーカーにより一部異なる場合があります。

1. i モードブラウザを使って、必要な i アプリのダウンロードリンク先を決めます。

i アプリのダウンロードは、i モードブラウザの操作を契機として開始されます。ダウンロードを行うための Web ページ（HTML コンテンツ）には、通常、i アプリの説明文や、ダウンロードを開始するためにクリックするリンクが表示されています。リンクには、アプリケーションディスクリプタファイル（".jam" 拡張子をもつファイル ADF と呼ばれます。以降本書では ADF とします）への参照が含まれています。ADF は SJIS コードのテキストファイルであり、その ADF に対応付けられている i アプリの情報を表すキーと値のペアが 1 行ずつ記述されています。

このファイルの目的の 1 つは、ダウンロードの開始前に、選択された i アプリを携帯電話に正しくダウンロードできるかどうかをアプリケーションマネージャが判定できるようにすることです。事前に正しくダウンロードできないと判定できれば、i アプリを実際に携帯電話にダウンロードするコストが節約されます。数 10K～数 100K バイト以上の i アプリに対して ADF は 100～300 バイト程度と小さく、i アプリ全体をダウンロードする前に ADF をダウンロードして保存に必要な記憶容量やその機種に対する互換性をチェックすることは、通信コストの点から有利となります。

なお ADF の内容に関する詳細は 16.5.1 項を、i アプリを携帯電話に配布するための Web サーバーの準備については第 17 章を参照してください。

2. リンクをクリックして、インストールプロセスを開始します。

このユーザー操作により、ブラウザは、アプリケーションマネージャに ADF の位置を示す URL を渡します。以降、i アプリのダウンロードとインストールはアプリケーションマネージャにより行われます。

3. ADF の内容を検査し、i アプリのインストール可否を確認します。

アプリケーションマネージャは、指定された ADF にすべての必須キーが指定されているかチェックします。そして ADF の記述内容（JAR ファイルサイズや ScratchPad サイズ、ターゲットとする機種情報など）から、その i アプリをその携帯電話にインストール可能かどうかチェックし、可能であれば次のステップに進みます。

なお以下の場合には、i アプリはすでに携帯電話にインストールされているものと見なされ新規にインストールすることはできません（1.4.3 項に示す i アプリの更新は可能です）。

- ・ インストールしようとしている i アプリと同じ URL の ADF を持つ i アプリがすでに携帯電話にインストールされている場合（このような i アプリのダウンロードが指示されると、携帯電話はその i アプリが更新可能か調べ、可能であれば 1.4.3 項に示す i アプリの更新処理に移ります）

- ・ インストールしようとしている i アプリと異なる URL の ADF に対応付けられている、同じ名前 (ADF の **AppName** キー) および JAR ファイル URL (ADF の **PackageURL** キー) を持つ i アプリがすでに携帯電話にインストールされている場合

4. i アプリのダウンロードが行われます。

i アプリが携帯電話上にインストール可能であると判断されたら、アプリケーションマネージャは ADF の **PackageURL** キーを使って JAR ファイルの URL を入手し (**PackageURL** キーが相対 URL で記述されている場合は、ADF の位置をベース URL として使用します)、HTTP ダウンロードプロセスを開始します。もしダウンロードプロセス中にユーザーの中断指示や通信エラーが発生したら、アプリケーションマネージャはダウンロード中の i アプリを破棄して携帯電話をダウンロードが行われる前の状態に戻します。

5. インストール済み i アプリのリストを更新します。

アプリケーションマネージャは、インストール済み i アプリのリストに前項でインストールされた i アプリを追加します。また ADF の情報と JAR ファイルを不揮発メモリ上に保存します。

ADF で i アプリが HTTP 通信などユーザーの許可が必要な機能を使用するように設定されている場合は、ここで i アプリにそれらの機能の使用を許可するかをユーザーに問い合わせます。

1.~5.の過程を経て i アプリは携帯電話上にインストールされ、利用可能 (実行可能) な状態になります。

注意事項：

- トルカのバージョン 2.0 フォーマットに対応している携帯電話では、トルカ (詳細) の HTML コンテンツ部分に i アプリダウンロード用のタグ (第 17 章を参照) を記述することでトルカビューア経由で i アプリをダウンロードすることもできます。

1.4.2 i アプリの起動

i アプリの通常の起動手順やユーザーインターフェースはメーカーによって異なりますが、一般には次のようにして行われます。

1. ユーザーの指示によりインストール済み i アプリのリストを表示します。
2. ユーザーがリストから起動する i アプリを選択します。
3. アプリケーションマネージャが ADF の **AppClass** キーで示される i アプリのメインクラス名を指定して仮想マシンを起動します。
4. 携帯電話のディスプレイに i アプリの画面が表示されます。

また、上記のユーザー操作による i アプリの起動だけではなく、i アプリ実行環境は以下のような起動形態のバリエーションを持っています。

- ・ タイマー起動
- ・ アプリケーション連携による、ブラウザ、メーラ、外部機器 (赤外線ポートなど)、i アプリからのフルアプリ起動
- ・ **WidgetView** によるミニアプリとしてのアプリ起動

タイマー起動については、16.5.1 項の **LaunchAt** キーの項を参照してください。またアプリケーション連携については第 11 章を参照してください。

1.4.3 i アプリの更新

i アプリの更新は、ユーザーがインストール済み i アプリのリストから、メニューなどにより i アプリ更新を指示することにより行われます(詳細な操作方法是メーカーにより異なります)。

i アプリをインストールした際に、アプリケーションマネージャは ADF の URL を保存しています。ユーザーが i アプリの更新を指示すると、アプリケーションマネージャは再度この URL にアクセスし、新たな ADF を取得します。そしてその内容に従って i アプリを更新する必要があるかどうかを判定し、必要と判断された場合のみ実際の更新処理を行います。

バグの修正や新しい機能の追加を行うなどの理由で i アプリを更新する場合には、コンテンツプロバイダは次のことを行う必要があります。

- ADF の **LastModified** キーに前のバージョンより後の日時を設定します。その他のキーは必要に応じて適切な値に設定します。ただし **ScratchPad** サイズ (**SPsize** キー) の値は、前のバージョンより小さい値は指定しないようにしてください (**ScratchPad** を分割管理している場合、分割された個々のサイズが以前のものより小さくならないようにします)。更新しようとしている i アプリの **ScratchPad** サイズが以前のものと同じか大きい場合に限り、以前の **ScratchPad** の内容が更新後の i アプリに引き継がれます。
- コンテンツプロバイダの Web サイトに更新済み JAR ファイルと ADF を登録します。新しい ADF の URL は以前のものと同じにします。ADF の URL を変更すると、すでに携帯電話に i アプリをダウンロードしているユーザーが i アプリの更新を行えなくなるため注意が必要です。

i アプリの更新を指示されても、ADF の **LastModified** キーが更新されていないければ、アプリケーションマネージャは i アプリの更新は行いません。

i アプリがフル+ミニアプリである場合、フルとミニを合わせて 1 つのアプリとして i アプリの更新が行われます。

ユーザー操作による i アプリ更新の他に、API を使用して i アプリからユーザーに i アプリ更新を促す機能があります。i アプリ更新 API については第 11 章を参照してください。

1.4.4 i アプリの削除

ユーザーの要求により i アプリを削除する際の手順は、メーカーによって異なります。アプリケーションマネージャは i アプリの削除をユーザーに指示されると、携帯電話上の JAR ファイルおよびその i アプリに割り当てられた **ScratchPad** を削除します。

1.4.5 携帯電話本体と外部メモリ間の i アプリの移動

オプション機能として、携帯電話本体と外部メモリの間での i アプリ (i アプリや **ScratchPad**、ADF などの付随情報一式) の移動をサポートする機種があります。ユーザーはこの機能を利用することで、携帯電話本体にある i アプリを外部メモリ上に退避したり、外部メモリ上にある i アプリを携帯電話本体内 (移動元とは別の機体でも可) に戻したりすることができるようになります。

ただしコンテンツ制作者の権利保護のため、i アプリの移動は以下のような制約下で行われます。

- 携帯電話本体から外部メモリに i アプリを移動する際、携帯電話本体内にあった i アプリは削除されます。また、外部メモリ上では i アプリは暗号化された形で保存されており、i アプリのファイルの内容を PC などで読み出すことはできません。

- ・ i アプリを外部メモリから携帯電話本体(当初その i アプリがダウンロードされた機体以外の機体も含みます)に戻す際、当初その i アプリがダウンロードされる際に使用された UIM カードが携帯電話に挿入されている必要があります。またその後、その i アプリを移動先の携帯電話上で実行する際も、ダウンロード時に使用された UIM カードが携帯電話に挿入されている必要があります。

なお本機能をサポートする携帯電話では、i アプリが自分自身について、外部メモリを経由して移動されたことがあるかどうかを調べるための API が提供されます。これら API の詳細は、API リファレンスの `StarApplicationManager` クラス (`isMoved()`、`isMovedFromOtherTerminal()`、`clearMoved()` の各メソッド) を参照してください。

なお、外部メモリから携帯電話本体に i アプリが移動される際には、移動先の携帯電話上で新たにその i アプリをダウンロードする場合と同様のチェックが行われます(同じ i アプリが存在していないか、i アプリや ScratchPad、FeliCa チップ上に必要なリソースは確保可能かなど)。これらの条件に該当した場合は、その i アプリを携帯電話上に移動することはできません。

1.5 DoJa バイナリ互換機能

Star プロファイルの i アプリ実行環境においても、i アプリの再コンパイルや JAR・ADF の修正・再作成などを行うことなしに、従来の仕様 (DoJa 仕様) に基づいた i アプリを実行することができます。このような仕組みのことを「DoJa バイナリ互換機能」と呼びます。本項では、DoJa バイナリ互換機能によって実行する、従来の DoJa プロファイルに対応した i アプリを「DoJa アプリ」と呼びます。また、Star プロファイルに対応した i アプリを「Star アプリ」と呼びます。

バイナリ互換機能による DoJa アプリの実行については、DoJa プロファイル上での実行と比較して、機能的な大きな制限はありません。ダウンロードや起動等の操作、待ち受け実行などを含め同様に利用可能ですが、あくまでも Star プロファイルの i アプリ実行環境で DoJa アプリを動作させるため、いくつか留意点があります。本項では以降それらについて説明します。

(A) DoJa アプリと Star アプリの判別

個々のアプリが DoJa アプリか Star アプリかは、i アプリをダウンロードする際に取得する ADF 中の `AppType` キーの有無で判別されます。`AppType` キーは、その i アプリがミニアプリとして動作するか、フルアプリとして動作するかを指定するためのキーです。(詳細は 16.5.1 項を参照。) ADF 中に `AppType` キーが存在しない場合、その i アプリは DoJa アプリとみなされ、DoJa バイナリ互換機能によって実行されることになります。なお、DoJa バイナリ互換機能が対応するプロファイルは DoJa-5.1 です。

(B) DoJa アプリのダウンロード管理

`AppType` キーを ADF 中に持たないアプリ (DoJa アプリ) を、バージョンアップ操作で `AppType` キーを持つアプリ (Star アプリ) にダウンロードして置き換えることはできません (バージョンアップが失敗します)。

(C) DoJa アプリの連携起動

ブラウザや他の i アプリなどから、DoJa アプリを従来どおり連携起動することができます。ただし、Star アプリでは連携起動される際に受け取ることができる連携起動パラメータのサイズは 20480 バイトまでですが、DoJa アプリが受け取ることができる連携起動パラメータのサイズは、DoJa 仕様と同様です (i アプリ連携の場合 20480 バイト、その他の場合は 255 バイトです)。

i アプリ連携以外の連携起動において、DoJa アプリに対して 256 バイト以上 20480 バイト以下のデータが連携起動パラメータとして渡された場合、i アプリからは連携起動パラメータとして null が得られます (起動に失敗するわけではないことに注意してください)。

i アプリから i アプリの連携起動については、DoJa アプリからは DoJa アプリのみが起動可能です。DoJa アプリからは Star アプリを連携モードあるいはランチャーモードのいずれの方法でも起動することはできません。

(D) DoJa の待ち受けアプリケーション

DoJa バイナリ互換機能では、DoJa アプリを待ち受け設定にし待ち受け実行することも可能です。Star プロファイルでは待ち受けアプリケーションをサポートしないため、Star アプリは待ち受け設定したり待ち受け実行したりすることはできません。待ち受けアプリケーション機能を利用する場合は、DoJa バイナリ互換機能を利用し、DoJa アプリのままでサービスを提供してください。

第 2 章 i アプリ実行環境

この章では、i アプリ対応携帯電話に搭載される i アプリ実行環境の機能構成、API 構成の概要を解説します。

2.1 i アプリ実行環境

i アプリ実行環境は、携帯電話のネイティブアプリケーション、アプリケーションマネージャ、仮想マシン、i アプリ API および CLDC で規定されている API から構成されています。次に i アプリ実行環境の構成を示します。

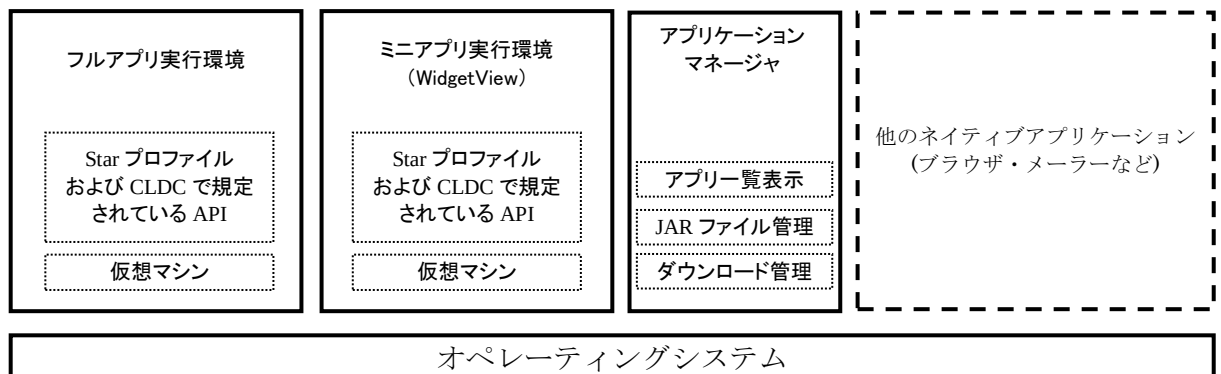


図 2 i アプリ実行環境

i アプリ実行環境には、フルアプリと DoJa アプリを実行するフルアプリ実行環境と、WidgetView と呼ばれるミニアプリ実行環境が搭載されています。

DoJa バイナリ互換機能による DoJa アプリの実行は、フルアプリ実行環境上で行われます。フルアプリ実行環境上では、同時に複数の i アプリを起動（フルアプリまたは DoJa アプリ）することはできません。また、フルアプリ実行環境とミニアプリ実行環境を同時に起動することもできません。

2.1.1 ネイティブアプリケーションとオペレーティングシステム

通常の電話やブラウザベースの i モードサービスなどは、携帯電話上のオペレーティングシステムおよびネイティブアプリケーション（ブラウザ、ネットワーキングコンポーネント、ダイヤラーなど）を使って行われます。i アプリ実行環境には、アプリケーションマネージャを通じてこれらのネイティブアプリケーションと連携動作するための機能も含まれています。

2.1.2 アプリケーションマネージャ

アプリケーションマネージャにより、JAR ファイルとして格納されている個々の i アプリの管理や仮想マシンとの通信が行われます。

アプリケーションマネージャの主な機能については 1.2 項を参照してください。

2.1.3 WidgetView

WidgetView は複数のミニアプリの起動や、実行状態を管理するネイティブアプリケーションです。ユーザーは、待ち受け状態からワンボタン操作を行うことにより WidgetView を呼び出すことができます。WidgetView の機能については第 9 章を参照してください。

2.1.4 仮想マシン

仮想マシンは、i アプリ実行環境および i アプリ（アプリケーションプログラム）自身を構成するクラスファイルを実行するためのネイティブコンポーネントです。i アプリ実行環境に搭載される仮想マシンは、メモリや CPU 能力、消費電力に制約がある小型組み込みデバイス向けに設計されています。

2.1.5 i アプリ API および CLDC で規定されている API

i アプリ API は、i アプリ対応携帯電話向けに設計された API セットです。i アプリ API では i アプリサービスに特化したアプリケーション基盤を提供しています。i アプリ API は、次のような要素から構成されています。

- ユーザーインタフェース API
- ネットワーキング API
- SJIS テキスト処理
- ScratchPad ローカル記憶装置
- アプリケーション連携起動 API およびネイティブ機能呼び出し API
- 赤外線ポート制御 (IrOBEX による外部接続および赤外線リモコン)

i アプリ API に含まれる API については、2.2.3 項を参照してください。また各機能の具体的な利用方法は第 4 章以降で詳細に解説されています。

なお i アプリ API は、以下の 3 つのカテゴリに分類されます。

- 共通仕様として API および動作が規定されており、全ての機種に共通的に搭載される i アプリ基本 API
- 共通仕様として API および動作が規定されているが、搭載有無の判断がメーカーに委ねられる i アプリオプション API
- 同一の機能を実現するものであっても、API および動作がメーカー毎に異なる可能性のある i アプリ拡張 API

全機種で動作することが求められる i アプリでは、i アプリオプション API および i アプリ拡張 API は使用しないようにしてください。

また、i アプリでは CLDC で規定されている API セットもサポートしています。アプリケーションプログラムは、i アプリ API および CLDC で規定されている API を使用して i アプリを開発します。CLDC で規定されている API セットの概要については、2.2.2 項を参照してください。

2.2 i アプリ実行環境における API の概要

次の各項では、i アプリ実行環境でサポートされる API について説明します。この節は、次の各項に分割されています。

- 2.2.1 ネットワークアーキテクチャ
- 2.2.2 CLDC で規定されている API
- 2.2.3 i アプリ API

CLDC で規定されている API は、Java 言語の標準ライブラリのサブセットクラス、および小型デバイス向けの CLDC 独自クラスから構成されています。一方、i アプリ API は i モードサービスに特化した API 群であり、NTT ドコモにより規定されています。

なお、本項では i アプリ API のうち i アプリ基本 API についてのみ記載します。i アプリオプション API および i アプリ拡張 API の詳細については、「i アプリコンテンツ開発ガイド i アプリオプション・i アプリ拡張編」および「i アプリコンテンツ開発ガイド オプション・拡張 API リファレンス編」を参照してください。

2.2.1 ネットワークアーキテクチャ

i アプリサービスのネットワークアーキテクチャは、従来の i モードサービスにおけるネットワークアーキテクチャとほぼ同様です。携帯電話とコンテンツプロバイダサイトの間のアプリケーションレベルの通信は、HTTP を使用して行われます。以下にネットワークアーキテクチャの概念図を示します。Star プロファイルは、FOMA で使用される IMT ネットワークを前提に策定されています。

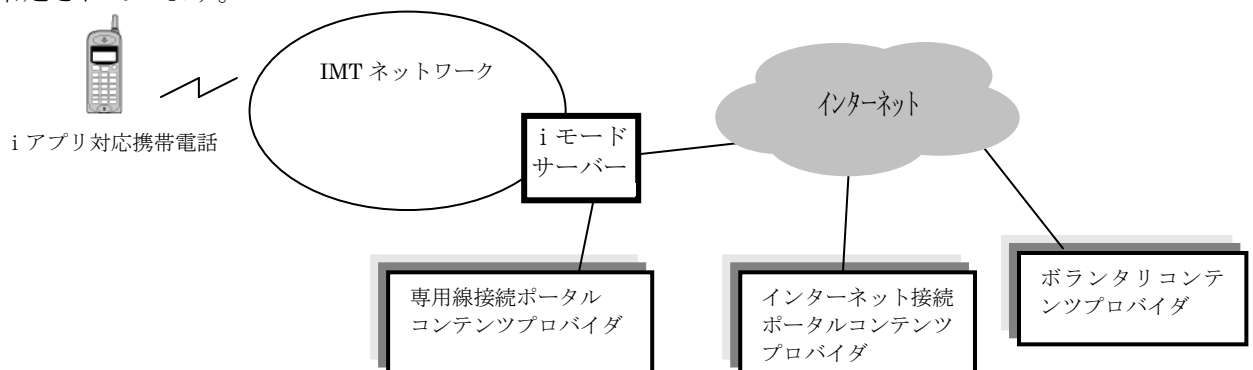


図 3 ネットワークアーキテクチャ概念図

i アプリサービスに関するネットワークオペレーションは以下の 2 種類に分類されます。

- i アプリのダウンロードとインストール
- 実行中の i アプリからのサーバーアクセス

i アプリサービスのネットワークオペレーションは、従来の i モードサービスと同様に i モードサーバーを介して行われます。携帯電話上でもアプリケーションレベルの通信プロトコルでは HTTP を使用しており、i モードサーバーではプロトコル変換に類する処理は行いません。ダウンロードされる i アプリも、コンテンツプロバイダ側 Web サイトに搭載されているファイルがそのまま携帯電話まで送信されます。

また i アプリ対応携帯電話では、携帯電話側にて SSL (Secure Sockets Layer) によるサーバー認証をサポートしています。i アプリ対応携帯電話の通信機能と SSL サポートを利用することで、セキュア HTTP (HTTPS) 通信を行うこともできます。この場合のセキュア区間は、

携帯電話からコンテンツプロバイダ側 Web サーバーまでの間となります。

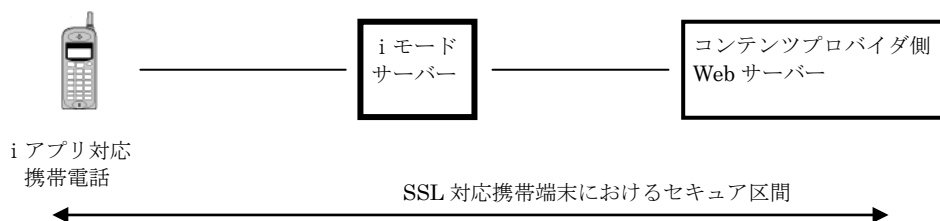


図 4 i モードサービスの SSL サポート

注意事項：

- i アプリで使用する URL (i アプリ API に指定する URL、および ADF など設定ファイルに指定する URL) では、日本語ドメイン名の使用は保証されません。日本語ドメインが割り当てられたサーバーを指定する必要がある場合には、日本語表記の URL ではなく、punycode 変換された形式の URL を指定する必要があります。

2.2.2 CLDC で規定されている API

CLDC は、Java 言語の標準ライブラリのサブセットクラス、および小型デバイス向けの CLDC 独自クラスから構成されています。CLDC 独自クラスでは、小型デバイス向けに最適化されたネットワーキング機能および入出力機能が提供されます。i アプリ実行環境では、アプリケーションプログラムはこれら CLDC で規定されている API を呼び出すことができます。

CLDC で規定されているクラス

CLDC で規定されているクラスには以下のものがあります。なお、ここに記載されたクラスやインタフェースの中には、メソッドやフィールドのレベルでサブセットとなっている (Java 言語標準ライブラリで利用可能なフィールドやメソッドでも利用できない場合がある) ものもありますので注意してください。

システムクラス

```
java.lang.Object
java.lang.Class
java.lang.Runtime
java.lang.System
java.lang.Thread
java.lang.Runnable (インタフェース)
java.lang.String
java.lang.StringBuffer
java.lang.Throwable
```

データ型クラス

```
java.lang.Boolean
java.lang.Byte
java.lang.Short
java.lang.Integer
java.lang.Long
java.lang.Character
java.lang.Double
java.lang.Float
```

参照オブジェクトクラス

java.lang.ref.Reference
java.lang.ref.WeakReference

Collection クラス

java.util.Vector
java.util.Stack
java.util.Hashtable
java.util Enumeration (インタフェース)

入出力クラス

java.io.InputStream
java.io.OutputStream
java.io.PrintStream
java.io.ByteArrayInputStream
java.io.ByteArrayOutputStream
java.io.DataInput (インタフェース)
java.io.DataOutput (インタフェース)
java.io.DataInputStream
java.io.DataOutputStream
java.io.Reader
java.io.Writer
java.io.InputStreamReader
java.io.OutputStreamWriter

Micro Edition クラス

javax.microedition.io.Connector
javax.microedition.io.ConnectionNotFoundException
javax.microedition.io.Connection (インタフェース)
javax.microedition.io.ContentConnection (インタフェース)
javax.microedition.io.Datagram (インタフェース)
javax.microedition.io.DatagramConnection (インタフェース)
javax.microedition.io.InputConnection (インタフェース)
javax.microedition.io.OutputConnection (インタフェース)
javax.microedition.io.StreamConnection (インタフェース)
javax.microedition.io.StreamConnectionNotifier (インタフェース)

カレンダーと時間のクラス

java.util.Calendar
java.util.Date
java.util.TimeZone

CLDC 仕様では時間帯として GMT がサポートされますが、i アプリ実行環境では GMT に加え JST をサポートしています。また、国際ローミングに対応する携帯電話の中には、日本以外の国・地域に対する時間帯をサポートする場合があります。

追加のユーティリティクラス

java.lang.Math
java.util.Random

エラーと例外クラス

エラークラス

java.lang.Error
java.lang.VirtualMachineError
java.lang.OutOfMemoryError
java.lang.NoClassDefFoundError

例外クラス

java.lang.Exception

```
java.lang.ClassNotFoundException
java.lang.IllegalAccessException
java.lang.InstantiationException
java.lang.InterruptedException
java.lang.RuntimeException
java.lang.ArithmeticException
java.lang.ArrayStoreException
java.lang.ClassCastException
java.lang.IllegalArgumentException
java.lang.IllegalThreadStateException
java.lang.NumberFormatException
java.lang.IllegalMonitorStateException
java.lang.IndexOutOfBoundsException
java.lang.ArrayIndexOutOfBoundsException
java.lang.StringIndexOutOfBoundsException
java.lang.NegativeArraySizeException
java.lang.NullPointerException
java.lang.SecurityException
java.util.EmptyStackException
java.util.NoSuchElementException
java.io.EOFException
java.io.IOException
java.io.InterruptedIOException
java.io.UnsupportedEncodingException
java.io.UTFDataFormatException
```

サポートされない仮想マシン機能

ソフトウェアプラットフォームのサイズを小さくするために、i アプリ実行環境の仮想マシンでは次の機能がサポートされません。

JNI

i アプリ実行環境の仮想マシンでは、Java Native Interface (JNI) はサポートされません。

スレッドグループ

i アプリ実行環境の仮想マシンにはマルチスレッド機能が実装されていますが、スレッドグループはサポートされません。したがって、スレッドの開始や停止などのスレッド操作は、個々のスレッドオブジェクトにしか適用できません。開発者がスレッドグループに対してスレッド操作を行いたい場合は、アプリケーションレベルでコレクションオブジェクトを明示的に使用してスレッドオブジェクトをグループ管理する必要があります。

ファイナライズ

CLDC 仕様で規定されている `Object` クラスには `finalize()` メソッドが含まれておらず、i アプリ実行環境の仮想マシンではオブジェクトのファイナライズはサポートされません。

Generic Connection フレームワーク (javax.microedition.io パッケージ)

CLDC 仕様では、入出力・ネットワークングフレームワークとして Generic Connection フレームワークと呼ばれるものが規定されています。i アプリ実行環境でも、このフレームワークを用いた入出力機能、ネットワークング機能が提供されます。この項では、Generic Connection フレームワークについて解説します。

Generic Connection フレームワークでは、すべての接続は `Connector` クラスのクラスメソッド `open()` を使って行われます。接続が正常に行われると、`open()` メソッドは Generic Connection インタフェース群 (`Connection` インタフェースをルートとするサブインタフェース群) の中の 1 つを実装したオブジェクトを返します。このインタフェース群は、図 5 に示されるように多数のインタフェースから構成されています。

`open()` メソッドは、URL パラメータを次の一般的な形式で受け取ります。

```
Connector.open("<protocol>:[target][parameters]");
```

Generic Connection フレームワークでは、アクセス手段やアクセス対象、およびそれに付随するパラメータは全て URL パラメータ文字列で表現し、アプリケーションプログラムはそれを **Connector.open()** メソッドに渡すことで接続を行います。サポートされているプロトコル実装のうちどのプロトコル実装を使用するかは、URL パラメータのプロトコル部により動的に決定付けられます。

このフレームワークは、同じセマンティクスをもつプロトコルクラスをグループ化した **Connection** インタフェースから継承されるサブインタフェース階層として実装されています。**Connector.open()** メソッドは、指定された URL パラメータ文字列を解析して適切な接続手段を選択し、接続処理を行った上で **Connection** インタフェースの実装オブジェクト (**Connection** オブジェクト) を返します。Generic Connection フレームワークで扱う基本的なインタフェースタイプには、全てのインタフェースのスーパーインタフェースである **Connection** インタフェースを除き、次の 6 つがあります。

- 基本的なシリアル入力 : **InputConnection**
- 基本的なシリアル出力 : **OutputConnection**
- データグラム指向の通信 : **DatagramConnection**
- ストリーム指向の通信 (**InputConnection** および **OutputConnection** のサブインタフェース)
 : **StreamConnection**
- クライアント/サーバー接続をサーバーに知らせる通知メカニズム
 : **StreamConnectionNotifier**
- コンテンツ送受信信用通信 (**StreamConnection** のサブインタフェース)
 : **ContentConnection**

Connection インタフェース群は 1 つの階層を形成し、階層がルート **Connection** インタフェースから継承されるに従って階層の機能が向上します。次に、Generic Connection クラス階層の主なクラス間の関係を示します。

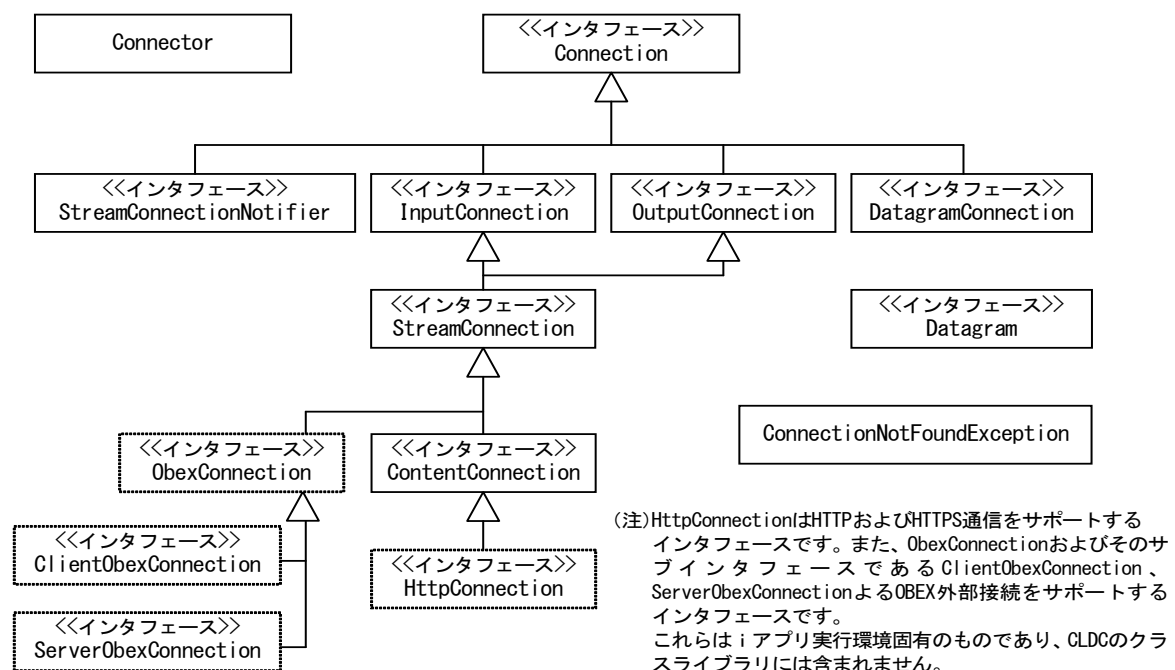


図 5 Generic Connection クラス階層

i アプリ実行環境では、i アプリが外部にアクセスする際に使用する以下のインタフェースをサポートしています。

```

HTTP(S)通信      : com.docomostar.io.HttpConnection
                   (URLConnection のサブインタフェース)

ScratchPad        : StreamConnection

リソース          : InputConnection

IrOBEX 外部接続   : com.docomostar.io.ObexConnection
                   com.docomostar.io.ClientObexConnection
                   com.docomostar.io.ServerObexConnection
                   (StreamConnection のサブインタフェース)
  実際には、これらのインタフェースを実装した以下のクラスが使用されます。
                   com.docomostar.io.IrClientObexConnection
                   com.docomostar.io.IrServerObexConnection

```

なお、`Connector.open()` メソッドにはオプションのパラメータとして以下の 2 つが用意されています。

- ・ アクセスモード (READ、WRITE、READ_WRITE)
- ・ タイムアウト例外要求フラグ

アクセスモードは、アクセス対象に行う操作の内容に応じて適切な値を設定します。また、タイムアウト例外要求フラグは、HTTP(S)通信および IrOBEX 外部接続では true を、ScratchPad およびリソースでは false を設定します。

2.2.3 i アプリ API

次の各項では、`com.docomostar` パッケージ、及び各サブパッケージ中の i アプリ基本 API に属するクラスについて説明します（一部のクラスには、i アプリオプション API に属するメソッドを含んでいるものがあります）。

なお、1 つの i アプリの中で、`com.docomostar` 配下の API と `com.nttdocomo` 配下の API を混在させて利用することはできません。Star プロファイルに対応する i アプリでは、`com.docomostar` 配下の API のみ、DoJa プロファイルに対応する i アプリでは `com.nttdocomo` 配下の API のみがそれぞれ利用可能です。

com.docomostar パッケージ

アプリケーションの基底クラス、アプリケーションの管理クラス、およびイベントハンドリングに関連するクラスが含まれています。

このパッケージに含まれるインタフェース：

名前	説明
<code>StarEventListener</code>	アプリケーションに対するイベントのリスナを定義します。 <code>StarEventObject</code> で定義されているイベントを受信するために用います。 イベントを受信するには、このリスナを <code>StarApplication.addEventListener(int, StarEventListener)</code> に登録する必要があります。

このパッケージに含まれるクラス：

名前	説明
AudioOutputEvent	Star アプリケーションイベントの 1 つで、携帯電話の音声出力先が変更されたことを表すイベントクラスです。イベント発生時に、音声出力先が本体スピーカあるいは外部音声機器（有線イヤホン、Bluetooth イヤホンなど）のどちらであるかを取得するメソッドが定義されています。 【Star-1.5】本クラスは Star-1.5 プロファイルで新設されました。
ClockTickEvent	Star アプリケーションイベントの 1 つで、ネイティブの時計の時刻が変化した事象を表すイベントクラスです。イベント発生時のネイティブの時計の時刻を取得するメソッドが定義されています。
StarApplication	StarApplication クラスは、i アプリの雛型クラスです。i アプリのメインクラス（i アプリの起動時に最初に呼び出されるクラス）は StarApplication のサブクラスでなければなりません。 このクラスはアプリケーションのライフサイクルを定義します。ライフサイクルの状態変更をアプリケーションに通知するメソッドや、アプリケーションから状態変更を要求するためのメソッドなどが定義されています。
StarApplicationManager	アプリケーションマネージャを表すクラスです。原則としてアプリケーションマネージャとやりとりするメソッドが定義されています。ただし、アプリケーションライフサイクルに関するメソッドや、リスナ登録に関するメソッドなど、アプリケーション基底として最低限必要なメソッドについては StarApplication に定義されています。
StarEventObject	アプリケーションにおけるイベント（Star アプリケーションイベント）を表すクラスです。 このクラスに定義されているイベントタイプは、StarApplication.addEventListener(int, StarEventListener)および StarApplication.addWakeUpEvent(int)に登録することができます。StarEventListener によって通知されるイベントは、全てこのクラスを継承しています。また、具体的なイベントタイプは getType() を呼び出すことで知ることができます。Star アプリケーションイベントについては 3.7 項を参照してください。

com.docomostar.device パッケージ

このパッケージには、携帯電話に搭載されている周辺デバイスを制御するためのクラスが含まれています。

このパッケージに含まれるクラス：

名前	説明
Camera	携帯電話に搭載されているカメラ機能呼び出して、静止画または動画を撮影する機能にアクセスする手段を提供します。実際の撮影は、i アプリが起動したカメラ機能に対してユーザーがシャッター操作を行うことにより行われます。動画撮影をサポートするかどうかはメーカーにより異なります。
CodeReader	携帯電話に搭載されたカメラデバイスを使用して、バーコード（JAN コードや 2 次元バーコード）を読み取るコード認識機能を提供します。
IrRemoteControl	赤外線リモコン機能を提供します。IrRemoteControl クラスは、IrRemoteControlFrame クラスで表現されるリモコン信号を赤外線ポートから送り出すための、送出器としての役割を持ちます。

IrRemoteControlFrame	赤外線リモコン機能から送出される、外部機器の制御内容を示すデータフレームを表現します。一般的に、どのようなデータフレームを受信した際にどのような制御を行うかは、その外部機器のメーカーが独自に決定しています。開発者は、対象とする外部機器の仕様に合致したデータフレームを送出するようアプリケーションプログラムを設計する必要があります。
DeviceException	DeviceException は、主に device パッケージや device パッケージの使用するネイティブ機能などが動作しているときに異常を検出した際にスローされる実行時例外クラスです。

com.docomostar.fs パッケージ

このパッケージには、内部ストレージのファイルシステムにアクセスするクラスが含まれています。

このパッケージに含まれるクラス：

名前	説明
InternalStorage	内部ストレージを表現するクラスです。内部ストレージはデータ BOX 内に設けられた領域で ScratchPad とは異なります。内部ストレージ内のフォルダやファイルの取得や削除、内部ストレージで利用できる空き容量を取得する機能を提供しています。 【Star-2.0】本クラスは Star-2.0 プロファイルで新設されました。
InternalStorageFile	内部ストレージに作成されたファイルを表現するクラスです。i アプリは InternalStorageFolder 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、フォルダ内のファイルを削除したり、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 InternalStorageFile クラスのオブジェクトとして表されます。実際にファイル内のデータにアクセスするには、このクラスのオブジェクトから com.docomostar.io.FileEntity クラスのオブジェクトを取り出して使用します。 【Star-2.0】本クラスは Star-2.0 プロファイルで新設されました。
InternalStorageFolder	内部ストレージに作成されたフォルダを表現するクラスです。初めて内部ストレージにアクセスすると、i アプリ固有のフォルダが生成されます。i アプリは、i アプリ固有フォルダ配下にフォルダやファイルを生成することができます。 【Star-2.0】本クラスは Star-2.0 プロファイルで新設されました。
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既に存在している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	ファイルシステムに空き容量がないためにフォルダやファイルの作成に失敗したことを示す例外です。
FolderException	フォルダへのアクセスが禁止されていたり、存在しないフォルダへのアクセスを指示したなどの理由で、フォルダにアクセスできなかったことを示す例外です。 【Star-2.0】本クラスは Star-2.0 プロファイルで新設されました。

com.docomostar.io パッケージ

com.docomostar.io パッケージでは、URL の形式で指定されたネットワーク資源に接続して入出力を行うためのインタフェースおよびクラスが提供されます。これらは、HTTP/HTTPS プロトコルによるネットワーク接続、および赤外線ポートを介した OBEX 外部接続をサポートします。これらの機能は Generic Connection フレームワーク上に構築されています。

またこのパッケージには、テキストのストリーム処理を効率よく行うために、入出力クラスを補完するクラスも含まれています。

なお、Generic Connection フレームワークについては 2.2.2 項を参照してください。

このパッケージに含まれるインタフェース：

名前	説明
ClientObexConnection	OBEX クライアントとしてのアクセスを定義するインタフェースです。 ClientObexConnection は、i アプリ実行環境において OBEX サーバー、OBEX クライアントの共通インタフェースを定義する ObexConnection のサブインタフェースです。
HttpConnection	HTTP 接続へのアクセスを定義するインタフェースです。 javax.microedition.io.Connector.open() メソッドに HTTP または HTTPS の URL を指定することで、本インタフェースの実装インスタンスが得られます。 HttpConnection は Generic Connection フレームワークにおける javax.microedition.io.ContentConnection インタフェースのサブインタフェースです。
ObexConnection	OBEX サーバー、OBEX クライアントの共通インタフェースを定義します。このインタフェースを直接実装したインスタンスをアプリケーションプログラムで使用することはありません。 このインタフェースには、レスポンスステータスやオペレーションコード定数、OBEX サーバーと OBEX クライアントで共通のアクセスサメソッドが定義されています。 ObexConnection は Generic Connection フレームワークにおける javax.microedition.io.StreamConnection インタフェースのサブインタフェースです。
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。
ServerObexConnection	OBEX サーバーとしてのアクセスを定義するインタフェースです。 ServerObexConnection は、i アプリ実行環境において OBEX サーバー、OBEX クライアントの共通インタフェースを定義する ObexConnection のサブインタフェースです。

このパッケージに含まれるクラス：

名前	説明
BufferedReader	バッファリングされたテキスト入力ストリームをサポートします。このクラスは、Java 言語標準ライブラリにおける java.io.BufferedReader クラスのサブセットに相当する機能を提供します。
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。

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。
IrClientObexConnection	IrOBEX クライアントとしての赤外線ポートへのアクセスを定義するクラスで、 ClientObexConnection インタフェースを実装しています。 javax.microedition.io.Connector.open() メソッドに URL “obex://irclient” を指定することで、本クラスのインスタンスが得られます。
IrServerObexConnection	IrOBEX サーバーとしての赤外線ポートへのアクセスを定義するクラスで、 ServerObexConnection インタフェースを実装しています。 javax.microedition.io.Connector.open() メソッドに URL “obex://irserver” を指定することで、本クラスのインスタンスが得られます。
PrintWriter	フォーマットされたオブジェクト表現を出力するためのテキスト出力ストリームをサポートします。このクラスは、Java 言語標準ライブラリにおける java.io.PrintWriter クラスのサブセットに相当する機能を提供します。
ConnectionException	このクラスは、 com.docomostar.io パッケージにおける入出力処理で異常が検出されたときにスローされます。例外の詳細は getStatus() メソッドで取得します。

HttpConnection を使用したクライアント/サーバプログラミングについては第5章で解説しています。また IrOBEX による外部接続プログラミングについては第10章で解説しています。

com.docomostar.lang パッケージ

このパッケージに含まれるクラス：

名前	説明
MemoryManager	アプリケーション実行メモリ（ヒープメモリ）の使用状況をモニタリングするためのユーティリティ機能を提供するクラスです。
IllegalStateException	操作やメソッド呼び出しが行われてはならない状態の時にそのような操作やメソッド呼び出しが行われると、実行時例外として IllegalStateException がスローされます。
IterationAbortedException	配列などに対する繰り返し処理が、何らかの例外発生によって中断したことを示す例外クラスです。このクラスのメソッドを使用して、繰り返し処理失敗の直接的な原因となった例外の内容を知ることができます。
UnsupportedOperationException	サポートされない操作やメソッドを実行すると、実行時例外として UnsupportedOperationException がスローされます。

com.docomostar.media パッケージ

このパッケージに含まれるインタフェース：

名前	説明
MediaImage	MediaImage は静止画像または動画像を表現します。 MediaImage の実装オブジェクトは MediaManager.getImage() メソッドを使って取得します。 MediaImage は VisualPresenter オブジェクトを使って再生します。使用可能な形式については4.5項を参照してください。

MediaMovie	MediaMovie は動画データを表現します。MediaMovie の実装オブジェクトは、MediaManager.getMovie() メソッドを使って取得します。MediaMovie は NativeMoviePlayer オブジェクトを使って再生します。使用可能な形式については、4.5 項を参照してください。
MediaResource	メディアデータインタフェース (MediaImage、MediaSound、MediaMovie) が継承しなければならないインタフェースを定義します。MediaManager を使用して取得した MediaResource インスタンスを実際に使用する前には use() メソッドを、使用した後は unuse() メソッドをそれぞれ呼び出す必要があります。MediaResource インスタンスをそれ以上使用しない場合は、dispose() メソッドを呼び出してインスタンスを破棄すべきです。
MediaSound	MediaSound はサウンドを表現します。MediaSound の実装オブジェクトは MediaManager.getSound() メソッドを使って取得します。MediaSound は AudioPresenter オブジェクトを使って再生します。サウンドデータとして使用可能な形式については、4.5 項を参照してください。

このパッケージに含まれるクラス：

名前	説明
EncodedImage	Canvas や Image の描画内容を、ImageEncoder クラスによりバイトイメージ (JPEG 形式など) にエンコードした結果を保持するクラスです。i アプリは本クラスを使用して、描画内容のバイトイメージを取り出したり、メディアイメージに再変換して取り出したりすることができます。
ImageReference	イメージリファレンスを定義します。イメージリファレンスクラスは画像データを直接グラフィックスコンテキストに描画する場合に使用します。オブジェクトを生成する際に、ヒープ上に画像データの展開をしないので、MediaImage を使用した場合と比較してメモリの消費量を抑えることができます。(画像データの展開は描画命令実行時に行なわれ、描画完了と共に解放されます。)
MediaManager	MediaResource を管理するためのクラスです。MediaManager はメディアデータ (MediaImage、MediaSound、MediaMovie) のファクトリクラスです。URL などのロケーション文字列、ストリーム、バイト配列のいずれかを指定してメディアデータを生成することができます。
MediaException	media パッケージのコンポーネントなどが実行時例外を起こしたことを示すために投げられます。例外の詳細は、getStatus メソッドで得ることができます。

com.docomostar.net パッケージ

このパッケージに含まれるクラス：

名前	説明
URLDecoder	このクラスには、URL エンコード形式を String にデコードするユーティリティメソッドが含まれています。入力データは SJIS コードをベースとしているものと見なされます。
URLEncoder	このクラスには、String を URL エンコード形式 (MIME タイプ x-www-form-urlencoded で使用されるエンコード形式) にエンコードするユーティリティメソッドが含まれています。エンコードの際、String は SJIS コードに変換されます。

com.docomostar.security パッケージ

このパッケージには、証明書やデジタル署名など、セキュリティに関する機能を提供するクラスが含まれます。

このパッケージに含まれるクラス：

名前	説明
PKCS7SignedData	デジタル署名付きデータを表現するクラスです。PKCS7Signer によりデジタル署名されたデータは、このクラスのオブジェクトとして表現されます。
PKCS7Signer	デジタル署名付きデータを生成する機能を提供するクラスです。任意のデータをこのクラスに与えることで、そのデータにデジタル署名を施すことができます。
PKCS7SignerInfo	デジタル署名に含まれる署名者情報を表現するクラスです。
X509Certificate	X509 証明書を表現するクラスです。i アプリの内部では、証明書はこのクラスのオブジェクトとして表現されます。
PKCS7SignatureException	デジタル署名に関するエラーが発生したことを示す例外です。デジタル署名処理に失敗した場合や、デジタル署名の検証に失敗した場合などに発生します。

com.docomostar.system パッケージ

このパッケージには、電話帳機能やブックマーク、スケジューラなど携帯電話のネイティブ機能を i アプリから利用するための次のクラスおよびインタフェースが含まれています。なお、このパッケージにはトラステッド i アプリにしか使用が許可されないクラスやメソッドが多く含まれています。ここではそれらの機能については解説しません。

このパッケージに含まれるインタフェース：

名前	説明
PhoneBookConstants	電話帳管理機能に関連する定数をまとめたインタフェースです。このインタフェースは PhoneBook クラスおよび PhoneBookParam クラスに実装されています。開発者がこのインタフェースの実装クラスを作成する必要はありません。

このパッケージに含まれるクラス：

名前	説明
ApplicationStore	携帯電話ネイティブの i アプリ管理機能にアクセスする機能を提供します。このクラスを使用することで、i アプリから携帯電話にダウンロードされている他の i アプリを起動（ランチャーモード）する際に必要となるエントリ ID を取得することができます。
Bookmark	携帯電話ネイティブのブックマーク管理機能にアクセスする機能を提供します。このクラスを使用することで、携帯電話にブックマーク項目を新規登録することができます。
ContentsDownloader	携帯電話ネイティブのコンテンツ取得保存機能にアクセスする機能を提供します。このクラスを使用することで、フル楽曲など i アプリで直接取り扱うことが難しいような大容量コンテンツをダウンロードし、携帯電話ネイティブの保存領域に直接保存することができます。
【Star-1.1】本クラスは Star-1.1 プロファイルで新設されました。	

ImageStore	携帯電話ネイティブの画像データ管理機能にアクセスする機能を提供します。このクラスを使用することで、携帯電話の画像データ保存領域からの画像の読み込みと、画像の新規登録を行うことができます。
Launcher	i アプリから他のアプリケーションを連携起動する機能を提供するクラスです。i アプリが他のアプリケーションを起動しようとする、ユーザー確認のダイアログが表示されます。ユーザーが許可をすると、目的のアプリケーションが起動されます。基本的には現在実行中の i アプリを終了して目的のアプリケーションを起動しますが、起動するアプリケーションの種類によっては i アプリをサスペンドしたまま目的のアプリケーションを起動します。
MovieStore	携帯電話ネイティブの映像データ（i モーション）管理機能にアクセスする機能を提供します。このクラスを使用することで、携帯電話の映像データ保存領域への映像データの新規登録を行うことができます。なお、映像データ保存領域からの映像データの取得は i アプリオプション API に該当し、メーカーによりサポート状況が異なります。
NativeMoviePlayer	携帯電話ネイティブのムービープレイヤーにアクセスする機能を提供します。このクラスを使用することで、携帯電話のネイティブのムービープレイヤーを用いて動画の再生を行うことができます。
Phone	Phone クラスはダイヤラーを表し、携帯電話のネイティブの通話機能にアクセスする手段を提供します。 i アプリから通話機能呼び出すと、アプリケーションマネージャはユーザーに通話発信を行ってもよいか確認を求めます。この確認を省略することはできません。
PhoneBook	携帯電話ネイティブの電話帳管理機能にアクセスする機能を提供します。このクラスを使用することで、携帯電話に電話帳エントリを新規登録することができます。
PhoneBookGroup	携帯電話ネイティブの電話帳管理機能のうち、電話帳グループ管理機能にアクセスする機能を提供します。このクラスを使用することで、携帯電話に電話帳グループを新規登録することができます。このクラスで作成した電話帳グループに対して、 PhoneBook クラスを使用して電話帳エントリを新規登録することもできます。
PhoneBookParam	新規登録用の電話帳エントリを表現するクラスです。このオブジェクトに名前や電話番号などの電話帳エントリデータを設定し、 PhoneBook クラスで携帯電話に登録します。
PhoneSystem	このクラスは、プラットフォームのネイティブ資源にアクセスするときに使用します。プラットフォームのネイティブ資源には、ディスプレイ上に表示されるアイコン情報（メールアイコンなど）を含みます。 また、このクラスでは携帯電話の個体識別情報（製造番号など、機体毎にユニークな情報）を取得するためのメソッドを備えています。
Schedule	携帯電話ネイティブのスケジューラにアクセスする機能を提供します。このクラスを使用することで、携帯電話にスケジュールを新規登録することができます。
ScheduleParam	スケジュールデータの新規登録用のパラメータを定義します。 生成したオブジェクトに対して、スケジュールの件名等を設定後、 Schedule.addEntry(ScheduleParam) メソッドによる登録を行うことができます。
InterruptedOperationException	i アプリが呼び出したネイティブアプリケーション機能をユーザーが操作している際に、通話着信などの外部イベントの発生によりその操作が中断されたことを示す例外です。
StoreException	携帯電話ネイティブのアプリケーションが管理するデータ領域にアクセスしようとしたときに、空きエリア不足などのアクセスエラーが発生したことを示す例外です。

注意事項：

- **com.docomostar.system** パッケージのクラスには、携帯電話のネイティブ機能と呼び出してユーザーデータを新規登録する機能が多く含まれています。携帯電話ユーザーは、携帯電話の設定によりこれらの機能を無効化することができます。またこれらの機能呼び出した場合、i アプリ実行環境はそれらのデータを登録することについてユーザーに確認を求めます。これらの機構によりユーザーの許可が得られなかった場合、そのデータを登録することはできません。

com.docomostar.ui パッケージ

このパッケージに含まれるインタフェース：

名前	説明
Audio3DListener	3D サウンド制御に関する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イベントリスナを定義するインタフェースです。本プロファイルでは、 SoundMotion （移動する定位）における移動完了イベントが定義されています。
Audio3DLocalization	定位を表すインタフェースです。3D サウンド制御機能では定位を表現する方法（クラス）が複数ありますが、どのクラスもこのインタフェースを実装しています。 なお、このインタフェースではメソッドやフィールドは定義されていません。
ComponentListener	コンポーネントイベントを受け取るリスナーインタフェースです。コンポーネントイベントの処理を行うクラスはこのインタフェースを実装する必要があります。リスナーオブジェクトは Panel.setComponentListener() メソッドを使用して Panel に登録します。コンポーネントイベントが発生するとリスナーオブジェクトの componentAction() メソッドが呼び出され、イベントのタイプ、イベントが発生したコンポーネント、イベントパラメータが引数として渡されます。
FocusManager	フォーカスマネージャとは、 Panel 上に配置されたコンポーネント間のフォーカス移動を管理するオブジェクトです。このインタフェースはフォーカスマネージャクラスが実装するインタフェースです。フォーカスマネージャに関する指定が特に行われていない場合、デフォルトのフォーカスマネージャクラスが使用されます。現在までのプロファイルでは、デフォルトフォーカスマネージャ以外のフォーカスマネージャを規定していません。
Interactable	ユーザーが操作できるコンポーネント（ Button 、 ListBox 、 TextBox など）が実装するインタフェースです。このインタフェースでは、コンポーネントへのフォーカス設定や操作可否設定を行うためのメソッドが定義されています。
KeyListener	高レベル API において、個別のキー操作を待ち受けるためのリスナーインタフェースです。本インタフェースの実装オブジェクトは Panel に登録して使用します。このインタフェースには、キープレスイベントとキーリリースイベントに対応する keyPressed() と keyReleased() の 2 つのメソッドが定義されています。
LayoutManager	レイアウトマネージャとは、 Panel 上へのコンポーネントの配置を管理するオブジェクトです。このインタフェースはレイアウトマネージャクラスが実装するインタフェースです。i アプリ実行環境では、レイアウトマネージャに関する指定が特に行われていない場合に使用されるデフォルトレイアウトマネージャ、および HTML 風のレイアウトを行う HTML レイアウトマネージャが定義されています。
MediaListener	このインタフェースは、マルチメディアデータを再生するメディアプレゼンターからイベントを受け取るためのメディアリスナーが実装するインタフェースです。このインタフェースで定義されている mediaAction() メソッドでは、メディアプレゼンターの再生開始・終了・停止などのイベントを受け取ることができます。

MediaPresenter	メディアデータの再生オブジェクトに実装されるインタフェースを定義します。プレゼンタークラスは、メディアデータを実際の内容に従って再生します。サポートされるメディアデータのタイプは端末に依存します。端末がサポートしないメディアデータが設定されると、 MediaException がスローされます。
SoftKeyListener	高レベル API において、携帯電話に 4 つ用意されているファンクションキー操作を待ち受けるためのリスナーインタフェースです。本インタフェースの実装オブジェクトは Panel に登録して使用します。このインタフェースには、ファンクションキープレスイベントとファンクションキーリリースイベントに対応する softKeyPressed() と softKeyReleased() の 2 つのメソッドが定義されています。

このパッケージに含まれるクラス：

名前	説明
AnchorButton	AnchorButton コンポーネントは、HTML のアンカー風のボタン（ボタンとしての操作が可能な下線付きテキスト）を表現します。テキストの前に見出し用のイメージ画像を指定することもできます。また、テキストは複数行に渡ることができます。アンカーボタン押下により処理を行う場合は、 ComponentListener を実装したオブジェクトを、 Panel の setComponentListener() メソッドによりリスナー登録します。アンカーボタンが操作されると、リスナーは BUTTON_PRESSED イベントを受け取ります。
Audio3D	各オーディオプレゼンタ毎に 3D サウンド制御を行うためのコントローラです。 Audio3D オブジェクトを使用して 3D サウンド制御の有効/無効を切り替えたり、定位の制御を行います。i アプリの中でいくつの仮想音源を使用できるか(3D サウンド制御リソース数) はメーカーにより異なります。詳細は 13.2 項を参照してください。
AudioPresenter	このクラスは、サウンドメディアデータ (MediaSound) を再生するときを使用します。 AudioPresenter は高レベル API 使用時、低レベル API 使用時のいずれの場合も使用することができます。
Button	Button コンポーネントはラベル付きボタンを表現します。ボタン押下により処理を行う場合は、 ComponentListener を実装したオブジェクトを、 Panel の setComponentListener() メソッドによりリスナー登録します。ボタンが操作されると、リスナーは BUTTON_PRESSED イベントを受け取ります。
Canvas	Canvas は画面上の空白の矩形域です。i アプリはこの領域に描画したり、この領域でユーザーの入力イベントをトラップしたりすることができます。開発者は Canvas を継承するサブクラスを定義して使用します。i アプリから Canvas に描画を行うには、 Canvas クラスの paint() メソッドをオーバーライドします。
Component	Component はグラフィックスで表現されたユーザーインタフェース部品であり、画面に表示したり、これを使ってユーザーと対話したりすることができます。 Component のサブクラスには Button 、 ListBox 、 TextBox などがあります。 Component は高レベル API に属し、 Panel と組み合わせて使用します。
Dialog	Dialog は、タイトル、テキストメッセージ、1 つ以上のボタンを備えたトップレベルウィンドウです。 Panel および Canvas のいずれの上にも Dialog を表示することができます。
Display	Display は画面領域を管理するクラスです。 Display のすべてのメソッドは static であり、そのインスタンスが作成されることはありません。 Display クラスは、画面 (Panel または Canvas) を切り替えたり、画面上に表示されている表示可能オブジェクトに入力イベントを渡したりします。このクラスは、低レベルイベントを受け取り、それを適切な表示可能オブジェクトに渡します。そして、このオブジェクトがイベントの処理方法を決定します。さらに、 Display には、表示領域の幅と高さや、キーやイベント処理で使用するさまざまな定数といったデバイス情報が含まれています。

Font	Graphics コンテキストや Component、Dialog に文字を表示する際のフォントを表します。どのようなフォントがサポートされるかは、メーカーにより異なります。
Frame	Frame は、画面を表現する abstract クラスです。携帯電話のディスプレイに一度に表示できる内容は、1 つの Frame インスタンスに対応します。Frame のサブクラスには Canvas、Dialog、Panel などがあります。画面の切り替えは、Display.setCurrent() メソッドを、切り替えたい画面に対応する Frame オブジェクトを引数として呼び出すことで行います（ただし Dialog のみ、Dialog.show() メソッドにより表示します）。
FrameChangedEvent	Star アプリケーションイベントの 1 つで、カレントのフレームが非カレントに切り替わったことを表すイベントクラスです。非カレントに移行するフレームを取得するメソッドが定義されています。
Graphics	Canvas および Image への描画とそれに関連する操作をカプセル化します。なお Image への描画は、その Image が Image.createImage() メソッドにより新規に作成された Image インスタンスである場合に限り可能です。
HTMLLayout	HTML 風のレイアウトを行うためのレイアウトマネージャです。Panel に追加された順にレイアウトしていく点ではデフォルトレイアウトマネージャと同様ですが、HTML レイアウトマネージャでは、用意されたメソッドにより HTML の改行、段落およびアライメントに相当する指定を行うことができます。
Image	Image クラスは静止画像を表現する abstract クラスです。静止画像から MediaImage インスタンスを作成し、getImage() メソッドを呼び出すことで Image の実装インスタンスを取得することができます。また Image.createImage() メソッドにより、新規に Image インスタンスを作成することもできます。静止画像として使用可能なデータ形式については、4.5 項を参照してください。
ImageButton	ImageButton コンポーネントは、静止画像を貼り付けることのできるボタンを表現します。イメージボタン押下により処理を行う場合は、ComponentListener を実装したオブジェクトを、Panel の setComponentListener() メソッドによりリスナー登録します。イメージボタンが操作されると、リスナーは BUTTON_PRESSED イベントを受け取ります。
ImageEncoder	Canvas や Image の描画内容を、JPEG 形式などのバイトイメージにエンコードする機能を提供します。エンコードされた結果は EncodedImage インスタンスとして返されます。各機種で共通的に使用可能なエンコード形式は JPEG 形式です。
ImageLabel	ImageLabel コンポーネントは、Panel に静止画像を表示するときに使用します。Image インスタンスを ImageLabel に設定し、それを Panel 上に配置することができます。
ImageMap	ImageMap は、何種類かの小さなイメージを縦横に並べて見かけ上大きなイメージを作成する機能を提供します。小容量の画像データで大きな背景画像などを構成するのに適しています。
Label	Label コンポーネントは、Panel にテキストを表示するときに使用します。Label は、1 行の読み取り専用テキストを表現します。テキストはアプリケーションプログラムからは変更することはできませんが、ユーザーが直接編集することはできません。
ListBox	ListBox コンポーネントは、複数のテキスト項目からなるスクロールリストを表示するときに使用します。インスタンス化時の指定によって、開発者は ListBox の外観や機能を選択することができます。
Palette	PalettedImage と組み合わせて使用する、色パレットを表すクラスです。
PalettedImage	GIF データなどパレット形式の画像データを元に作成される、色パレットを指定することができるイメージです。

Panel	Panel は、高レベル API 使用時において、コンポーネントを画面に表示するためのコンテナクラスです。Panel が表すスペースには任意のコンポーネントを付加することができます。Panel は、同時に 1 つしか画面に表示できません。
Sprite	スプライト機能において、画面上を移動する個々のスプライトを表現します。Sprite はそれ単独では表示させることはできず、SpriteSet と組み合わせて使用します。
SpriteSet	スプライト機能において、複数のスプライトを束ねる役割を持ちます。アプリケーションプログラムは必要な数のスプライトをスプライトセットに属させ、スプライトセットを描画します。
TextBox	TextBox コンポーネントは、テキストの入力や表示を行うときに使用します。TextBox は、ユーザーが任意の文字列を入力するための唯一のコンポーネントです。
Ticker	Ticker コンポーネントは、左方向に自動スクロールするテキストを表示するために使用するコンポーネントです。このコンポーネントは、テキストをティッカーテープ形式（i モード HTML の MARQUEE タグ相当）で表示します。
VisualPresenter	VisualPresenter コンポーネントは、画像メディアデータ (MediaImage) を高レベル API で再生するときに使用するプレゼンタークラスです。
UIException	UIException は、UI パッケージや UI パッケージの使用するネイティブ機能などが動作しているときに異常を検出した際にスローされる実行時例外クラスです。

com.docomostar.ui.graphics3d パッケージ

このパッケージには、3D グラフィックス描画機能を提供するためのクラスおよびインタフェースが含まれています。

このパッケージに含まれるインタフェース：

名前	説明
Graphics3D	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。Image に対応するグラフィックスコンテキストがこのインタフェースをサポートするかどうかはメーカーにより異なります。

このパッケージに含まれるクラス：

名前	説明
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。フレームは全ての 3D オブジェクトが内部に保持している「現在時刻」と対応しており、後述の Figure オブジェクトに現在時刻を設定することで、3D モデルにポーズを取らせることができます。

DrawableObject3D	レンダリング可能な 3D オブジェクトの基底となるクラスです。このクラスを継承する Figure クラス、 Group クラス、 Primitive クラスの各オブジェクトは、 Graphics3D.renderObject3D() メソッドによるレンダリング対象とすることができます。また、3D 空間内で 2 つのオブジェクト同士が衝突しているかどうかを簡易に判定するためのメソッドや、ブレンド機能、テクスチャ歪み補正機能などのメソッドを備えています。
Figure	3D モデルのデータを保持するクラスです。このオブジェクトはレンダリング可能であり、 Graphics3D.renderObject3D() メソッドにレンダリング対象として指定することができます。
Fog	フォグ効果を与えるためのデータを保持するクラスです。線形フォグモードと指数フォグモードをサポートします。
Group	3D オブジェクトの集合体であるグループを表現するクラスです。このオブジェクトはレンダリング可能であり、 Graphics3D.renderObject3D() メソッドにレンダリング対象として指定することができます。複数の Figure オブジェクトや Primitive オブジェクトを属させて一度にレンダリングを行わせたり、 Fog オブジェクトや Light オブジェクトを属させてグループ内のレンダリングに効果を与えたりすることができます。
Light	光源データを保持するクラスです。いくつまでの光源をサポートするかはメーカーにより異なります (getMaxLights() メソッドで確認することができます)。環境光源、平行光源、点光源、スポット光源をサポートします。
Object3D	全ての 3D オブジェクトの基底となるクラスです。3D オブジェクトとは、3D グラフィックスのレンダリング結果に影響を与えるオブジェクト全般を指します。3D オブジェクトには、フィギュアやプリミティブのように直接レンダリングの対象物とすることのできるオブジェクトと、ライトやフォグのようにレンダリング結果に効果を与える性質を持つオブジェクトに大別されます。このクラスでは 3D オブジェクトの生成方法として createInstance() を規定していますが、3D オブジェクトの種類によってはコンストラクタを使用してオブジェクトを生成するものもあります。
Primitive	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などを格納するための、プリミティブ配列を保持するクラスです。このオブジェクトはレンダリング可能であり、 Graphics3D.renderObject3D() メソッドにレンダリング対象として指定することができます。
Texture	テクスチャのデータを保持するクラスです。テクスチャオブジェクトを使用することで、モデルマッピングに使用するテクスチャと環境マッピングに使用するテクスチャの両方を取り扱うことができます。なお、本機能における環境マッピング用テクスチャの使用法は i アプリ拡張 API の高レベル 3D グラフィックス描画機能における使用方法から変更されています。本機能では、環境マッピング用のテクスチャは 3D グラフィックスコンテキスト (Graphics3D オブジェクト) ではなく各モデルマッピング用テクスチャオブジェクトに設定して使用します。

com.docomostar.ui.sound3d パッケージ

このパッケージには、3D サウンド機能を提供するためのクラスおよびインタフェースが含まれています。これらのクラスやインタフェースは、3D サウンドを制御するためのコントローラである **com.docomostar.ui.Audio3D** クラスと組み合わせて使用します。

このパッケージに含まれるインタフェース：

名前	説明
SoundPosition	聴者の位置と仮想音源の位置によって表される、固定された定位を表すインタフェースです。このインタフェースを実装したクラスでは、聴者の位置と仮想音源の位置を指定することによって両者の位置関係を表します。

このパッケージに含まれるクラス：

名前	説明
CartesianListener	CartesianPosition クラスで定位を表現する際に、デカルト座標系上に置く聴者を表現するクラスです。デカルト座標系で定位を表すために必要となる聴者の向きの設定は、このクラスを使用して行います。
CartesianPosition	3D サウンドの制御において、x 軸、y 軸、z 軸の 3 軸から構成されるデカルト座標系を使用して聴者と仮想音源の位置関係（定位）を表します。
PolarPosition	極座標系を使用して聴者と仮想音源の位置関係を表します。極座標系ではその原点にある決められた方向を向いた聴者がいるものとし、そこからの方位角、仰角、距離を指定して仮想音源の位置を表現します。
SoundMotion	SoundMotion クラスは「移動する定位」を表現するためのクラスです。移動開始からの経過時間とその時点での定位を経路情報として設定することで、その経路に沿って仮想音源が移動していくようにユーザーに感じさせることができます。

com.docomostar.ui.util3d パッケージ

このパッケージでは、3D グラフィックス機能や 3D サウンド機能を使用する上で必要となる数値演算やベクトル演算などを行うためのユーティリティ的な機能が提供されます。

このパッケージに含まれるクラス：

名前	説明
FastMath	3D 関連機能での使用を前提とした高速数値演算ユーティリティです。本プロファイルに対応する携帯電話の i アプリ実行環境では浮動小数点数演算が使用できますが、このクラスでは浮動小数点数演算を内部的に整数演算に置き換えます。このため、通常の浮動小数点数演算より大きな誤差を含む可能性はありますが高速に演算を行うことができます。
Transform	3 次元アフィン変換用の行列を扱うクラスです。このクラスでは、4 行 4 列の行列でアフィン変換行列を表現します。
Vector3D	3 次元ベクトルを表すクラスです。内積・外積計算、正規化といったユーティリティメソッドを備えています。座標系上の位置や向きなどを表現するために使用されます。

com.docomostar.util パッケージ

このパッケージに含まれるインタフェース：

名前	説明
EventListener	i アプリ API で取り扱うイベントのリスナーインタフェースは、全てこのインタフェースを継承しています。

TimeKeeper	i アプリ API では、一定時間が経過したことをイベントとして i アプリに通知するタイマー機能をサポートしています。タイマークラス (<code>com.docomostar.util.Timer</code>) はこのインタフェースを実装しています。 なお、タイマーの解像度 (処理可能な最小のタイマー間隔) は機種によって異なりますが、<code>getMinTimeInterval()</code> メソッドにより、その機種でタイマーイベントを発生させることのできる最小間隔を調べることができます。また <code>getResolution()</code> メソッドでは、<code>getMinTimeInterval()</code> の値から何 msec 刻みでタイマーイベントを発生させることのできるかを調べることができます。つまり <code>getMinTimeInterval()</code> の値を x、<code>getResolution()</code> の値を y とすると、その機種では $x+ny$ msec (n は整数) の間隔でタイマーイベントを発生させることができます。アプリケーションプログラムがタイマーの時間間隔としてこの式に合致しない値を指定した場合、この式に合致する最小の値が指定されたものとして切り上げられます。
TimerListener	このインタフェースは <code>com.docomostar.util.Timer</code> の通知するタイマーイベントを取得するリスナーを実装するためのインタフェースです。

このパッケージに含まれるクラス：

名前	説明
Base64	バイト列の Base64 形式文字列へのエンコード機能、および Base64 形式文字列のバイト列へのデコード機能を提供するユーティリティクラスです。
JarInflater	JAR 形式のファイルイメージから、それに含まれるエントリを伸張して取り出す機能を提供します。JAR ファイルイメージは、バイト配列またはバイトストリームのいずれかの形式のものを使用することができます。
MessageDigest	任意のバイトデータからメッセージダイジェスト (ハッシュ値) を求めるための機能を提供します。現在のプロファイルでは MD5 および SHA-1 アルゴリズムがサポートされています。
ScheduleDate	スケジュール時刻を表現するためのクラスです。このクラスは i アプリからネイティブアプリケーションにスケジュール時刻を設定する際に使用します。このクラスを使用して、スケジューラやアラーム設定などのネイティブアプリケーションにスケジュールを登録することができます。
Timer	タイマークラスです。このクラスは、ワンショットタイマーとインタバルタイマーをサポートします。 <code>TimerListener</code> オブジェクトを登録することでタイマーイベントを受け取ることができます。タイマーイベントが発生すると、リスナーオブジェクトの <code>timerExpired()</code> メソッドが呼び出されます。タイマーの有効期間は i アプリの起動から終了までの全体に渡り、画面が切り替わってもタイマー動作は継続されます。ただし、i アプリの実行が中断された場合、タイマーは停止します。
JarFormatException	Jar ファイル形式のエラーが発生したことを示す例外クラスです。この例外は、 <code>JarInflater</code> クラスにおいて不正な形式のデータを処理しようとした場合に発生します。

StarApplication

`com.docomostar.StarApplication` クラスは、全ての i アプリの雛型となるクラスです。i アプリのメインクラス（ADF の `AppClass` キーで指定される、i アプリ起動時に最初に呼び出されるクラス）はこのクラスを継承していなければなりません。

`StarApplication` クラスには、主に i アプリの起動から終了までのライフサイクルに関わるメソッドやリスナー登録に関わるメソッドが定義されています。（ライフサイクルに関する詳細は、3.6 項を参照してください）

- i アプリの実行は、`StarApplication.started(int)`メソッドから開始されます。開発者はこのメソッドをオーバーライドして、i アプリの処理をクラスに実装します。
`StarApplication.started(int)`メソッド呼出し後に、`StarApplication.activated(int)`メソッドが呼び出されます。
- i アプリの実行をアプリが自ら中断するには、`StarApplication.suspend()`メソッドを呼び出します。`suspend()`メソッド呼出し後、一定時間経過後に自動的にレジュームすることを指示するための `StarApplication.setWakeupTimer()` も定義されています。
- i アプリの実行を終了するには、`StarApplication.terminate()`メソッドを呼び出します。
- i アプリの実行中に、電話着信などの優先度の高い外部イベントが発生した場合、i アプリの実行は中断されて外部イベントに対応した処理が行われます。また、その後外部イベントの処理が終了すると、i アプリの実行は自動的に再開されます。i アプリの実行が再開される時には `StarApplication.activated(int)`メソッドが呼び出されます。開発者はこのメソッドをオーバーライドすることで、i アプリ実行再開時に必要な処理を実装することができます。また、`StarApplicationManager.getSuspendInfo()`メソッドを使用すると、前回 i アプリが中断するきっかけとなった理由と、中断中に発生した事象の情報を取得することができます。
- i アプリ実行中に発生したイベントを検知するには、検知したいイベントの種類とイベントリスナーを指定して `StarApplication.addEventListener()`メソッドを呼び出します。
- i アプリ実行中に、現在の i アプリの表示状態や実行タイプを取得することができます。

注意事項：

- i アプリの中で、`StarApplication` およびそのサブクラスを直接インスタンス化してはいけません。これらは i アプリ実行環境がインスタンス化します。
- 明示的に i アプリメインクラスのコンストラクタを定義する場合、必ずアクセス指定子 `public` を指定してください。また i アプリ実行環境は、メインクラスのインスタンス化時に引数のあるコンストラクタは使用しません。明示的にメインクラスのコンストラクタを定義する際は、必ず引数なしのコンストラクタを設けるようにしてください。
- CLDC 仕様では、`java.lang.Runtime` クラスおよび `java.lang.System` クラスに `exit()`メソッドが定義されています。しかし i アプリ実行環境では、i アプリの実行を終了するためにこれらのメソッドを使用することはできません（これら呼び出した場合には、`SecurityException` がスローされます）。i アプリの実行を終了するためには、`StarApplication.terminate()`メソッドを呼び出すようにしてください。

テキスト変換

i アプリでは、テキスト処理に日本語文字セットと ASCII 文字セットを使用します。`java.io` パッケージには、Unicode 文字ストリームと、非 Unicode テキストのバイトストリームとの間で変換を行うクラスが含まれています。`byte` ストリームを `character` ストリームに変換するには `InputStreamReader` クラスを、`character` ストリームを `byte` ストリームに変換するには `OutputStreamWriter` クラスをそれぞれ使用します。テキスト変換に関する詳細は、第 6 章を参照してください。

ScratchPad

携帯電話の i アプリ向け記憶装置モデルを ScratchPad と呼びます。ScratchPad に接続するには、Connector クラスの `open()` メソッドを使用します。このメソッドに ScratchPad の URL を渡すと ScratchPad への接続がオープンされ、Generic Connection フレームワークで定義されている `javax.microedition.io.StreamConnection` インタフェースの実装インスタンスが返されます。ScratchPad 領域の読み取りや書き込みには、`StreamConnection` インタフェースを使用します。ScratchPad の利用に関する詳細は、第 7 章を参照してください。

第3章 設計上の考慮事項

この章では、利用可能なシステム資源を効率的に使用し、無線通信環境に適合した使いやすい i アプリを設計するためのガイドラインを提供します。

3.1 i アプリ対応携帯電話の特性

携帯電話の表示画面やアプリケーション実行メモリ、アプリケーション記憶メモリは、携帯電話の物理的なサイズをできるだけ小さくし、バッテリー寿命をできるだけ長くするため、パーソナルコンピュータに比べ非常に小さくなっています。さらに、プロセッサもパーソナルコンピュータより遅い速度で動作します。液晶（LCD）画面の電力消費量は非常に少ないのですが、リフレッシュ速度は他のタイプのディスプレイに劣ります。テキスト入力には携帯電話のキーパッドを使って行い、しかも各キーに複数の文字が割り当てられているため、テキストの入力はユーザーにとって煩雑な操作です。したがって、テキスト入力は最小限に留めるべきです。

i アプリを活用したサービスを設計する場合、このようなプラットフォーム固有の特性を考慮して i アプリの設計を行う必要があります。次の各節では、これらの問題を設計の観点から検討し、使いやすい i アプリを設計するためのガイドラインを提供します。

3.2 メモリの問題

次に、3つのiアプリに対してアプリケーションマネージャが記憶領域（JAR ファイルとScratchPad の保存領域）を割り当てる例を示します。

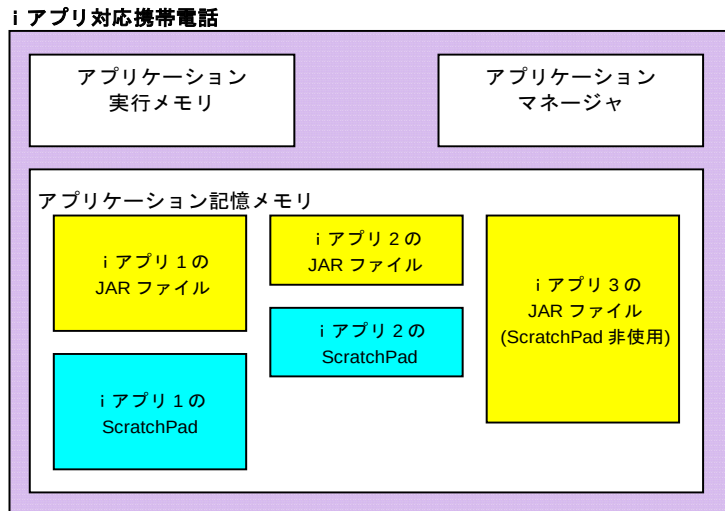


図 6 i アプリ対応携帯電話のメモリ

開発者にとって、メモリには次の2つの制約があります。

- 限られたアプリケーション実行メモリ。i アプリはこの領域で実行されます。実行時に生成されるオブジェクトもここに格納されます。
- 限られたアプリケーション記憶メモリ。ここにはi アプリのJAR ファイルと、そのi アプリに対応するScratchPad が格納されます。

アプリケーション実行時サイズに対する絶対的な制約は、i アプリが実行される携帯電話に搭載されているアプリケーション実行メモリの量によります。搭載されるアプリケーション実行メモリの量、およびi アプリ実行時にアプリケーション実行メモリがどのように使われるかはメーカーにより異なります。

またアプリケーション記憶サイズは、以下のように制限されています（以降、1K バイト=1024 バイトとします）。これらのサイズの制限を越えるi アプリは携帯電話に正しくダウンロードできないのでご注意ください。

プロファイル	フルアプリ		フル+ミニアプリ		ミニアプリ	
	JAR ファイル	ScratchPad	JAR ファイル (*2)	ScratchPad (*1)	JAR ファイル	ScratchPad
Star-1.x	JAR ファイルと ScratchPad の合計で 2048K バイト		JAR ファイルと ScratchPad の合計で 2048K バイト		50K バイト	200K バイト
Star-2.x	JAR ファイルと ScratchPad の合計で 10240K バイト		JAR ファイルと ScratchPad の合計で 10240K バイト		50k バイト	200K バイト

(*1)フル+ミニアプリでは、フルアプリとミニアプリでScratchPadを共有することができます。ただし、ミニアプリからアクセス可能なScratchPadは、0番目のもの(scratchpad:///0)だけであり、またそのサイズは200K バイト以下でなければなりません。

(*2)フル+ミニアプリにおける、フルアプリおよびミニアプリのJAR ファイルのサイズの制限は、それぞれ該当プロファイルにおけるフルアプリあるいはミニアプリのJAR ファイルと同様です。フル+ミニアプリにおけるJAR ファイルの構成については16.5.2 項を参照してください。

注意事項：

- ScratchPad サイズは、メーカーによっては規定のサイズより大きいサイズをサポートする場合があります。またメーカーの採用する領域管理方式によっては、新たにインストールしようとしている i アプリの ScratchPad サイズが、インストール済み i アプリが使用している ScratchPad サイズの合計サイズにより制限を受ける場合もあります。いずれの場合にも、i アプリが ADF の **SPsize** キーで宣言した容量の ScratchPad を確保できない場合、その i アプリは携帯電話にダウンロードすることはできません。
- 携帯電話に 100K バイトを超える JAR ファイルをダウンロードできるようにするためには、i アプリの配布に使用する Web サーバーがレンジリクエストによるバイト単位の部分的 GET に対応していなければなりません。（詳細は、第 17 章を参照してください）

3.2.1 メモリ使用量の最小化

ここでは、大きなカテゴリとして、アプリケーション実行メモリとアプリケーション記憶メモリに分けて説明します。

アプリケーション実行メモリ（ヒープメモリ）

通常、アプリケーション実行メモリを占める最大のコンポーネントは実行可能プログラム（i アプリ実行環境および i アプリを構成するクラス群）そのものです。i アプリとそれに必要なすべてのオブジェクトが携帯電話のアプリケーション実行メモリに入りきらない場合は、実行可能プログラムのサイズか生成されるオブジェクトセットのサイズ、またはその両方を削減する以外に方法はありません。

実行可能プログラムのサイズを削減する方法については、3.2.2 項を参照してください。

生成されるオブジェクトのサイズを削減する場合は、次に示す内容を検討します。

- メモリにはできるだけ少ないオブジェクトを保持する。または生成するオブジェクトの数およびサイズを最小限に抑える。
 - アプリケーションプログラムの分割を検討する（例えば複数のレベルをもつゲームの設計では、レベル毎にアプリケーションプログラムを分割し複数の i アプリとするなど）。
 - 可能な限り **static** を使用する。
 - 過度のメソッド呼び出しのネストや再帰を避けてスタックの深さを最小限に抑える。
 - メソッドに渡すパラメータの数を最小限に抑える。
 - クラスの数、および個々のクラスのサイズを最小限に抑える。
 - スレッドの多用を避ける。
 - サイズがあらかじめ決められたクラスを作成する。**Hashtable**、**ByteArrayInputStream**、**ByteArrayOutputStream**、**Vector** など、サイズが動的に大きくなるクラスを使用すると、メモリのフラグメンテーションを引き起こすことがあります。
 - 必要がなくなったメディアデータやイメージは速やかに破棄（**dispose()**）する。
 - イメージリファレンスを通じて描画を行う。
 - ガベージコレクタを頻繁に呼び出す、もしくは確保したメモリやオブジェクトの再利用を図りガベージが発生しにくいプログラムとする。特に搭載メモリ量の少ない i アプリ実行環境では、アプリケーション実行メモリ上に発生したフラグメンテーションが原因で、一度にまとまったサイズのメモリを確保できなくなるケースが発生します（**OutOfMemoryError**、**UIException**、**ConnectionException** などの例外が発生したり、i アプリの動作が不安定になるなどの問題として現れます）。この場合、本項の対策を行うことで問題が解決することがあります。
- なお携帯電話に搭載される i アプリ実行環境の実装によっては、アプリケーションプログラムからの指示（**System.gc()**メソッド呼び出し）ではガベージコレクタの制御を行えない場合があります。このような実装では、アプリケーションプログラムから **System.gc()**メソッドを呼び出しても無視され、

システムが必要と判断した契機のみでガベージコレクタが呼び出されます。このような実装では、本項に関する対策方法が制約される場合があります。

生成されるオブジェクトサイズを可能な限り小さくし、アプリケーション実行メモリを効率的に使用しているにもかかわらず、まだメモリが不足する場合は、次の内容を検討してください。

- 変数に保存しているデータを ScratchPad に移す。
- メモリを消費しやすい処理をクライアント側からサーバー側に移す。
- サーバー側からデータをダウンロードする場合、一括してダウンロードするのではなく、必要最小限のものに絞り部分的にダウンロードする。

`com.docomostar.lang.MemoryManager` クラスを使用すると、アプリケーション実行メモリの使用状況をモニタリングすることができます。

CLDC で規定されている `java.lang.Runtime` クラスにも `freeMemory()` や `totalMemory()` といった類似の機能がありますが、`MemoryManager` クラスではそれら機能に加え、アプリケーション実行メモリ上の最大連続空き容量（フラグメントによって分断されていない連続空きエリアのうち、最大のエリアの容量）を知ることができます。また、i アプリ実行環境が複数のアプリケーション実行メモリ区画を持っている（オブジェクト用のヒープの他にメディアデータ専用のヒープを持っているなど）場合には、区画ごとにこれらの情報をモニタリングすることができます。

アプリケーション記憶メモリ

i アプリを保存するために、アプリケーション記憶メモリの中に以下の2つのブロックが確保されます。

- JAR ファイルの格納に割り当てられるブロック
- ScratchPad に割り当てられるブロック

JAR ファイル

JAR ファイルには、できるだけコンパクトにした必須のファイルだけを入れる必要があります。i アプリオプション API や i アプリ拡張 API を使用するなどの理由で携帯電話の機種ごとに異なるモジュールを i アプリに入れる必要がある場合は、機種ごとに別の i アプリ（別の JAR ファイル）を作成することを検討してください（機種別にダウンロードを制御する方法については第 17 章を参照）。このようにすれば、携帯電話にダウンロードされる JAR ファイルに、不要なエントリが含まれることはありません。

また JAR ファイルには画像やサウンドなどのリソース（第 7 章参照）を含めることができますが、これは JAR ファイルサイズに大きな影響を与えます。使用頻度の低いリソースは、i アプリ実行時に通信機能を使用して入手することを検討してください。

なお、i モードサービスでは通信料金にパケット課金システムが採用されています。JAR ファイルを小さくすることは、i アプリダウンロード時のユーザーの通信コストを削減することにもつながります。

ScratchPad

必要以上に大きな ScratchPad は指定しないでください。さらに、どの位の ScratchPad 容量を割り当てるにしろ、できる限り効率的に使用してください。ScratchPad サイズは、i アプリの ADF に指定します（詳細は 16.5.1 項を参照）。なお、i アプリが使用可能な ScratchPad サイズには、3.2 項で解説した制限があります。

なお、ScratchPad にどのような物理デバイス（ハードウェア）を使用するかはメーカーにより異なります。必要以上に大きな ScratchPad を割り当てると、機種によっては ScratchPad へのアクセス速度が低下して i アプリの実行パフォーマンスにも影響を与えることがありますので注意してください。

3.2.2 クラスファイルサイズ

アプリケーションプログラムをコンパイルしたクラスファイルにはデバッグのための情報が含まれることがあり、この情報によりクラスファイルのサイズが増大することがあります。定義するクラスの数をもっと抑えようとするとともに、コードを**-g:none** オプションでコンパイルすることでデバッグ情報を取り除き、クラスファイルのサイズを小さくするようにしてください。

なお、NTT ドコモの提供する i アプリ開発環境のビルド機能を利用して i アプリをビルドする場合、コンパイル時には**-g:none** オプションが設定されています。i アプリ開発環境の詳細については、第 16 章を参照してください。

3.3 使い易さと GUI の設計

この節では、簡単に直感的に使用できる i アプリを作成するためのガイドラインを示します。携帯電話のようなプラットフォームでは、一般的にはたくさんのキー操作を必要とする複雑な i アプリは敬遠され、よりキー操作が少ない i アプリが求められることが想定されます。注意すべき点を次に列挙します。

- i アプリの全体を通して一貫した Look&Feel を維持します。
- 物理的な画面領域が小さいため、1つの画面に表示する UI コンポーネントの数を最小限に抑えます。特にユーザー操作可能な UI コンポーネント（ボタンなど）を1つの画面に多数表示した場合、フォーカス移動操作などの手間により、ユーザーにとって操作が煩わしく感じられることがあります。
- UI レイアウト管理はメーカーによって異なりますが、ほとんどの場合、デフォルトのレイアウト管理は簡単なフローレイアウトです。UI レイアウトをシンプルにし、i アプリ実行画面の外観が、メーカー間で大きく変わらないようにします。
- スクロールを最小限に抑えます。なお、**Panel** は垂直方向のスクロールをサポートしますが、**Canvas** を使用する場合は画面のスクロールはアプリケーションプログラム側で制御する必要があります。
- ファンクションキーは、ユーザー操作を簡単にする便利な方法です。複雑なユーザー操作が発生するような画面では、ファンクションキーをショートカットキーのように使用することを検討してください。たとえば多くの入力用コンポーネント部品が配置された画面において、ファンクションキーはどのコンポーネントにフォーカスがある状態でも確定動作を行わせるためのキーとして使用することができます。
- 携帯電話でのテキスト入力は煩雑なため、入力する文字数を最小限に抑えます。可能な限り選択肢を表示し、その中からユーザーが選択する形の入力方式にします。
- よく行われる操作は、多数のメニュー階層を移動したり、多数のキー操作をしなくても使用できるように画面構成を設計します。

3.3.1 ターゲットユーザー向けの設計

電話はだれでも使いますが、コンピュータはだれもが使うわけではありません。したがって、UI コンポーネント間のフォーカス移動やスクロールという考え方は、全てのユーザーにとって直感的にわかるわけではありません。広範なユーザーに使用してもらうためには、ユーザーインタフェースはできるだけ直感的に操作できなければなりません。i アプリのユーザーインタフェースを設計する際は、それを使用すると見込まれているターゲットユーザー層を見極める必要があります。

3.3.2 i アプリの操作方法の設計指針

ユーザー操作は、それがどのくらい頻繁に行われるかや、ユーザーにとってどの程度重要なものであるかによって、いくつかのカテゴリに分類する必要があります。一般には、頻繁に行われる操作はより簡単に使えるようにすべきです。たとえば、銀行残高の確認は口座間の振り替えよりも頻繁に行われます。したがって理想的には、残高の確認は振り替えよりも少ないキー操作でできるようにすべきです。

一方、個人識別 ID の入力など、いくつかのキー操作が必要な操作は、それが重要な機能である限り（個人識別 ID の例ではユーザーの銀行口座などへの不正アクセスの防止）、ユーザーに受け入れられます。さらに i アプリを携帯電話にインストールして最初に使い始める際に、多少煩雑ではあってもその後の操作を簡単にする目的で、一定の情報入力をユーザーが行うようにするという点も検討の余地があります。

操作には一貫性がなければなりません。項目の選択方法は、どの画面でも一貫しているべきです。ファンクションキーは、誤操作や使いづらさを避けるため、i アプリ全体を通して一貫した考え方に基づいて使用できなければなりません。また最も重要かつ頻繁に使用される操作は、最も少ないキー操作でできるようにすべきです。

3.3.3 スクロール

理論的には、物理画面よりも大きな仮想画面を設計することができます。しかしそのような画面では、画面全体を見たり UI コンポーネント間をフォーカス移動するには、多くのスクロールが必要になります。スクロールは必ずしもレスポンスのよい画面操作ではないため、ユーザーはスクロールの多い画面を使いづらいと感じる可能性があります。スクロールは最小限に抑えることを目標にすべきです。デフォルトレイアウトマネージャまたは HTML レイアウトマネージャが設定されている **Panel** は垂直方向のスクロールをサポートします。

3.3.4 キーおよびファンクションキー

i アプリでは以下の標準的なキーを使用することができます。

- ・ 上下左右キーおよび選択（決定）キー
- ・ 0～9 までの数字キーおよび#キー、*キー

メーカーによっては、i アプリオプションのカテゴリに属する機能としてこれら以外の特別なキーをサポートする場合があります。

また、i アプリ対応携帯電話には上記以外にファンクションキーが設けられます。ファンクションキーはフルアプリでは 4 つ、ミニアプリでは 2 つ使用することができます。ファンクションキーの外観や場所はメーカーによって異なります。前述の通り、ファンクションキーを使用するとユーザーインターフェースの使用感が大幅に改善されることがあります。i アプリの設計ではファンクションキーの使用を考慮すべきですが、特定の外観や場所に依存することは避けるようにしてください。

3.3.5 データ入力

ユーザーがデータを入力する手段は、原則的に携帯電話のキーパッドだけです。したがって、テキストデータを入力するよりも数字データを入力の方がより簡単に行えます。これはテキストデータの場合、1文字を入力するために複数回のキー操作が必要になるためです。テキスト入力は最小限に抑えるべきです。

3.3.6 パスワード入力

パスワードの入力は次のいずれかの方法で行います。

- パスワードを表示しない場合は、`TextBox` のコンストラクタで表示モードパラメータに `TextBox.DISPLAY_PASSWORD` を渡します。すべての入力文字は「*」で表示されます。
- パスワードを表示する場合は、`TextBox` のコンストラクタで表示モードパラメータに `TextBox.DISPLAY_ANY` を渡します。この場合、入力された文字はすべて画面上に表示されます。

1つ目の方法を使用する場合、一般的にはパスワードには数字のみを使用します。携帯電話ではPCと異なり、数字以外の文字をパスワード入力することはユーザーにとって非常に困難な作業となります。パスワードに使用する文字を数字のみとする場合、

`TextBox.setInputMode()` メソッドを使用して `TextBox` の初期入力モードを数字入力モードとしておくことで、ユーザーの誤操作を防ぐことができます。

パスワードに数字以外の文字（テキスト文字）を許可する場合は、2つ目の方法を使用します。

3.3.7 スレッドの使用

スレッドは、ネットワーク入出力やUIでの対話など、長時間を要するブロック操作を行うときには非常に便利です。しかし、どのようなマルチスレッドシステムでも、アクティブスレッド間のコンテキスト切り替えにはオーバーヘッドが必要です。携帯電話の処理能力は限られているため、ユーザー操作をストレスなく可能にするためにスレッドの数は最小限にすべきです。

また、アプリケーションプログラム側でマルチスレッドプログラミングを行っていなくても、システムがイベント処理などのためにスレッドを生成することがあります。どの機種でも動作するiアプリを作成するには、スレッドセーフなプログラム設計が必要です。

注意事項：

- iアプリ実行環境では、以下のような状況などでダイアログやポップアップが表示されます。
 - ・ アプリケーションプログラム自身が `Dialog` クラスを使用してダイアログを表示させる場合
 - ・ HTTP(S)通信において、Webサーバーから認証（BASIC 認証）を求められた場合
 - ・ HTTPS 通信においてサーバー証明書の内容に問題があり、ユーザーに警告を示す場合
 - ・ IrOBEX 外部接続機能により赤外線ポートを使用する場合
 - ・ アプリケーション連携機能により、他のアプリケーションの起動や呼び出しをユーザーに確認する場合

これらのダイアログやポップアップは排他的にのみ表示可能であり、複数のスレッドからこれらを同時に表示させることはできません。これらを複数のスレッドから同時に表示させようとした際の振る舞いはメーカーにより異なります。開発者は、これらの状況が複数スレッドで同時に発生することのないようアプリケーションプログラムを設計する必要があります。

- マルチスレッドシステムにおけるスレッド切り替え制御の詳細は、iアプリ実行環境の実装によって異なる場合があります。あるシステム上で意図通りに動作するマルチスレッドプログラムが、別のシステム上ではスレッド切り替え制御の違いにより、意図通りの動作をしない可能性があります。（タイミングの調整に空ループを用いるなど、システムによる暗黙のスレッド切り替えを期待したアプ

リケーションプログラムでは、スレッド間のタイミングが合わずに動作不良を引き起こしたり、アプリケーションプログラムが期待するようなタイミングや頻度でスレッド切り替えが発生せず実行性能低下やフリーズが発生したような見え方をすることがあります。）

このような問題が発生した場合は、原因となっているロジック（上記、タイミング調整の例では空ループを実行している箇所）に明示的に `Thread.sleep()` や `Thread.yield()` などスレッド制御を伴う命令を組み込むことで動作が改善する場合があります。

3.4 セキュリティ

クライアントとサーバー間で機密情報を送受信する i アプリを設計する場合には、HTTPS プロトコルを使用することを検討します。ただし、クライアント側でこのメカニズムを利用する際には、CPU 負荷が高くなることに注意してください。これは、各 SSL セッションの始めにクライアント側（この場合 i アプリ）でマスターキーを生成する必要があるのに加え、データが処理されてサーバーとのデータ送受信が行われるたびに、これらのキーに基づくデータの暗号化や暗号化解除が必要になるためです。

HTTPS プロトコルを利用した通信については、5.3 項を参照してください。

なお、i アプリがサーバーとの通信を行う場合、その接続先は i アプリのダウンロード元とプロトコル、サーバーアドレス、ポート番号が同じでなければなりません。したがって、HTTPS プロトコルを使用する i アプリは HTTPS プロトコルを使用してダウンロードされている必要があります。

3.5 無線ネットワークでの操作

i アプリは、通信接続の中断がごく普通に発生する環境でも使用されることに留意してください（ユーザーは携帯電話の電波が届く区域を時に頻繁に出入りします）。また i モードサービスではパケット課金システムが採用されているため、i アプリが通信を行う場合は、必要なデータだけを送受信するようにしてください。

クライアント/サーバー型の i アプリの設計においては、小さいデータパケットトランザクションをいくつも送受信する（切断によるデータの喪失を最小限にする）か、1 つの大きなデータパケットを送受信する（通信オーバーヘッドを最小限にする）かについてバランスをとる必要があります。

接続の確立、テスト、終了に関する問題や、トランザクションやセッションの管理に関する問題の詳しい説明は、第 5 章を参照してください。さらに、次項「i アプリのライフサイクル」を参照してください。

注意事項：

- 通信を行う i アプリでは、i アプリ開発者は通信エラーや通信の結果得られたデータのエラーをリカバーするために通信処理のリトライを行う処理を組み込むことがあります。

通信に関するエラーを i アプリが検出して自動的にリトライを行うことは、通信品質（電波状況）が頻繁に変化する携帯電話上で i アプリを快適に使用できるようにする観点で有用な手段です。しかし i アプリが無限に通信のリトライを行わないよう、そのような処理を組み込む場合は必ず繰り返し回数に上限を設けるようにしてください。

i モードサービスではパケット課金システムが採用されており、通信料金は実際に行われた通信量に比例して課金されます。1 回の通信サイズが小さくても、上限なく通信のリトライが行われると短時間のうちに非常に高額の課金が行われる可能性があります。

通信処理、または通信処理を含むアプリケーションロジックのリトライは多くても数回程度までとし、その回数で目的の処理を完遂することができなかった場合はダイアログ表示などによりユーザーにエラーを通知するような仕組みとすることを強く推奨します。

3.6 i アプリのライフサイクル

アプリケーションマネージャは i アプリ実行中に通話の着信があると、動作している i アプリを自動的に中断します。そしてユーザーが通話を終了すると、Resume（再開）イベントを i アプリに送信して実行を再開します。このように、i アプリを実行中の携帯電話上で、より優先度の高い別の機能が動作する必要がある場合、i アプリ実行の中断や再開が行われる場合があります。本項ではこのようなアプリの起動から終了に至る実行状態遷移(ライフサイクル)について説明していきます。

3.6.1 i アプリの実行状態遷移

下図に i アプリの実行状態の遷移を示します。

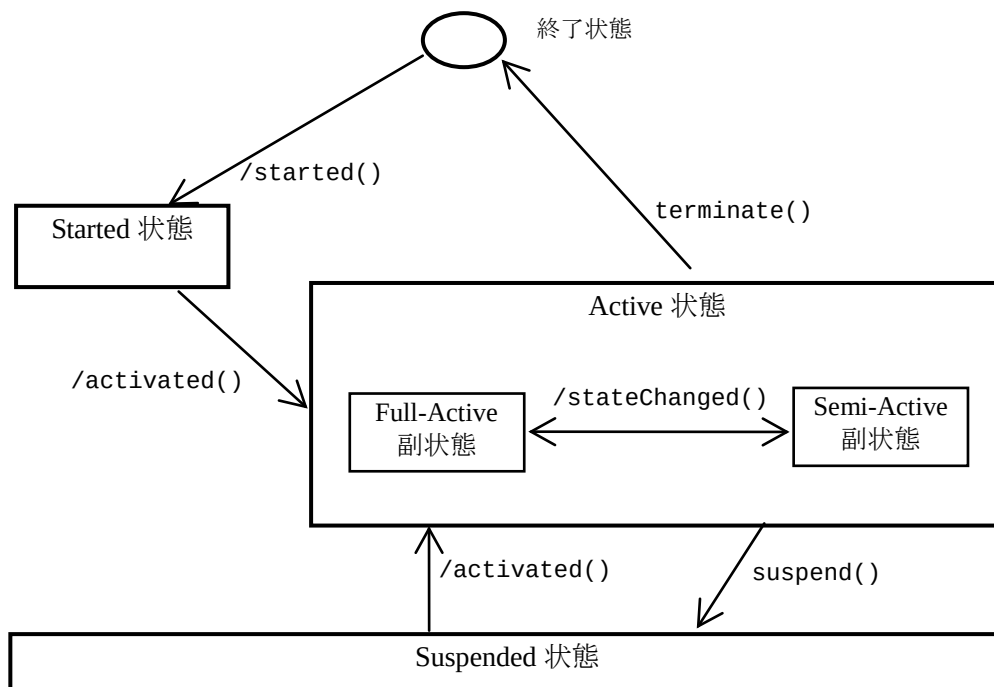


図 7 i アプリの実行状態の遷移

上図中の、矩形は i アプリの実行状態を表し、矢印は状態遷移（イベント）を表します。矢印に付記されたメソッド名のうち、/で始まるものは、その状態遷移によってそのコールバックメソッドの呼び出しが生じることを表します。/で始まらないメソッドは、そのメソッドを i アプリから呼び出すことで、対応する状態遷移が発生することを表しています。

以降、各状態について説明します。

Started 状態

アプリケーション起動直後の状態です。この状態に入ると `StarApplication.started()` メソッドが呼び出されます。

アプリケーション開発者は `started()` メソッドをオーバーライドすることで、その i アプリの起動時処理を行なわせることができます。

`started()` メソッドの実行が完了すると Started 状態は終了し、i アプリは次の Active 状態に遷移します。

Active 状態

アプリケーション実行中の状態です。この状態は副状態として「Full-Active 副状態」「Semi-Active 副状態」を持ちます。副状態は、複数の i アプリが同時に動作する場合があるミニアプリのために導入された考え

方です。フルアプリにおいては、常に Full-Active 副状態であり Semi-Active 状態に遷移することはありません。

Active 状態にあるミニアプリの副状態が変化すると `StarApplication.stateChanged()` が呼び出されます。副状態を切り替えるためのメソッドはありません。ミニアプリの状態遷移については第 9 章を参照してください。

Full-Active 副状態

リソースを制限なく利用可能な状態です。フルアプリは常にこの状態です。

Semi-Active 副状態

リソースの利用制限状態です。この状態になる場合があるのはミニアプリのみであり、`WidgetView` で一覧表示画面になったときに遷移します。

Suspended 状態

アプリケーションが一時停止している状態です。i アプリを実行中の携帯電話上で、より優先度の高い別の機能が動作する必要があると、i アプリは `Suspended` 状態に遷移します。また、`StarApplication.suspend()` メソッドを呼ぶことによってもこの状態に遷移します。この状態に入る条件やその影響については次項を参照してください。

Terminated 状態

アプリケーションが終了し、動作していない状態です。`StarApplication.terminate()` メソッドを呼ぶことによってもこの状態に遷移します。

3.6.2 ライフサイクルイベント処理

i アプリの実行は `StarApplication.started()` メソッドから開始されます。アプリケーション開発者はこのメソッドをオーバーライドして、i アプリの起動時処理を実装します。`started()` メソッドの実行が完了するまでが `Started` 状態です。`started()` メソッドが終了すると、i アプリの状態は `Active` 状態に遷移し、`StarApplication.activated()` メソッドが呼び出されます。

i アプリの実行中に、電話着信などの優先度の高い外部イベントが発生した場合、i アプリの実行は中断 (`Suspended` 状態) します。i アプリが `Suspended` 状態になる条件は、電話着信時以外にもいくつか存在します。以下に、i アプリの中断、再開が行われるケースの例を示します。

- i アプリ実行中に電話着信があった場合 (i アプリが通信実行中の場合、携帯電話の i モード着信設定により処理が中断されない場合もあります)。
- i アプリ実行中に、他のアプリケーションが起動された場合。これには、i アプリ実行中に時刻アラームなどのネイティブアプリケーションが割り込んで自動起動した場合や、アプリケーション連携機能により i アプリから他のネイティブアプリケーションや i アプリを起動した場合を含みます。なお、起動するアプリケーションの種類によっては、呼び出し元の i アプリは中断から再開されることなく終了します。
- i アプリから、ユーザー操作を伴うネイティブ機能呼び出した場合。これには、アプリケーション連携機能を中心に、システムによりユーザー確認が求められる多くの機能が含まれます。
- ユーザーが i アプリの強制終了操作を行い、終了確認ダイアログでユーザーが強制終了をキャンセルした場合。
- 携帯電話が上記以外に i アプリを中断、再開するための機構 (中断・再開キーなど) を持っており、その操作が行われた場合。
- `WidgetView` 上のミニアプリが、ユーザーの `WidgetView` 操作によって一時中断する場合 (`WidgetView` の一覧表示状態から他の状態に遷移した場合など)

i アプリの実行が再開されると、i アプリメインクラス

(`com.docomostar.StarApplication` を継承したクラス) の `activated()` メソッドが呼び出されます。また、`Active` 状態の i アプリ (ミニアプリのみ) の副状態が変化したとき、`StarApplication.stateChanged()` メソッドが呼び出されます。開発者は、これらのメソッドをオーバーライドすることで状態の変化時の処理を実装することができます。

スタンドアロンの電卓アプリケーションなど、ある種の i アプリでは、開発者が中断/再開のために特別なステップを行う必要はありません。これは、i アプリが中断されても内部的には何の変化も起らないことと、アプリケーションプログラムの処理進行が完全にユーザー生成のイベント（キーの操作）によって駆動されるからです。しかし特定の種類の i アプリでは、実行を再開する際に特別なステップを行う必要があります。一般に、次のような状況では、i アプリの実行を再開する際に何らかのアクションをとる必要があります。

- クライアント/サーバー型の i アプリでは、i アプリの実行が再開されたら通信結果をチェックし、必要に応じて通信のリトライを行います。これは、i アプリが通信処理を行っている際に中断・再開が発生すると、その通信処理は失敗するためです。
i アプリが中断、再開する際には、その i アプリが中断する前に動作していたタイマー (Timer) は停止しています。これらのタイマーは、i アプリ再開時に明示的に再スタートを行う必要があります。
- i アプリ中断時に **AudioPresenter** がサウンドを再生していた場合、**AudioPresenter** は動作を停止します。i アプリ再開後も自動的に再生を開始することはありません。この場合、i アプリから明示的に再生指示 (**play()**) を行うことにより、サウンドデータの再生はデータの先頭から行われます。なお、**VisualPresenter** が動画を再生しているときに i アプリの実行が中断された場合は、動画再生は一時的に停止しますが、i アプリの再開とともに中断時のフレームから動画再生も再開されます。
- i アプリの中断・再開により **Canvas** が再表示される際、i アプリ実行環境からその **Canvas** の **paint()** メソッドが 1 回呼び出されることが保証されます。しかしその **paint()** メソッド呼び出しで全画面が完全には復元されないようなケース (アプリケーションプログラムがイベントに応じて差分描画のみ行うようになっているなど) では、どの程度システムが画面を復元できるかは機種により異なります。i アプリの再開時にシステムが **Canvas** の描画内容をすべて復元できた場合は、**paint()** メソッドは呼び出されません。
- i アプリの中断・再開が発生した場合、再開した i アプリが適切なプロンプトメッセージを表示して一旦停止することが望ましい場合があります。たとえば、多くのアクションゲームのようにユーザーとの即座対話が必要なケースでは、i アプリが再開されてからユーザーの操作の準備が整うまでの間に若干の時間的な猶予を持たせることで、再開時の i アプリの操作性を向上させることができます。
- i アプリの中断・再開が発生した際に、中断の理由と中断中に発生した事象を取得するための **StarApplicationManager.getSuspendInfo()** メソッドが用意されています。

注意事項：

- i アプリが通信を行っている場合、電話の着信で i アプリが中断するか否かは、携帯電話の i モード着信設定の内容に従います。i モード通信中に電話の着信を許可しない設定になっている場合は、i アプリが通信を行っている間は電話の着信を受けても i アプリの実行は中断されません。
- i アプリの実行が中断された際に未処理のイベントがイベントキューにキューイングされていた場合、i アプリ再開時にはそれらは全て破棄されます。
- **activated()** メソッドは **Active** 状態への遷移が発生したことを i アプリに通知するためのメソッドです。従ってこのメソッドは、中断された i アプリが再開される場合だけでなく、**Started** 状態が終了して **Active** 状態に遷移するとき (**started()** メソッドが終了した直後) にも呼び出されます。
- i アプリ実行再開時のスレッド動作順序、およびスレッドスイッチ制御の詳細は i アプリ実行環境のスレッド制御に委ねられるため、**activated()** メソッドの呼び出しは i アプリ実行の再開において必ず最初に行われるとは限りません。また、ライフサイクルイベントと他のイベントが同時に発生するような状況において、イベントの処理順序は保証されません。ライフサイクルイベントと次項で解説する **Star** アプリケーションイベントなどが同時に発生するようなケース (例えばサスペンド中の i アプリが、起床イベントとして登録された **Star** アプリケーションイベントの発生を受けてレジュームする場合など) で、アプリケーション開発者は特定の実行順に依存した処理を記述するべきではありません。

3.7 Star アプリケーションイベント

Star アプリケーションには、前項に記載したライフサイクルイベントや第4章で解説されるユーザーインタフェースのイベントの他に、i アプリメインクラスに通知されるイベント（Star アプリケーションイベント）がいくつか存在しています。

i アプリは、`StarApplication.addEventListener()` メソッドを利用してあらかじめイベントリスナオブジェクトを登録することで、Star アプリケーションイベントを受け取ることができます。

Star アプリケーションイベントには以下のようなものが定義されています。

`StarEventObject` クラスでは、これらのイベントタイプを表す定数が定義されています。

`StarEventObject` クラスは、Star アプリケーションイベントのイベントリスナに通知されるイベントオブジェクトの基底クラスです。イベントの種類によって、このクラスを直接利用するものと、そのイベント専用を用意されたサブクラスを使用するものがあります。

- **STAR_CLOCK_UPDATED**

ネイティブの時計の時刻が変化したことを示すイベントタイプです。ネイティブの時計の時刻が毎分正秒（00 秒）を指したときにこのイベントが発生します。

このイベントタイプで使用する `ClockTickEvent` クラスには、このイベントが発生した時刻をその日の 0 時 0 分 0 秒からの経過秒数として取得する `getSecondOfDay()` メソッドが定義されています。

- **STAR_FRAME_CHANGED**

フレームが切り替わったことを示すイベントタイプです。アプリケーションから

`Display.setCurrent()` メソッドが呼び出されたことによりカレントのフレームが切り替わったときにこのイベントが発生します。

このイベントタイプで使用する `FrameChangedEvent` クラスには、非カレントに移行したフレームを取得する `getHidingFrame()` メソッドが定義されています。

なお、このイベントタイプは起床イベントとして登録することはできません。

- **STAR_STATECHANGE_CLAM_OPEN**

折りたたみ型またはフリップ型の携帯電話において、折りたたみ状態またはフリップ状態が開かれたことを示すイベントタイプです。このイベントは折りたたみ型またはフリップ型の携帯電話でのみ発生します。サポートされていない携帯電話においては、イベントリスナや起床イベントとして登録することはできません。

- **STAR_STATECHANGE_CLAM_CLOSE**

折りたたみ型またはフリップ型の携帯電話において、折りたたみ状態またはフリップ状態が閉じられたことを示すイベントタイプです。このイベントは折りたたみ型またはフリップ型の携帯電話でのみ発生します。サポートされていない携帯電話においては、イベントリスナや起床イベントとして登録することはできません。

- **STAR_FACE_STATECHANGE_MINI_FOCUSED**

ミニアプリにおいて、表示状態が `Focused` 状態に遷移したことを示すイベントタイプです。このイベントはミニアプリにおいてのみ発生します。フルアプリからはイベントリスナや起床イベントとして登録することはできません。

- **STAR_FACE_STATECHANGE_MINI_SELECTED**

ミニアプリにおいて、表示状態が `Selected` 状態に遷移したことを示すイベントタイプです。このイベントはミニアプリにおいてのみ発生します。フルアプリからはイベントリスナや起床イベントとして登録することはできません。

- **STAR_FACE_STATECHANGE_MINI_UNFOCUSED**

ミニアプリにおいて、表示状態が `Unselected` 状態に遷移したことを示すイベントタイプで

す。このイベントはミニアプリにおいてのみ発生します。フルアプリからはイベントリスナや起床イベントとして登録することはできません。

● **STAR_AUDIO_OUTPUT_PATH_CHANGED**

携帯電話の音声出力先が変更されたことを示すイベントタイプです。このイベントは以下の場合に発生します。

- 携帯電話に外部音声出力機器(有線イヤホン、Bluetooth イヤホンなど)が 1 つも取り付けられていない状態で、外部音声出力機器が新たに取り付けられた場合
- 外部音声出力機器との接続が切断された結果、携帯電話に外部音声出力機器が 1 つも取り付けられていない状態になった場合

このイベントタイプで使用する **AudioOutputEvent** クラスには音声出力先を取得する **getAudioOutputPath()** メソッドが定義されています。

【Star-1.5】

イベントタイプ **STAR_AUDIO_OUTPUT_PATH_CHANGED** は Star-1.5 プロファイルで追加されました。

Star アプリケーションイベントは、**StarApplication.addWakeUpEvent()** メソッドを使用して、起床イベントとして登録することができます。i アプリは **StarApplication.suspend()** メソッドを使用して自らをサスペンドさせることができますが、**suspend()** メソッドによりサスペンドしている状態で起床イベント(**addWakeUpEvent()** メソッドで登録した Star アプリケーションイベント)が発生すると、i アプリはレジュームして Active 状態に復帰します。また **suspend()** メソッドによってサスペンドしている i アプリは、ユーザーのキー操作（厳密にはキープレス操作）によってもレジュームして Active 状態に復帰します。

なお、i アプリオプションのカテゴリに属する機能の Star アプリケーションイベントの中には、対象の i アプリが起動されていない状態（Terminated 状態）で発生したことを、後からその i アプリを起動した際に検知できるものや、i アプリが一時停止している状態（Suspended 状態）で発生したことを i アプリがレジュームした際に検知できるものがあります。これらの i アプリオプションのカテゴリに属する機能の Star アプリケーションイベントについては、「i アプリコンテンツ開発ガイド i アプリオプション・i アプリ拡張編」を参照してください。

注意事項：

- Star アプリケーションイベントと他のイベントが同時に発生するような状況において、イベントの処理順序は保証されません。Star アプリケーションイベントと前項で解説したライフサイクルイベントなどが同時に発生するようなケース（例えばサスペンド中の i アプリが、起床イベントとして登録された Star アプリケーションイベントの発生を受けてレジュームする場合など）で、アプリケーション開発者は特定の実行順に依存した処理を記述するべきではありません。

3.8 ハードウェアのアクセス

i アプリ実行環境では、携帯電話のハードウェア（プラットフォームリソース）にアクセスする手段として **PhoneSystem** クラスを提供しています。このクラスを使用することで、ディスプレイのバックライトやバイブレータの制御などを行うことができます。i アプリからプラットフォームリソースに頻繁にアクセスすると、携帯電話のバッテリーを大量に消費する場合は

ありますので注意してください。i アプリでこれらの設定を変更した場合でも、i アプリ終了時には自動的に元の設定に戻されます。

3.9 エラー処理

i アプリ実行環境の提供する API では、処理中に異常を検出した場合にエラーや例外をスローします。アプリケーションプログラムでは発生した例外をキャッチして、必要に応じて適切な処理（例外メッセージをユーザーに提示するなど）を行うようにしてください。

なお、実行時例外（`java.lang.RuntimeException` およびサブクラス）やエラー（`java.lang.Error` およびサブクラス）がアプリケーションプログラムでキャッチされなかった場合、その i アプリは異常終了します。

3.10 ミニアプリにおける制限

ミニアプリは `WidgetView` 上で最大 9 つまで同時に動作することが可能です。またミニアプリを `WidgetView` に貼り付け設定することで、ユーザーは `WidgetView` を開けばすぐにそのミニアプリの画面を見ることができるようになるなど、使いやすいサービスをユーザーに提供することができます。しかし、ミニアプリはフルアプリに比べ、以下の点において制約があります。

- アプリケーションサイズ（JAR ファイルおよび ScratchPad）
- 画面サイズ
- 利用可能な機能や API
- 利用可能な CPU リソースやメモリ量

ミニアプリの詳細は第 9 章を、機能制限の詳細は付録 A を参照してください。

なお、ミニアプリとフルアプリをパッケージングにより同梱することで、普段はミニアプリとして実行し、ミニアプリでは利用できない機能を利用する際に

`StarApplication.changeAppType()` メソッドを利用しフルアプリに切替えてその機能を利用するといったことが可能になります。

3.11 携帯電話の機種判別

i アプリが現在どのような機種で実行されているかを知るための手段として、以下の 2 つが用意されています。

- ・ i アプリ実行環境におけるシステムプロパティ `microedition.platform`

アプリケーションプログラムは、`java.lang.System.getProperty()` メソッドを使用してシステムプロパティ（システムの情報を取得するためのプロパティセット）を参照することができます。i アプリ実行環境では、このプロパティの値に携帯電話の機種名が格納されます。i アプリは、この値を調べることで自身が動作している携帯電話の機種を知ることができます。

このプロパティに格納される機種名は、ユーザーエージェントに含まれている名前と同等のものとります。

- ・ HTTP 通信時にサーバーに送出される `User-Agent` リクエストヘッダ

ブラウザからの Web アクセスと同様、i アプリから HTTP 通信を実行した際にはリクエストヘッダにユーザーエージェント（`User-Agent` ヘッダ）が含まれます。i アプリと通信を行うサーバーアプリケーションは、`User-Agent` ヘッダの内容を調べることで現在対話を行っている携帯電話の機種を知ることができます。

3.12 FOMA 携帯電話における複数アプリケーション同時起動

FOMA 携帯電話の中には、i モードのサイト閲覧と通話を同時に行うなどのために、複数の端末アプリケーション（例えばブラウザと通話機能など）を同時に起動することができるものがあります。ここでは、そのような携帯電話上で i アプリのライフサイクルが受ける影響について解説します。

(1) i アプリ実行中の、ユーザー操作による他アプリケーションへの切り替え

i アプリ実行中にユーザー操作（タスクキー操作等）により他のアプリケーションへの切り替えが指示された場合、原則としてその i アプリの実行は中断します。ユーザー操作により再度その i アプリへの切り替えが行われた時点で i アプリの実行は再開されます。なお、どのようなアプリケーションが i アプリと同時に起動した状態にできるかはメーカーにより異なります。i アプリ起動中に、ユーザーがその機種で i アプリと同時に起動できないアプリケーション（例えばブラウザなど）の起動を指示すると、i アプリが終了してそのアプリケーションが起動されます。

アプリケーションプログラムは `StarApplicationManager.getSuspendInfo()` メソッドを使用して i アプリ実行の中断理由と中断中に発生した事象を取得することができます。本項で説明したアプリケーション切り替えによる中断が発生した後でこのメソッドを呼び出すと、`StarApplicationManager.SUSPEND_MULTITASK_APPLICATION` を含む値が返ります。

(2) i アプリ実行中のメール着信動作

i アプリ実行中にメールの着信を受けると i アプリを中断することなくバックグラウンドでメールの取得を行いません。バックグラウンドでメールの取得が行われている状態で、パケット通信と同時に実行することのできない機能（赤外線リモコン機能など）を i アプリが使用すると例外が発生します。

第 4 章

ユーザーインタフェースの設計

ユーザーインタフェース（本書では UI と記すことがあります）機能には、次のような基本的な要件があります。

- ボタンやラベルといった高レベルなユーザーインタフェース部品（コンポーネント）の作成
- イベント処理
- レイアウト処理
- 画像ファイルの表示、サウンドファイルの再生機能

i アプリ実行環境のユーザーインタフェース API には、高レベル API と低レベル API の 2 種類があります。次の各節では、個々の基本的な機能分野について説明し、そのコード例を示します。

高レベル API では、さまざまなコンポーネントが提供されます。開発者は、これらを組み合わせて i アプリのユーザーインタフェースを作成します。高レベル API を使用した i アプリはハードウェア特性を考慮する必要性が低く、メーカーの異なる携帯電話でも容易に動作させることができます。高レベル API では開発者の自由度は限定されますが、少ないコード量で高機能な i アプリを作成することができます。高レベル API のユーザーインタフェースを構成する部品をコンポーネントと呼びます。開発者は、これらのコンポーネントを組み合わせて i アプリを作成します。高レベル API を使用した画面は、おおむね i モード HTML と同程度の表現を行うことができます。

低レベル API では、携帯電話の画面サイズやキーパッドなどハードウェアの特性を考慮しながら i アプリを作成します。低レベル API を使用すれば、携帯電話の各機種の特徴を引き出した i アプリを作成することができます。この場合、開発者の自由度は増しますが、グラフィックスの描画や画面サイズと論理座標値といった細かい点を考慮する必要があります。

1 枚の画面は、**Panel** または **Canvas** のインスタンスとして表現されます。**Panel** は高レベル API を使用する画面（コンポーネントベースで構築される画面）で、**Canvas** は低レベル API を使用する画面（描画ベースで構築される画面）で利用されます。ディスプレイには、**Panel** または **Canvas** のいずれかを 1 面だけ表示することが可能です。**Canvas** によって構成される画面でコンポーネントを使用することはできません。ただし **Panel** によって構成される画面には、イメージを取り扱うためのコンポーネントを使用することで低レベル描画を取り入れることができます。

1つのiアプリは複数のPanelやCanvasを持つことができます。アプリケーションプログラムはDisplay.setCurrent()メソッドを使用して、ディスプレイに表示するPanelやCanvasを切り替えることができます。

ボタンを押したり、タイマーが切れるといったユーザーインタフェースイベントの処理は、高レベルAPIと低レベルAPIでは異なります。低レベルAPIでは、キーの押下や解放を含め、すべてのイベントをiアプリで受け取ることができます。一方高レベルAPIでは、コンポーネントによっては、いくつかの低レベルのイベントを内部的に処理したり、より高レベルのイベントに変換してiアプリに通知したりします。

低レベルAPIの場合は、キーの押下や解放などによって発生した低レベルイベントはCanvasオブジェクトに通知されます。低レベルAPIでは、開発者はCanvasクラスを継承したサブクラスを作成して使用しますが、このときCanvasクラスのprocessEvent()メソッドをオーバーライドすることで低レベルイベントを受け取ることができるようになります。イベントが発生するとこのメソッドが呼び出されるので、その中でイベントに対する処理を行います。processEvent()メソッドには、発生したイベントのタイプと、イベントに特有の引数が渡されます。

高レベルAPIの場合は、ボタン押下などの高レベルイベントはPanelオブジェクトに通知されます。iアプリの中でこのような高レベルイベントに対する処理を行う場合は、Panelオブジェクトにリスナーオブジェクトを登録する必要があります。Panelオブジェクトは、イベントが発生するとPanelオブジェクトに登録されているリスナーオブジェクトメソッドを呼び出し、イベントの発生を通知します。高レベルイベントのリスナーインタフェースは、イベント発生源ごとに異なるものが用意されています。イベント発生時にリスナーメソッドに通知される情報も、リスナーの種類により異なります。

イベントリスナーの詳細は、4.4項を参照してください。

次に、ユーザーインタフェースのためのクラス構成を示します。なお、このクラス構成図では第13章にて別途解説する3Dグラフィックス描画機能および3Dサウンド制御機能に関するクラス、インタフェースは省略しています。

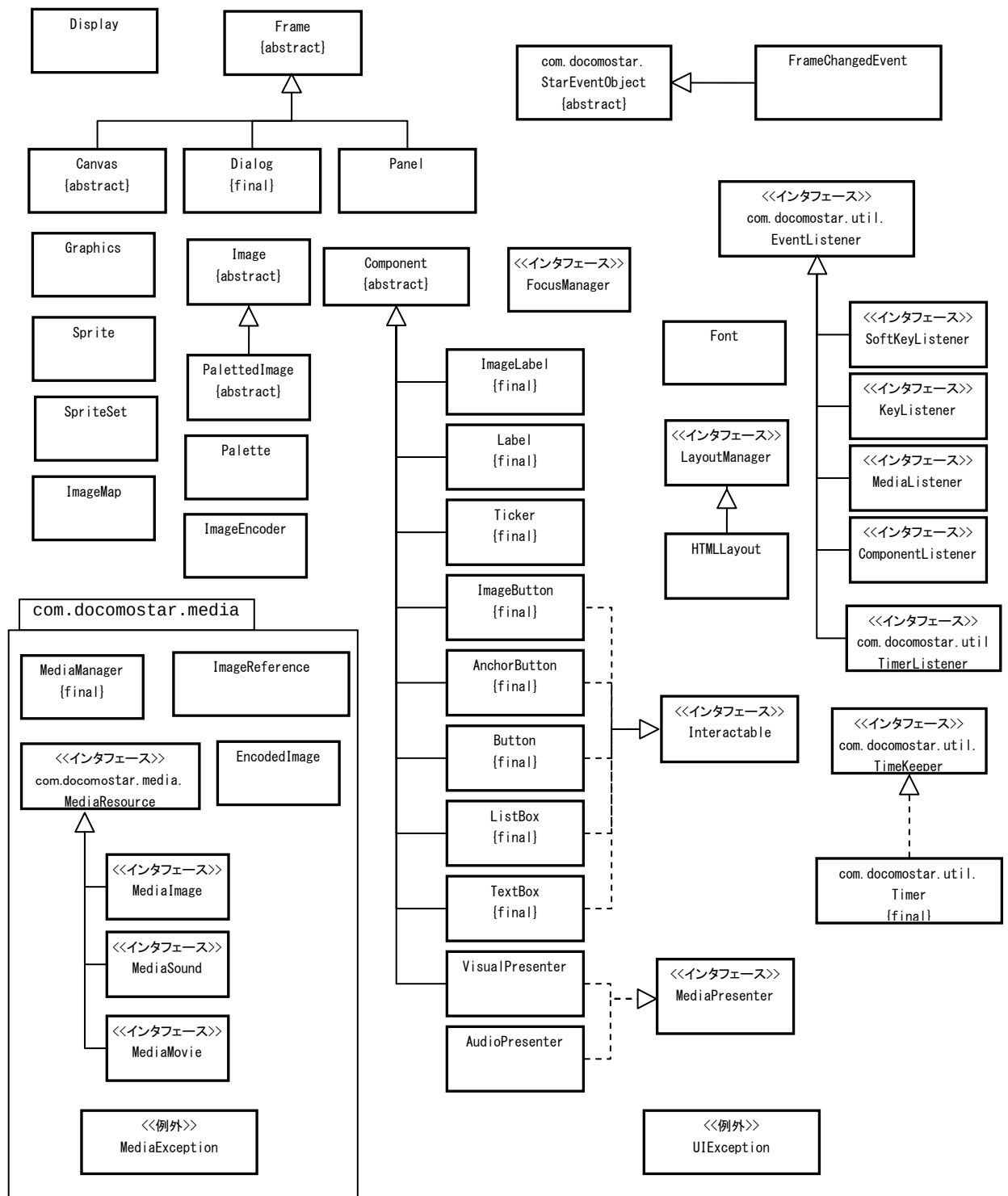


図 8 UI クラス階層

4.1 高レベル UI コンポーネントによるプログラミング

i アプリ実行環境では、9 つの基本的な UI コンポーネントが提供されます。

- Label
- ImageLabel
- Button
- AnchorButton
- ImageButton
- ListBox
- TextBox
- Ticker
- VisualPresenter

この他の基本 UI 部品として **Dialog** があります。上の各要素の動作は携帯電話の各メーカー間でほぼ同様ですが、**Look&Feel** は異なる可能性があります。これは、Windows、Macintosh、UNIX といったコンピュータプラットフォーム向けに開発された標準的な GUI アプリケーションの場合と同じです。これらのアプリケーションはどのプラットフォームでも同じように動作しますが、**Look&Feel** は異なっています。たとえば、ボタンの形状は、プラットフォームによって丸みがかっていたり四角だったり、枠が太かったり細かったりとそれぞれ異なります。これは携帯電話の場合も同様です。低レベルのグラフィックルーチンは、メーカーによって異なる外観を生成することがあります。デフォルトのレイアウト、スクロール、フォーカスの管理に関する方針は、どのメーカーでもほぼ同様です。ただしメーカーは個別に、自社の携帯電話向けにレイアウト、スクロール、フォーカスの管理を拡張している場合があります。次の各項の要素やレイアウトマネージャの説明は一般的なものであり、携帯電話各機種の詳細な振舞いは各機種毎に異なる場合があります。

なお、i アプリ実行環境には UI コンポーネントの共通的な特性を定義する **Component** クラスがありますが、このクラスはメーカーが上記の基本 UI コンポーネントを提供するために用意されているものです。開発者は **Component** クラスを継承した独自のコンポーネントクラスを定義、使用することはできません。

4.1.1 Panel の使用

Panel クラスは、UI 部品のシンプルなコンテナとして機能します。画面には、同時に 1 つのパネルしか表示できません。ただし、これは、1 つの i アプリが 1 つのパネルしか定義できないことを意味するわけではありません。多くの i アプリは複数のパネルから構成されています。ユーザーは、通常、これらのパネル間をナビゲーションしながら i アプリを操作します。

Panel には、その機能や目的を表示するためのタイトルを設定することができます。

4.1.2 Dialog の使用

ダイアログは、テキストメッセージを表示して警告や情報をユーザーに示したり、ユーザーに決定を促がすときに使用します。**Dialog** は **Panel** または **Canvas** の上に重なるように表示される画面であり、**Frame** クラスのサブクラスです。ダイアログには、タイトルとテキストメッセージ、それに最低 1 つのボタンがあります。ダイアログは同時に 1 つしか表示できません。

ダイアログには次のような種類があります。

- 情報
- 警告
- エラー
- Yes/No
- Yes/No/Cancel

注意事項：

- 同時に2つ以上のダイアログを表示しようとすると、例外がスローされます。
- ダイアログはモーダルです。
- バックグラウンドのスレッドがダイアログのモーダル操作に干渉しないようにアプリケーションプログラム側では注意が必要です。例えばダイアログの表示中にバックグラウンドスレッドが背後の **Panel** や **Canvas** を切り替えようとした際の動作は、機種により異なりますので注意してください。
- **Dialog** は、**Panel** や **Canvas** と同様に **Frame** クラスのサブクラスですが、**Dialog** を表示するために **Display.setCurrent()** メソッド（画面に表示されているフレームを切り替えるメソッド）を使用することはできません。**Dialog** を画面に表示するには、**Dialog.show()** メソッドを使用します。

4.1.3 コンポーネントの使用

UI コンポーネント (**com.docomostar.ui.Component** のサブクラス) には、**Label**、**ImageLabel**、**Button**、**AnchorButton**、**ImageButton**、**TextBox**、**ListBox**、**Ticker**、**VisualPresenter** があります。**AnchorButton** を除き、コンポーネントのサイズが画面サイズ (**Panel** にタイトルが設定されている場合はタイトル領域を除いた画面サイズ) を超えることはできません。

コンポーネントのオブジェクトは、1 つにつき 1 回だけ **Panel** に追加 (**Panel.add()** メソッド) することができます。

次の各項では、各 UI コンポーネントについて詳しく説明します。

Label

Label クラスは **Panel** 上に 1 行のテキストを表示するためのコンポーネントです。**Label** コンポーネントでは、枠のない文字列が表示されます。**Label** コンポーネント内のテキストの表示位置は、左、中央、または右に位置合わせすることが可能です。次のコードは、**Label** コンポーネントの例です。

例: **Label** コンポーネント

```
/**
 * ラベルウィジェットとそれに対して呼び出された setSize メソッドの
 * 機能を示すクラス。
 * このクラスは 1 つのボタンと 2 つのラベルウィジェットを作成する。
 * ラベルウィジェットの可視/不可視の機能を示すため、
 * ボタンウィジェットを使用する。
 */
public class LabelDemo extends Panel {
    Label lbl1, lbl2;
    Button btn1;
    int count = 0;

    LabelDemo() {
        lbl1 = new Label("Label1");
```

```

        lbl2 = new Label("This is a Label2");
        btn1= new Button("Button1");
        lbl1.setSize(25,15);    // ラベルのサイズを設定する。

        this.add(btn1);
        this.add(lbl1);
        this.add(lbl2);
        ListenerClass lclass = new ListenerClass();
        this.setComponentListener(lclass);
    }
}

```

注意事項：

- Label コンポーネントがフォーカスを得ることはありません。

ImageLabel

ImageLabel クラスは **Panel** に画像を表示するためのコンポーネントです。**ImageLabel** コンポーネントでは、静的なイメージ（静止画像）のみ表示可能です。イメージが設定されていない **ImageLabel** は、そのコンポーネントの背景色で塗りつぶされます。次のコードは、**ImageLabel** コンポーネントの例です。

例: **ImageLabel** コンポーネント

```

public class ImageLabelDemo extends Panel {
    ImageLabelDemo() {
        ImageLabel l1;

        // メディアイメージを取得する。
        MediaImage mi =
            MediaManager.getImage("resource:///UIDemo/img/cup.gif");
        try {
            mi.use();
        } catch (Exception ce) {
            // リソース取得時に発生した例外の例外処理を記述
        }
        Image im = mi.getImage();

        // イメージオブジェクトをもつイメージラベルを作成する。
        l1 = new ImageLabel(im);
        // イメージラベルをパネルに追加する。
        add(l1);
    }
    ...
}

```

注意事項：

- **ImageLabel** コンポーネントがフォーカスを得ることはありません。
- **Image** から取り出した **Graphics** オブジェクトへの描画機能を用いて、すでに画面に表示されている **ImageLabel** のイメージに描画を行った場合、メーカーによってはその内容が即時には表示に反映されない場合があります。このような場合、**ImageLabel.setImage()** メソッドによりイメージを再設定するなどによりコンポーネント全体を再描画することで描画内容が画面に反映されます。

Button

Button クラスはボタンを表現するコンポーネントで、ユーザーの操作を受け付けるためのインタフェース **Interactable** を実装しています。**Button** コンポーネントには、テキストを設定することができます。このコンポーネントは有効または無効な状態に設定できます。次のコードは、**Button** コンポーネントの例です。

例: **Button** コンポーネント

```
public class ButtonDemo extends Panel implements ComponentListener {
    Button b1, b2, b3, b4;
    ButtonDemo() {
        // 4つのボタンを作成し、それをパネルに追加する。
        b1 = new Button("B4 enabled");
        b2 = new Button("B4 disabled");
        b3 = new Button("ChangeB4");
        b4 = new Button("B4");

        add(b1);
        add(b2);
        add(b3);
        add(b4);
        ...
    }
    ...
}
```

AnchorButton

AnchorButton クラスは HTML のアンカー風ボタン（下線付きテキストとして表示されるボタン）を表現するコンポーネントで、ユーザーの操作を受け付けるためのインタフェース **Interactable** を実装しています。**AnchorButton** コンポーネントには、テキストおよびその見出しとなるイメージを設定することができます。コンポーネントの幅より長いテキストが指定された場合、テキストは自動的に折り返され複数行に渡って表示されます。

AnchorButton の縦方向のサイズは、画面の縦方向のサイズに制限されることはありません。ただし横方向のサイズは画面の横方向のサイズに制限されます。

このコンポーネントは有効または無効な状態に設定できます。無効化された **AnchorButton** のテキストは下線付きテキストではなく通常のテキストのような **Look&Feel** で表示されます。このため無効化された **AnchorButton** は、**Panel** においてユーザーに長い文章などを提示するためのコンポーネントとしても使用することができます。

次のコードは、**AnchorButton** コンポーネントの例です。

例: **AnchorButton** コンポーネント

```
public class AnchorButtonDemo extends Panel implements ComponentListener {
    AnchorButton anchorButton1, anchorButton2;
    AnchorButtonDemo() {
        // 操作可能なアンカーボタンを作成する。
        anchorButton1 = new AnchorButton("次画面へジャンプ");
        this.add(anchorButton1);
        // 操作不能なアンカーボタンを作成し、ユーザーに文章を提示するために使用する。
        anchorButton2 = new AnchorButton(
            "操作不能に設定されたアンカーボタンは、ユーザーに" +
            "長いテキストを提示するために使用することもできます。");
        anchorButton2.setEnabled(false);
        add(anchorButton2);
    }
}
```

```

        ...
    }
    ...
}

```

注意事項：

- **Image** から取り出した **Graphics** オブジェクトへの描画機能を用いて、すでに画面に表示されている **AnchorButton** のイメージに描画を行った場合、メーカーによってはその内容が即時には表示に反映されない場合があります。このような場合、**AnchorButton.setImage()** メソッドによりイメージを再設定するなどによりコンポーネント全体を再描画することで描画内容が画面に反映されます。

ImageButton

ImageButton クラスはイメージを貼り付けることのできるボタンコンポーネントで、ユーザーの操作を受け付けるためのインタフェース **Interactable** を実装しています。有効なイメージが設定されていない **ImageButton** は、そのコンポーネントの背景色で塗りつぶされています。

このコンポーネントは有効または無効な状態に設定できます。

次のコードは、**ImageButton** コンポーネントの例です。

例: **ImageButton** コンポーネント

```

public class ImageButtonDemo extends Panel implements ComponentListener {
    MediaImage mImage;
    Image image;
    ImageButton imageButton;
    ImageButtonDemo() {
        // イメージボタンを作成し、それをパネルに追加する。
        try {
            mImage = MediaManager.getImage("resource:///label.gif");
            mImage.use();
            image = mImage.getImage();
        } catch (Exception ce) {
            ...
        }
        imageButton = new ImageButton(image);
        add(imageButton);
        ...
    }
    ...
}

```

注意事項：

- **Image** から取り出した **Graphics** オブジェクトへの描画機能を用いて、すでに画面に表示されている **ImageButton** のイメージに描画を行った場合、メーカーによってはその内容が即時には表示に反映されない場合があります。このような場合、**ImageButton.setImage()** メソッドによりイメージを再設定するなどによりコンポーネント全体を再描画することで描画内容が画面に反映されます。

ListBox

ListBox クラスは各種の選択リストを表現するためのコンポーネントで、ユーザーの操作を受け付けるためのインタフェース **Interactable** を実装しています。**ListBox** コンポーネントは矩形の枠をもっており、その中で項目をスクロール表示することが可能です。**ListBox** に表

示する項目全体が画面に入りきらないと、上下のスクロールを示すインジケータが表示されます。**ListBox** には次のような種類があります。

- 単一選択リスト
- 複数選択リスト
- ラジオボタン
- チェックボックス
- 番号付きリスト
- オプションメニュー（ポップアップ）

枠の内部に含まれる項目の表示と選択方法は、使用する **ListBox** のタイプによって異なります。

タイプ	選択	
	単一選択	複数選択
リスト	単一選択リスト (SINGLE_SELECT)	複数選択リスト (MULTIPLE_SELECT)
番号付きリスト	番号付きリスト (NUMBERED_LIST)	なし
チェックボックス	なし	チェックボックスリスト (CHECK_BOX)
ラジオボタン	ラジオボタンリスト (RADIO_BUTTON)	なし
ポップアップ	オプションメニュー (CHOICE)	なし

このコンポーネントは有効または無効な状態に設定できます。次のコードは、**ListBox** コンポーネントの例です。

例: **ListBox** コンポーネント

```
/**
 * リストボックスウィジェットの機能を示すクラス。
 * ここでは、リストタイプとしてラジオボタン、番号付きリスト、チェックボックス、
 * オプションメニュー、単一選択リスト、複数選択リストを示す。
 */
public class ListBoxDemo extends Panel implements SoftKeyListener {
    ListBox l1, l2, l3, l4, l5, l6;
    int count = 0;
    int newItemCnt=0;
    ListBoxDemo() {
        Label radioLabel, checkLabel, numberLabel;
        Label choiceLabel, singleLabel, multipleLabel;

        radioLabel = new Label("Radio Button List");
        // RADIO_BUTTON タイプのリストボックスを作成し、いくつかの項目を追加する。
        l1 = new ListBox(ListBox.RADIO_BUTTON);
        l1.append("radio1");
        l1.append("long radio label");
        l1.append("radio3");
        l1.append("radio4");

        checkLabel = new Label("Checkbox List");
        // CHECK_BOX タイプのリストボックスを作成し、いくつかの項目を追加する。
        l2 = new ListBox(ListBox.CHECK_BOX);
        l2.append("checkbox1");
        l2.append("checkbox2");
```

```

l2.append("checkbox3");
l2.append("checkbox4");
l2.append("checkbox5");

numberLabel = new Label("Number List");
// NUMBERED_LIST タイプのリストボックスを作成し、いくつかの項目を追加する。
l3 = new ListBox(ListBox.NUMBERED_LIST);
l3.append("number1");
l3.append("number2");
l3.append("number3");
l3.append("number4");
l3.append("number5");

singleLabel = new Label("Single List");
// SINGLE_SELECT タイプのリストボックスを作成し、いくつかの項目を追加する。
l4 = new ListBox(ListBox.SINGLE_SELECT);
l4.append("single1");
l4.append("single2");
l4.append("single3");
l4.append("single4");
l4.append("single5");

multipleLabel = new Label("Multiple List");
// MULTIPLE_SELECT タイプのリストボックスを作成し、いくつかの項目を追加する
l5 = new ListBox(ListBox.MULTIPLE_SELECT);
l5.append("multiple1");
l5.append("multiple2");
l5.append("multiple3");
l5.append("multiple4");
l5.append("multiple5");

choiceLabel = new Label("Choice List");
// CHOICE タイプのリストボックスを作成し、いくつかの項目を追加する。
l6 = new ListBox(ListBox.CHOICE);
l6.append("choice1");
l6.append("choice2");
l6.append("choice3");
l6.append("choice4");
l6.append("choice5");

this.add(radioLabel);
this.add(l1);
this.add(checkLabel);
this.add(l2);
this.add(numberLabel);
this.add(l3);
this.add(singleLabel);
this.add(l4);
this.add(multipleLabel);
this.add(l5);
this.add(choiceLabel);
this.add(l6);

// このパネルに softkeylistener を設定する。
this.setSoftLabel(Frame.SOFT_KEY_2, "List");
setSoftKeyListener(this);
}
...
}

```

注意事項：

- 番号付きリストは単一選択リストの特別な形式です。ユーザーは、番号付きリストの先頭から9個目までの項目を、携帯電話の数字キーを使用して選択することができます。最初の項目を選択するには数字1のキー、2つめの項目を選択するには数字2のキーというように使用します。10個目以後の項目は単一選択リストと同じ操作により選択します。

- オプションメニュー（ポップアップ）も単一選択リストの特別な形式です。オプションメニューにフォーカスがある状態で携帯電話の「選択」ボタンを押すと、ポップアップが表示されます。ユーザーはこのポップアップから目的の項目を選択します。なお、他の種類の **ListBox** と異なり、オプションメニューは初期状態では先頭の項目が選択された状態になっています。
- カレントフレームに追加済みの **ListBox** へのアプリケーションプログラムからの項目設定処理（**setItems()**や**append()**など）と、その **ListBox** に対するユーザー操作が同時に行われた場合の振る舞いはメーカーにより異なります。カレントフレーム上にある **ListBox** に、アプリケーションプログラムから項目の設定を行う際は、**setEnabled()**メソッドによりユーザー操作を無効化するようにしてください。

TextBox

TextBox クラスは入力フィールドを表現するコンポーネントで、ユーザーの操作を受け付けるためのインタフェース **Interactable** を実装しています。**TextBox** コンポーネントでは、単一行または複数行のテキストが表示されます。テキスト入力可能なコンポーネントは **TextBox** だけです。アプリケーションプログラムで **TextBox** の編集モードを編集可能に設定すると、ユーザーが IME（Input Method Editor）を起動して、テキストを入力したり変更したりすることが可能になります。編集モードを編集不可に設定すると、**TextBox** は読み取り専用になります。

TextBox のコンストラクタには、表示モードパラメータとして **DISPLAY_PASSWORD** または **DISPLAY_ANY** を指定します。入力テキストを非表示とする必要のある i アプリでは、**TextBox** コンポーネントを **DISPLAY_PASSWORD** モードで使用すると、すべての文字が"*"で表示されます。表示モードパラメータを **DISPLAY_ANY** に設定すると、**TextBox** コンポーネントは入力テキストをそのまま表示します。

初期入力モード（かな漢字入力、アルファベット入力などの入力モード）を変更するには、**setInputMode()**メソッドに初期入力モードパラメータとして **KANA**（かな漢字入力モード）、**ALPHA**（アルファベット入力モード）、または **NUMBER**（数字入力モード）を指定します。表示モードが **DISPLAY_ANY** の **TextBox** では、**setInputMode()**メソッドを呼び出していない状態では初期入力モードは **KANA** になっています。

このコンポーネントは、有効または無効な状態に設定することができます。次のコードは、**TextBox** コンポーネントの例です。

例: **TextBox** コンポーネント

```
public class TextBoxDemo extends Panel implements ComponentListener {
    TextBox t;                // 単一行テキストボックス (テキストフィールド)
    TextBox t2;               // 複数行テキストボックス (テキスト領域)
    Button b;
    TextBoxDemo() {
        t2 = new TextBox(
            "This is a test for textbox",20,3,TextBox.DISPLAY_ANY);
        t = new TextBox(
            "This is a test for textbox",20,1,TextBox.DISPLAY_ANY);
        b = new Button("TestText");

        this.add(b);
        this.add(t2);
        this.add(t);
        ...
    }
    ...
}
```

【Star-1.2】

携帯電話の機種によっては、マルチリンガル文字（日本語、英語以外の言語の文字）に対応している場合があります。**TextBox**でのマルチリンガル文字入力に対応するため、**Star-1.2** プロファイル以降、**TextBox**の初期入力モード定数が拡張されています。

Star-1.2 プロファイルでサポートされる追加の初期入力モードは以下の通りです。

- ・ **TRADITIONAL_HANZI** : 中国語(繁体字)入力モード
- ・ **SIMPLIFIED_HANZI** : 中国語(簡体字)入力モード
- ・ **HANGUL** : 韓国語(ハングル)入力モード
- ・ **PORTUGUESE** : ポルトガル語入力モード

ただし、マルチリンガルに対応するかどうか、および対応する場合にどの言語をサポートするかはメーカーにより異なります。また、端末設定によりマルチリンガル文字への対応を有効としていない場合は、これらの指定を行っても初期入力モードとして有効になりません。

注意事項：

- ユーザーが **TextBox** でテキストの編集を行っている状態（IME が起動している状態）では、バックグラウンドのスレッドが IME のモーダル操作に干渉しないようにアプリケーションプログラム側では注意が必要です。例えば IME が起動されているときにバックグラウンドスレッドが背後の **Panel** を他のフレームに切り替えようとした際の動作は、機種により異なりますので注意してください。
- **TextBox** に設定するテキストの長さ（**setText()**メソッドなどで設定するテキストや IME で入力するテキストの長さ）を制限するための **setInputSize()**メソッドは、メーカーによってはサポートされない場合があります。

Ticker

Ticker クラスはスクロールする文字列を **Panel** に表示させるためのコンポーネントです。**Ticker** コンポーネントでは、テキストがティッカーテープ表示形式で表示されます。文字列は、画面を右から左にスクロールします。

次のコードは、**Ticker** コンポーネントの例です。

例: Ticker コンポーネント

```
// ティッカーウィジェットの機能を示すクラス。
// 最初にパネルが表示されたときには、ティッカーのデフォルト文字列が動いている。
// ユーザーは、テキストボタンを押してテキストを変更できる。
// ティッカーは可視または不可視に設定可能

public class TickerDemo extends Panel implements ComponentListener {
    Ticker ticker;           // ティッカーウィジェット
    Button textButton;       // テキストの変更

    static int    whichText  = 1;
    static String textString1 = "This is a test text 1 for the ticker panel";
    static String textString2 = "Now you see test text 2";

    TickerDemo() {
        ticker      = new Ticker(textString1);
        textButton = new Button("change text");

        this.add(ticker);
        this.add(textButton);
        this.setComponentListener(this);
    }
}
```

```
// Component c でイベントが発生したことを示している。
//
// @param c Component。このイベントが発生したコンポーネント
// @param type イベントタイプ
// @param param イベントのパラメータ
public void componentAction(Component c, int type, int param) {

    // コンポーネントアクションが発生した。textButton がクリックされると、
    // ティッカーの文字列が他のものに変更される。
    if (c == textButton) {
        if (whichText==1) {
            whichText=2; ticker.setText(textString2);
        } else if (whichText==2) {
            whichText=1; ticker.setText(textString1);
        }
    }
}
}
```

注意事項：

- コンポーネント内のテキストのスクロール動作をアプリケーションプログラムから制御することはできません。
- **Ticker** コンポーネントがフォーカスを得ることはありません。
- テキストのスクロール速度はメーカーにより異なります。

VisualPresenter

VisualPresenter クラスは動画像を再生するためのコンポーネントで、**MediaPresenter** を実装しています。**VisualPresenter** コンポーネントは、画面に表示されるメディアデータ（GIF アニメーションなど）の再生を行う高レベル API のコンポーネントです。このコンポーネントでは、メディアデータの再生に関するイベント処理のために **MediaListener** を使用することができます。イベント処理の詳細は、4.4 項を参照してください。メディアデータを再生するコード例は、4.5 項で説明します。

注意事項：

- **VisualPresenter** はコンポーネントであり、高レベル API でのみ使用することができます。
- **VisualPresenter** コンポーネントがフォーカスを得ることはありません。

4.1.4 コンポーネントの配置

開発者は、コンポーネントの配置をレイアウトマネージャに任せるか、レイアウトマネージャを無効にして **Component** クラスの **setLocation()** メソッドでコンポーネントごとに配置位置を指定するかを選択することができます。

レイアウトマネージャは **Panel** に設定して使用します。**Panel** にレイアウトマネージャを設定するには、**Panel.setLayoutManager()** メソッドを使用します。

i アプリ実行環境で用意されているレイアウトマネージャには、**Panel** を生成した初期状態で設定されているデフォルトレイアウトマネージャと、HTML 風の指定方法によりレイアウトを行うための **HTML** レイアウトマネージャの 2 つがあります。

デフォルトレイアウトマネージャの基本レイアウト方針は、シンプルなフローレイアウトです。パネルに追加されたコンポーネントは画面の左から右に向けて順次横並びに配置されていき、画面の横幅で収まりきらなくなった時点で次のコンポーネント配置が可能な位置（1 段下の列）へと配置されます。

また HTML レイアウトマネージャでは、コンポーネントを **Panel** に追加していく過程で改行（BR タグに相当）、段落（P タグに相当）、アライメント（CENTER や DIV タグなどに相当）を指定することで、開発者がより明示的にレイアウトを指示することができます。

どのメーカーの携帯電話にも適用できる汎用的な i アプリを作成するためには、コンポーネントのレイアウト構成をシンプルに保つことを推奨します。なお、配置可能なコンポーネントの数は、メモリ使用状況や各機種固有の制限などの影響を受けます。

注意事項：

- **LayoutManager** インタフェースはメーカーがレイアウトマネージャを実装するために用意されているものであり、開発者が本インタフェースを利用して独自のレイアウトマネージャを作成することはできません。**Panel** オブジェクトの **setLayoutManager()** メソッドを呼び出すことで、その **Panel** が使用するレイアウトマネージャを変更することができます。
- レイアウトマネージャを無効にするには、**Panel** オブジェクトに対して、パラメータに **null** を指定して **setLayoutManager()** メソッドを呼び出します。レイアウトマネージャを無効にした場合、アプリケーションプログラムは各コンポーネントの **setLocation()** メソッドを使用して、コンポーネント毎にそれを配置する位置を指定します。なお、レイアウトマネージャを無効にした場合、機種によってはスクロール機能に制限が発生します。
- 1 つの HTML レイアウトマネージャオブジェクトは、2 つ以上の **Panel** に同時に設定することはできません。1 つの i アプリの中に HTML レイアウトを適用したい **Panel** が複数ある場合には、それぞれの **Panel** に対し別々の HTML レイアウトマネージャオブジェクトを割り当てるようにしてください。
- HTML レイアウトマネージャは、**setLayoutManager()** メソッドで **Panel** に設定される度に初期状態（アライメントが左寄せで改行、段落指定のない状態）に戻ります。

次のコードは、HTML レイアウトマネージャを使用してコンポーネントの配置を行う例を示したものです。

例: HTML レイアウトマネージャによるコンポーネントの配置

```
public class HTMLLayoutDemo extends Panel {
    Label      label1, label2, label3, label4;
    HTMLLayout  lm;

    HTMLLayoutDemo() {
        label1 = new Label("Label 1");
        label2 = new Label("Label 2");
        label3 = new Label("CENTER");
        label4 = new Label("RIGHT");

        // HTML レイアウトマネージャを生成し、Panel に設定する。
        lm = new HTMLLayout();
        setLayoutManager(lm);

        // レイアウトを指定しつつ Panel に Label を追加する。
        add(label1);
        //以降のコンポーネントは次行に配置する（BR タグ相当）。
        lm.br();

        add(label2);
        // 以降のコンポーネントは次行に配置する（BR タグ相当）。
        lm.br();
    }
}
```

```

// begin()から対応する end()までの間に追加されたコンポーネントを
// 中央寄せで表示する。
lm.begin(HTMLLayout.CENTER);
add(label3);
lm.end();

// begin()から対応する end()までの間に追加されたコンポーネントを
// 右寄せで表示する。
lm.begin(HTMLLayout.RIGHT);
add(label4);
lm.end();
    }
}

```

また次のコードは、レイアウトマネージャを無効化して、アプリケーションプログラムが独自にコンポーネントの配置を行う例を示したものです。

例: setLocation()によるコンポーネントの配置

```

public class NullLayoutDemo extends Panel {
    Button button1, button2;
    Button offScreen;
    TextBox text;
    Label title, button1Label, button2Label;

    NullLayoutDemo() {
        // ここですべてのコンポーネントを使用する。
        title = new Label("Null Layout ");
        button1 = new Button("Button1");
        button2 = new Button("Button2");
        button1Label = new Label("ONE:");
        button2Label = new Label("TWO:");
        text = new TextBox("A simple sentence in a textbox", 40,
                           2, TextBox.DISPLAY_ANY);

        /*
        * 画面外に配置するコンポーネントの例としてこのボタンを作成する。
        */
        offScreen = new Button("This button is never seen");

        /*
        * ここでレイアウトマネージャに null を設定する。したがって、個々のコンポー
        * ネットの場所を設定する必要がある。注: 物理的な画面の座標外に配置される
        * コンポーネントは全く表示されない。null レイアウトにより、スクロールは無効に
        * される場合がある。
        */
        setLayoutManager(null);

        /* 注: この場合でも、コンポーネントを追加する順序には意味がある。コンポーネン
        トは、
        * この順序に従ってフォーカスを与えられる。
        */
        add(title);
        add(button1Label);
        add(button1);
        add(button2Label);
        add(button2);
        add(text);
        // このコンポーネントも追加する。
        add(offScreen);

        /*

```

ウト

を

スを

}

- * ここで、各コンポーネントの場所を設定する必要がある。つまり、パネルのレイアウトを独自に指定する。コンポーネントのサイズは自動的に設定することも、setSizeを直接呼び出してサイズを指定することもできる。

```
title.setLocation(12, 0);
button1Label.setLocation(10, 18);
button1.setLocation(45, 18);
button2Label.setLocation(10, 40);
button2.setLocation(45, 40);
text.setLocation(2, 60);
```

```
/*
```

- * 注：このボタンは画面外に配置される。したがって、表示もされないし、フォーカスを

- * 得ることもない。

```
*/
```

```
offScreen.setLocation(1000, 1000);
```

```
}
```

4.1.5 フォーカスとスクロール

フォーカス管理とスクロール管理の詳細な振る舞いは、メーカーにより異なる場合があります。i アプリ対応携帯電話にはシンプルなデフォルトフォーカスマネージャが搭載されています。典型的なデフォルトフォーカスマネージャでは、コンポーネントが **Panel** に追加された順序がフォーカスを取得する順序になります。i アプリが起動されて画面が表示されると、有効な状態に設定されている最初の対話可能なコンポーネントにフォーカスが設定されます。ただし、フォーカスが設定されているコンポーネントがディスプレイの領域外に配置されている状態では、そのコンポーネントを操作することはできません。

デフォルトのスクロール動作は、フォーカス管理と緊密な関係があります。これは、画面のスクロールがフォーカスの移動に伴って行われるためです。

注意事項：

- **FocusManager** インタフェースはメーカーがフォーカスマネージャを実装するために用意されているものであり、開発者が本インタフェースを利用して独自のフォーカスマネージャを作成することはできません。**Panel** オブジェクトの **setFocusManager()** メソッドを呼び出すことで、その **Panel** が使用するフォーカスマネージャを変更することができます。ただし現在までのプロファイルでは、デフォルトフォーカスマネージャ以外のフォーカスマネージャは規定されていません。
- フォーカスマネージャを無効化することはできません。

4.1.6 高レベルイベントの処理

Panel により構成された画面では、コンポーネントによって生成されたイベントは、**ComponentListener** を実装したオブジェクトの **componentAction()** メソッドに渡されます。またファンクションキーイベントは、**SoftKeyListener** を実装したオブジェクトの **softKeyPressed()** または **softKeyReleased()** メソッドに渡されます。このように、高レベルイベント処理にはリスナーモデルが採用されています。

ComponentListener も **SoftKeyListener** も **Panel** に登録して使用します。高レベルイベント処理におけるイベントリスナーとコード例の詳細は、4.4 項を参照してください。

4.1.7 コンポーネントのフォントサポート

Component オブジェクトや **Dialog** オブジェクトに対して **setFont()** メソッドを呼び出すことで、これらに表示される文字のフォントをアプリケーションプログラムから指定することができます。使用したいフォントを表す **Font** オブジェクトを生成して **setFont()** メソッドの引数に指定することで、デフォルトフォント以外のフォントを表示することができます。

なおコンポーネントにフォントを設定する場合、このメソッドは 1 つのコンポーネント上に表示される文字列全体に作用します。同一コンポーネント上に異なる複数のフォントで構成される文字列を表示することはできません。また、文字を表示する機能を持たないコンポーネントに対しては、このメソッドを呼び出して何も行われません。

また、ダイアログにフォントを設定する場合、このメソッドはダイアログの本文テキスト全体にのみ作用します。ダイアログタイトルやダイアログ上のボタンのフォントを変更することはできません。

注意事項：

- パネルタイトルやファンクションキーラベルの表示文字のフォントを変更することはできません。

4.2 低レベル API によるプログラミング

ゲームアプリケーションなどの分野では、コンポーネントを使用したユーザーインタフェースよりもグラフィックス描画機能を多用する場合があります。この節では、主にグラフィックス描画を行うための低レベル API の使い方とイベントの処理方法について説明します。

4.2.1 Canvas の使用

Canvas は、グラフィックスを描画するための画面領域です。開発者は **Canvas** クラスのサブクラスを作成し、**paint()** メソッドをオーバーライドすることによって、**Canvas** に描画することができます。**paint()** メソッドは、システムが **Canvas** の描画が必要と判断したときやアプリケーションプログラムが明示的に再描画を要求したとき (**repaint()** メソッドを呼び出したとき) にシステムからコールバックされます。

また、**Canvas** に対して、初めて **Display.setCurrent(Frame)** メソッドが実行された場合、**init(Graphics)** メソッド、及び **paint(Graphics)** メソッドがコールバックされます。**Canvas** オブジェクトに対する 2 回目以降の **setCurrent()** メソッド呼び出しでは、**init(Graphics)** メソッドはコールバックされず、**paint(Graphics)** メソッドのみコールバックされます。

Panel と同様に、1 つの **Canvas** オブジェクトは 1 つの画面に相当します。したがって、1 つの画面に複数の **Canvas** を表示したり、**Canvas** と **Panel** を同時に 1 つの画面に表示することはできません。ただし複数の **Canvas** や **Panel** を、1 つの画面上に順次切り替えて表示することは可能です。

次に **Canvas** に簡単な描画を行う例を示します。

例: **Canvas** への描画

```
public class CanvasDemo extends Canvas {
    int x = 20;
    int y = 10;
```

```

CanvasDemo() {
    // ファンクションキーラベル「back」を設定する。
    setSoftLabel(Frame.SOFT_KEY_1, "Back");
    // 決定キーラベル「決定」を設定する。
    setSoftLabel(Frame.SELECT_KEY, "決定");

}
/**
 * キャンバスの初期化处理
 */
public void init(Graphics g) {
    g.setColor(Graphics.getColorOfRGB(0xFF, 0x00, 0x00));
    g.fillRect(0, 0, getWidth(), getHeight());
    g.setColor(Graphics.getColorOfRGB(0x00, 0xFF, 0x00));
}
/**
 * キャンバス領域に文字列と矩形を描画する。
 */
public void paint(Graphics g) {
    g.lock();
    g.clearRect(0, 0, 1000, 1000);
    g.drawString("Hello", x, y);
    g.drawString("Canvas", x, y+15);
    g.drawRect(x+10, y+30, 20, 5);
    g.unlock(true);
}
...
}

```

注意事項：

- Canvas オブジェクトに対する初回の **setCurrent()** メソッド呼び出し時、**init()** メソッドと **paint()** メソッドが同時並行的に実行されることはありません。最初に **init** メソッドがコールバックされ、**init** メソッドの終了後に **paint(Graphics)** メソッドがコールバックされます。
- **repaint()** メソッドはシステムに対し描画の必要があることを指示するためのものであり、1 回の **repaint()** メソッド呼び出しが 1 回の **paint()** メソッド呼び出しに対応することを保証するものではありません。このためアプリケーションプログラムから見た場合に、複数の連続した **repaint()** メソッド呼び出しが 1 回の **paint()** メソッド呼び出しにまとめられることがあります。開発者は、**paint()** メソッド呼び出しが 1 回にまとめられたような場合にも画面の描画が正しく行われるよう **paint()** メソッドを記述する必要があります。
- カレントフレームに設定されていない Canvas に対する **repaint()** メソッドの呼び出しは無視されます。
- メーカーによっては ADF の **DrawArea** キーを使用することで、デフォルト (**DrawArea** キー未使用時) の表示領域よりも広い表示領域を使用できる場合があります。そのような機種には、**Canvas.setSoftLabelVisible()** メソッドを使用することで、Canvas 表示時のファンクションキーラベルの表示のオン、オフを切り替えることができます。なお、**setSoftLabelVisible()** メソッドは **Frame** クラスで定義されたメソッドですが、現在のプロファイルではフレームオブジェクトのうち、**Panel** や **Dialog** に対してこのメソッドを呼び出すことはできません。

4.2.2 グラフィックスの描画

線や図形、テキストなどの描画には、グラフィックスオブジェクト (**Graphics** クラスのオブジェクト) を使用します。このオブジェクトには描画可能な領域、フォント、色などの情報が含まれているため、グラフィックスコンテキストとも呼ばれます。前項の例ではグラフィックスコンテキスト **g** を使い、**clearRect()** メソッドによる画面のクリアの後に **drawString()** メソッドを呼び出してテキストを描画し、続いて **drawRect()** メソッドを呼び出して矩形を描画しています。

上の例で使用している `lock()` メソッドと `unlock()` メソッドは、グラフィックス描画のダブルバッファリングを管理するために **Graphics** クラスに用意されているメソッドです。ダブルバッファリング中は、指示された描画はオフスクリーンバッファに対して行われ、ダブルバッファリング終了とともにオンスクリーンにコピーされます。このため、ダブルバッファリングを使用するとフリッカーが発生しません。

ダブルバッファリングは、メーカーによりサポートされない場合があります。ダブルバッファリングをサポートしない機種では、この 2 つのメソッドは何も行いません。ダブルバッファリングをサポートするデバイスでは `lock()` メソッドが呼び出されると（1 回または複数回）、`unlock()` メソッドが引数に `true` が指定されて呼び出されるか、引数に `false` が指定されて `lock()` メソッド呼び出しと同じ回数だけ呼び出されるまで、画面への描画は一時的に保留されます。

注意事項：

- ダブルバッファリングの実装は携帯電話によって異なります。アプリケーションプログラムの正しい動作を保証するために、ダブルバッファリングを使用する際は次のガイドラインに従うようにしてください。
 - 異なる **Graphics** オブジェクトの `lock()` メソッドと `unlock()` メソッドの呼び出しを混在して行わない。
 - `paint()` メソッドを終了する前には必ず **Graphics** オブジェクトのロックを解除する。

次のコードに、ロックおよびロック解除の正しい使い方を示します。

例：正しい実装（上記ガイドラインに従っている）

```
public void paint(Graphics g) {
    Graphics g2 = g.copy();
    g.lock();
    // 何かを描画する .....
    g.unlock(false);
    g2.lock();
    // さらに描画する .....
    g2.unlock(false);
}
```

次のコードに、ロックおよびロック解除の正しくない使い方を示します。

例：正しくない実装

```
public void paint(Graphics g) {
    Graphics g2 = g.copy();
    g.lock();
    // 何かを描画する .....
    g2.lock();                      // g と g2 のロック区間が重複している
    // さらに描画する .....
    g.unlock(false);               // g2 のロック解除が行われていない
}
```

- i アプリ実行環境では、`repaint()` メソッドの引数に矩形領域を設定することで指定された矩形領域のみの再描画を行うことができます。
- i アプリ実行環境では透過 GIF がサポートされており、透過 GIF のメディアイメージから生成したイメージオブジェクトを描画すると透過部分は描画対象外となります。ただしこの効果は透過 GIF から生成されたイメージオブジェクトを描画する際のみであり、その描画結果からは透過情報は失われます。
- i アプリの再開時にどの程度システムが画面を復元できるかは機種により異なります。
 - i アプリの再開時にシステムが **Canvas** の描画内容をすべて復元できた場合は、`paint()` メソッドは呼び出されません。

- 低レベル文字列描画に使用されるデフォルトフォントは 24 ドットフォントです。またどの機種でも共通にサポートされているフォントサイズは 12 ドット、16 ドット、24 ドット、30 ドット、32 ドット、48 ドット、60 ドットです。メーカーによってはそれ以外のサイズのフォントをサポートしている場合があります。共通にサポートされているフォントサイズを含め、携帯電話がサポートしているすべてのフォントサイズは `Font.getSupportedFontSizes()` メソッドを使用して調べることができます。

【Star-2.0】

Star-2.0 プロファイルに対応する携帯電話では、どの機種でも共通に 12 ドットから 60 ドットの範囲内で 1 ドット単位のフォントサイズをサポートしています。これらのフォントサイズを含め、携帯電話がサポートしているすべてのフォントサイズは `Font.getSupportedFontSizes()` メソッドを使用して調べることができます。

`Font.getFont()` メソッドを使用して、フォントサイズを数値で直接指定することで、そのサイズに対応する `Font` オブジェクトを取得することができます。

高精細化の進んだ LCD デバイス上では同一ドット数のフォントでも視覚上のサイズは小さくなるため、画面設計においては状況に応じてより大きなフォントを使用することも検討してください。

- `Canvas` に対応付けられているグラフィックスオブジェクトへの描画はその `Canvas` がカレントフレームに設定されている場合のみ行われます。対応付けられている `Canvas` がカレントフレームに設定されていない場合、そのグラフィックスオブジェクトに対する描画メソッドの呼び出しは無視されます。

4.2.3 Image オブジェクトへの描画

アプリケーションプログラムは、`Image` オブジェクトから `Graphics` オブジェクトを取り出すことで、`Image` オブジェクトに対しても低レベル API を用いた描画を行なうことができます。

`Image` オブジェクトからの `Graphics` オブジェクトの取得は、`Image.getGraphics()` メソッドを呼び出すことにより行います。`Image` オブジェクトから取得した `Graphics` オブジェクトへの描画指示は、元となる `Image` オブジェクトの表示内容に反映されます。このようにして描画された `Image` オブジェクトを `ImageLabel` や `ImageButton` に設定することで、アプリケーションプログラムが行った描画処理の結果を `Panel` 使用時にも表示することができます。

`Image` オブジェクトから `Image.getGraphics()` メソッドにより `Graphics` オブジェクトを取得するには、その `Image` オブジェクトが `Image.createImage()` メソッドにより生成された新規の `Image` オブジェクトでなければなりません。`MediaImage.getImage()` メソッドで取得した `Image` オブジェクトに対して `getGraphics()` メソッドを呼び出した場合、`UnsupportedOperationException` がスローされます。

なお、`Image.createImage()` で生成されたイメージは、初期状態ではその機種における `Canvas` のデフォルト背景色と同じ色で塗りつぶされています。ただし `Image` 自身には背景色の概念はありません。`Image` から取得した `Graphics` に対して描画内容のクリアを指示するには、`Graphics.clearRect()` メソッドではなく、適切な色の設定を行った上で `Graphics.fillRect()` メソッドを呼び出します。

次のコードは、アプリケーションプログラムからの指示により描画された `Image` オブジェクトを `ImageLabel` で使用する例です。

例: `Image` オブジェクトへの描画

```
public class ImageDrawDemo extends Panel {
    ImageLabel    iLabel;
    Image         img;
    Graphics      g;
```

```

ImageDrawDemo() {
    // Image オブジェクトを新規に生成し、Graphics オブジェクトを取得する。
    int width = 100;
    int height = 50;
    img = Image.createImage(width, height);
    g = img.getGraphics();

    // 取得した Graphics オブジェクトに描画する。
    g.lock();
    g.drawRect(0, 0, width-1, height-1);
    g.drawString("Graphics による", 5, 16);
    g.drawString("Image への", 5, 32);
    g.drawString("描画のサンプル", 5, 48);
    g.unlock(true);

    // Image オブジェクト ImageLabel に設定し、Panel 上に表示する。
    iLabel = new ImageLabel(img);
    add(iLabel);
}
}

```

注意事項：

- **Image.createImage()** メソッドでは、少なくともその機種におけるネイティブ描画領域の長辺を 1 辺とする正方形のサイズまでの **Image** を生成できることが全機種で保証されます。これより大きなサイズの **Image** を生成可能かどうかはメーカーにより異なりますので注意してください。

4.2.4 カラーサポート

Graphics オブジェクトには、オブジェクトの色情報が含まれています。**Graphics** クラスには次のメソッドが含まれており、それぞれが色を表す整数値を返します。

<code>getColorOfName(int name)</code>	: 色名 (Graphics 定数) に対応する整数値を返します。
<code>getColorOfRGB(int r,int g,int b)</code>	: RGB 値に対応する整数値を返します。
<code>getColorOfRGB(int r,int g,int b,int a)</code>	: RGB 値およびアルファ値に対応する整数値を返します。

アルファ値を含む色指定を使用することで、半透明の描画（描画先のピクセル色と描画元のピクセル色のブレンド）が可能となります。アルファ値 **a** には 0～255 までの値を指定し、255 が完全に不透明であることを意味します。ただし、色のアルファ値要素が有効となる描画メソッドは以下のように限定されています。

- `fillArc()`
- `fillPolygon()`
- `fillRect()`

他の描画メソッドや高レベル UI の色指定メソッドではアルファ値要素の設定内容は無効となり、アルファ値として 255 が指定されたものとして処理されます。

色の管理方法はメーカーに依存するため、i アプリ実行環境内部で色に割り当てられる整数値（これらのメソッドの返り値）はメーカーによって異なる可能性があります。開発者はこの整数値にいかなる仮定も置くことはできません。たとえば、あるメーカーの白黒画面では黒の値が 1 であったとしても、別のメーカーのカラー画面では黒の値は 255 かもしれません。アプリケーションプログラムでは上記のメソッドを使用して、機種間で共通の色の定義（名前または RGB 値）を各機種固有の色の定義に変換する必要があります。

以下に、正しくないコードの例を示します。

例: グラフィックスキャンバス

```
public class MyCanvas extends Canvas {
    ...
    public void paint(Graphics g) {
        ...
        if(Graphics.getColorOfName(Graphics.BLACK) == 1) { //何かをする。
            // あるメーカーの機種では結果が真になり、このコードが実行されたとしても、
            // 他のメーカーの機種では偽になり、コードはスキップされるかもしれない。
        }
        ...
    }
}
```

i アプリではカラー対応絵文字（i モードサービス向けに NTT ドコモが定義した外字セット）を使用することができます。カラー対応絵文字では、1 つ 1 つの絵文字に対しデフォルトの表示色が割り当てられています。

Graphics クラスには、Graphics オブジェクトに描画色が明示的に指定されている場合に、絵文字をデフォルトの表示色で描画するか Graphics オブジェクトに指定された描画色で描画するかを選択するためのメソッド `setPictoColorEnabled()` が用意されています。

4.2.5 低レベルイベントの処理

Canvas 使用時に適用される低レベルイベント処理の方法は、Panel 使用時における高レベルイベント処理の場合とは異なります。Canvas オブジェクトは低レベルイベントだけを処理します。開発者は Canvas クラスの継承クラスを作成する際に、`processEvent()` メソッドをオーバーライドすることができます。このメソッドには低レベルイベント発生時に、イベント種別とそれに伴うパラメータが渡されます。次の表は、Canvas に通知される有効なイベントタイプを示したものです。これらのイベントタイプは Display クラスに定義されています。

イベントタイプ	イベントタイプ値	パラメータ
KEY_PRESSED_EVENT	0	押されたキーの値
KEY_RELEASED_EVENT	1	離されたキーの値
RESUME_VM_EVENT	4	使用されない
UPDATE_VM_EVENT	6	使用されない

次のコードは、低レベルイベントを処理する `processEvent()` メソッドの実装例です。

例: イベント処理の実装

```
/**
 * このクラスは、特定のハイスコアサーバーに設定されている歴代の殿堂入りハイスコアを表示するときに
 * 使用する。スコアとその名前がアニメーション風に表示される。
 * このクラスは、ユーザー入力を受け取ると、次の画面に移る。
 */
public class HighScoreScreen extends Canvas implements Runnable {
    private boolean done;

    /** このクラスの新しいインスタンスを作成する。 */
    public HighScoreScreen() {
        ...
    }
    /** このインスタンスを初期状態にリセットする。 */
    public void reset(String[] list) {
```

```

        ...
    }
    /** Runnable インタフェースに対して必要。 */
    public void run() {
        ...
    }
    /** この画面を表示する。 */
    public void paint(Graphics g) {
        ...
    }

    /** イベント処理を行うために Canvas の processEvent() をオーバーライドする。 */
    public void processEvent(int type, int param) {
        if (type == Display.KEY_PRESSED_EVENT) {
            done = true;
            SpaceInvaders.WELCOME.reset();
            Display.setCurrent(SpaceInvaders.WELCOME);
        }
    }
}

```

i アプリ再開時は、i アプリの実行が再開されたことを示す **RESUME_VM_EVENT** の通知を行った後、i アプリメインクラスの **activated()** メソッドが呼び出されます。

i アプリ再開時に、描画データの破壊を意味する **UPDATE_VM_EVENT** が発生する場合は、**RESUME_VM_EVENT** が発生した後で **UPDATE_VM_EVENT** が発生します。

注意事項：

- **processEvent()** メソッドの呼び出しは、ユーザーの操作など非同期なイベント発生を契機として行われます。**processEvent()** メソッドの呼び出し時に、それがどのようにスレッドに割り当てられるかはメーカーにより異なりますので注意してください。例えばあるメーカーの機種では、同時に発生した複数のイベントに対して 1 つのスレッドで処理を行うかもしれません。**processEvent()** メソッド内で行われる処理は極力短時間で完結するようにしてください。
- **RESUME_VM_EVENT** は **StarApplication.suspend()** メソッドを呼び出して遷移したサスペンド状態からの再開時にも **processEvent()** メソッドに通知されます。

4.2.6 Canvas における IME の利用

Panel を用いた画面では **TextBox** のユーザー操作により **IME** を起動することができ、ユーザーが入力した文字列を i アプリに与えることができます。これに対し **Canvas** を用いた画面では、**IME** 起動用の API を使用してアプリケーションプログラムから直接 **IME** を起動することができます。この API を利用することで、**Canvas** 使用時も **Panel** 使用時と同様にユーザーが入力した文字列を i アプリに与えることが可能となります。

アプリケーションプログラムが **Canvas.imeOn()** メソッドを呼び出すと **IME** が起動されます。このメソッドは非ブロック型であり、**IME** の起動が完了するとアプリケーションプログラムに復帰します。

ユーザーが **IME** の操作を完了（確定またはキャンセル）すると、アプリケーションプログラムは **IME** イベントの通知を受け取ります。開発者は **Canvas.processIMEEvent()** メソッドをオーバーライドすることにより、**IME** の処理結果を受け取ることができます。

次のコードは、**IME** を利用する **Canvas** の例です。

例: **Canvas** からの **IME** の利用

```

class MainCanvas extends Canvas {
    String inputString = "";

```

```

public void processEvent(int type, int param) {
    // ファンクションキー1のリリースでIMEを起動する。
    if(type==Display.KEY_RELEASED_EVENT&&param==Display.KEY_SOFT1)
        imeOn("initial string", TextBox.DISPLAY_ANY, TextBox.KANA);
    // IMEの起動が完了するとアプリケーションプログラムに制御が戻る。
}

public void paint(Graphics g) {
    g.lock();
    g.clearRect(0,0,200,200);
    g.drawString("InputString: '" + inputString + "'", 0, 20);
    g.unlock(true);
}

public void processIMEEvent(int type, String text) {
    // IMEのユーザー操作が完了するとこのメソッドが呼び出される。
    // IMEイベントのパラメータには、イベントタイプ（確定またはキャンセル）と
    // 入力文字列がある。
    if (type==Canvas.IME_CANCELED) inputString = "[CANCELED]";
    else if (type==Canvas.IME_COMMITTED) inputString = text;
    repaint();
}
}

```

【Star-1.2】

携帯電話の機種によっては、マルチリンガル文字（日本語、英語以外の言語の文字）に対応している場合があります。**Canvas.imeOn()**メソッドでのマルチリンガル文字入力に対応するため、Star-1.2 プロファイル以降、**Canvas** の初期入力モード定数が拡張されています。

Star-1.2 プロファイルでサポートされる追加の初期入力モードは以下の通りです。

- **TRADITIONAL_HANZI**：中国語(繁体字)入力モード
- **SIMPLIFIED_HANZI**：中国語(簡体字)入力モード
- **HANGUL**：韓国語(ハングル)入力モード
- **PORTUGUESE**：ポルトガル語入力モード

ただし、マルチリンガルに対応するかどうか、および対応する場合にどの言語をサポートするかはメーカーにより異なります。また、端末設定によりマルチリンガル文字への対応を有効としていない場合は、これらの指定を行っても初期入力モードとして有効になりません。

注意事項：

- IME が起動している状態では、バックグラウンドのスレッドが IME のモダル操作に干渉しないようにアプリケーションプログラム側では注意が必要です。例えば IME が起動されているときにバックグラウンドスレッドが背後の **Canvas** を他のフレームに切り替えようとした際の動作は、機種により異なりますので注意してください。
- **imeOn()**メソッドの呼び出しは、その **Canvas** がカレントフレームに設定されていない状態や **Dialog** が表示されている状態では無視されます。
- **processIMEEvent()**メソッドの呼び出しは、ユーザーの IME 操作完了による非同期なイベント発生を契機として行われます。**processIMEEvent()**メソッドの呼び出し時に、それがどのようにスレッドに割り当てられるかはメーカーにより異なりますので注意してください。例えばあるメーカーの機種では、IME イベントとその他のイベントに対して 1 つのスレッドで処理を行うかもしれません。**processIMEEvent()**メソッド内で行われる処理は極力短時間で完結するようにしてください。
- **imeOn()**メソッドのバリエーションで、IME で入力可能なテキストの長さを制限するための引数を持つものもあります。ただしこのメソッドのサポートはオプションであり、メーカーによってはサポートされない場合があります。

4.2.7 イメージリファレンスによる直接描画

イメージリファレンスは、ヒープメモリの使用量を極力抑えつつイメージ描画を行うために用意された機構です。

通常のイメージ描画では、描画命令を実行する前にアプリケーションプログラムが **MediaImage** オブジェクトを用意します。**MediaImage** では、各々のオブジェクトが個々に、

- ・ オリジナルデータ（GIF 形式や JPEG 形式などの画像ファイルデータ）
- ・ 内部データ（オリジナルデータを画素情報などの形に変換したデータ）

といったデータを持っています。（4.5.7「メディアデータ利用時のメモリ管理」も参照してください）。従って、イメージをあらかじめ多数保持しておくような i アプリでは、これらのデータを保持するために相当量のヒープメモリが常時消費されていることになります。

イメージリファレンスを用いたイメージ描画では、上記のようなデータの展開と開放を描画命令実行時に自動的に行います。ヒープメモリを要する処理が描画命令内に局所化されるため、使用メモリ量の少ないアプリケーションプログラムを容易に記述することができます。イメージリファレンスによる描画の使用は、利用できるヒープメモリ量が限られているミニアプリにおいて特に有効です。

イメージリファレンスオブジェクトは、**com.docomostar.media.ImageReference** クラスの static メソッド **createImageReference()** に、ScratchPad または JAR リソースファイル中の画像データの URL を指定することで生成します。またイメージリファレンスの描画は、**Graphics.drawImageReference()** メソッドを使用して行います。

例：イメージリファレンスの利用

```
ImageReference ref= ImageReference.createImageReference("scratchpad:///0");
Graphics g;
:
g.drawImageReference(ref, 0, 0, true);
```

なお、イメージリファレンスを用いた描画では描画命令のたびに画像データの展開処理が行われるため、描画速度は遅くなります。この速度低下を補うために、キャッシュの仕組みが用意されています。キャッシュを使用する際には、**Graphics.drawImageReference()** メソッドの第 4 引数 **forceReload** に **false** を指定します。

コード例は以下のとおりです。

```
ImageReference ref= ImageReference.createImageReference("scratchpad:///0");
Graphics g;
:
g.drawImageReference(ref, 0, 0, false); // 1 回目は低速。
g.drawImageReference(ref, 0, 0, false); // キャッシュされているため高速に描画される。
```

この場合、1 回目の描画速度は画像展開などのオーバーヘッドがかかるため低速ですが、2 回目以降はキャッシュ内に展開された画像が再使用されるため、パフォーマンスの向上が期待できます。

なお、読み込むべき画像データの内容を更新したなどの理由でキャッシュを読み込ませたくない場合には、**forceReload** 引数に **true** を指定することでキャッシュを破棄して最新のデータでの描画を行うことができます。

注意事項：

- キャッシュ中のデータの管理・更新の方法はメーカーによって異なります。
- 1つの画像データだけでキャッシュ領域に収まらない場合にはキャッシュは行われません。このとき、その描画データと同じ JAR や ScratchPad を参照する古いキャッシュデータが存在した場合、その古いキャッシュデータは削除されます。なお、その際にキャッシュ領域中の他のキャッシュされているデータが削除されることはありません。
- キャッシュ領域は展開された内部形式のデータ量換算で概ね 512K バイトが用意されます。なお、画像がどのように内部形式に展開されるかはメーカーによって異なります。

4.3 ファンクションキー/決定キー/上下左右キー矢印

i アプリ対応携帯電話には4つのファンクションキーが設けられます。ファンクションキーの外観や場所はメーカーによって異なります。前述の通り、ファンクションキーを使用するとユーザーインタフェースの使用感が大幅に改善されることがあります。i アプリの設計ではファンクションキーの使用を考慮すべきですが、特定の外観や場所に依存することは避けるようにしてください。

一般に、アプリの描画領域の下部にはファンクションキーや決定キーラベル、および方向キーの矢印ラベルを表示するための領域が設けられています、これらのラベルは必要に応じて i アプリから設定することができます。

以下は、ファンクションキーを設定した画面の例です。

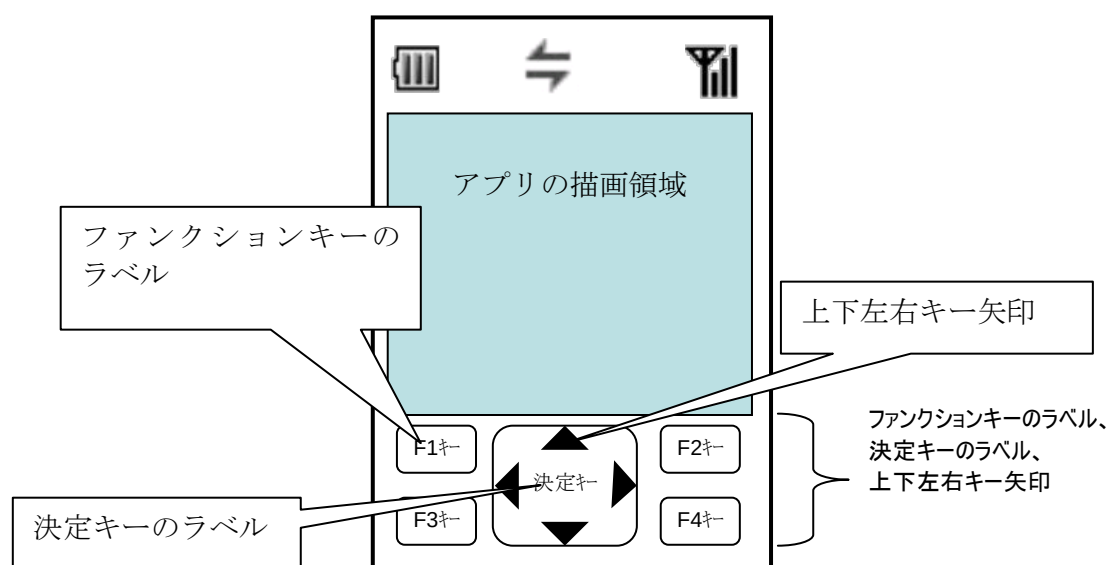


図 9 ファンクションキー/決定キー/上下左右キー矢印の表示例

これらのラベルを設定するメソッドについてそれぞれ解説します。

- **setSoftLabel()**
ファンクションキー、決定キーのラベル文字列を設定します。ラベルを設定したフレー

ムがカレントフレームに設定されると携帯電話上の画面に設定した文字列が表示されます。

各ファンクションキーラベルには全角 4 文字（半角 8 文字）まで、決定キーラベルには全角 2 文字（半角 4 文字）まで設定することができます。

- **setSoftLabelVisible()**
ファンクションキーラベル、決定キーラベル、および上下左右キー矢印の表示のオン、オフを切替えます。
- **setSoftArrowLabel()**
上下左右キー矢印の表示のオン、オフを設定します。
setSoftLabelVisible()メソッドにより上下左右キー矢印を表示に設定している場合に、どの方向の矢印を表示するかを設定します。非表示に設定されている場合、このメソッドを呼び出しても矢印は表示されません。

これらのメソッドはいずれも **Frame** クラスで定義されています。しかし、**Frame** クラスのサブクラスによって利用可能なメソッドは異なります。

以下に、各メソッドと **Frame** クラスの各サブクラスのメソッド呼び出し可否を示します。

メソッド	Canvas	Panel	Dialog
setSoftLabel()	○	○	△ (何もしない)
setSoftLabelVisible()	○	×	×
setSoftArrowLabel()	○	×	×

○…呼び出し可能 ×…呼び出し不可（例外が発生する。） △…その他

注意事項：

- **setSoftLabelVisible()**メソッドは i アプリの全画面表示をサポートしない機種では使用することはできません。

4.4 イベントリスナー

イベントリスナーとは、イベントを通知するオブジェクト (**Panel** や **MediaPresenter** など) に登録され、特定タイプのイベントを待ち受けるオブジェクトです。イベントが発生すると、オブジェクトは適切なイベントタイプを構築し、そのイベントを対応するイベントリスナーに通知します。ユーザーインターフェースに関連する基本的なイベントリスナーには、次の 5 つのタイプがあります。

- **ComponentListener**
- **SoftKeyListener**
- **KeyListener**
- **MediaListener**
- **TimerListener**

ダイアログで使用されるイベントリスナーはありません。ダイアログを画面に表示するには、**Dialog.show()**メソッドを使用します。**Dialog.show()**メソッドは、ユーザーがダイアログとの対話を終了したときにアプリケーションプログラム側に制御を戻します。

Dialog.show()メソッドの戻り値は、ユーザーがダイアログ内のどのボタンを押したかを表しています。

注意事項：

- リスナーメソッドの呼び出しは、ユーザーの操作やタイマーなど非同期なイベント発生を契機として行われます。リスナーメソッドの呼び出し時に、それがどのようにスレッドに割り当てられるかはメーカーにより異なりますので注意してください。例えばあるメーカーの機種では、同時に発生した複数のイベントに対して1つのスレッドで処理を行うかもしれません。リスナーメソッド内で行われる処理は極力短時間で完結するようにしてください。

4.4.1 ComponentListener

ComponentListener インタフェースは、ディスプレイに表示されている **Panel** 上の UI コンポーネントに対して起動されたイベントを待ち受けるときに使用します。このインタフェースのメソッドは **componentAction()** だけです。コンポーネントリスナーの定義と登録は、以下のように行います。

- リスナーとして使用するクラスに、**ComponentListener** インタフェースおよび **componentAction()** メソッドを実装します。コンポーネントイベントが発生すると、**componentAction()** メソッドが呼び出され、イベントタイプ、イベントが発生したコンポーネント、およびその他の必要なパラメータがイベントリスナーに渡されます。
- **Panel** オブジェクトに対して **setComponentListener()** メソッドを呼び出し、上記で実装したリスナークラスのインスタンスを登録します。

ComponentListener には、次の3つのイベントタイプが定義されています。

- **BUTTON_PRESSED:**
Button が押されたときに発生するイベント。イベントパラメータは使用しません。
- **SELECTION_CHANGED:**
ListBox の選択内容が確定されたときに発生するイベント。イベントパラメータとして、選択または状態変更された項目のインデックスが渡されます（複数選択リストでは選択された項目の中で最も小さいもののインデックスが渡されます）。
- **TEXT_CHANGED:**
TextBox の文字入力が終了したときに発生するイベント。イベントパラメータは使用しません。

次に、**ComponentListener** を内部クラスとして定義し、使用する例を示します。

例: **ComponentListener** を実装する内部クラス

```
/**
 * ラベルウィジェットとそれに対して呼び出された setSize メソッドの機能を示すクラス。
 * このクラスは 1 つのボタンと 2 つのラベルウィジェットを作成する。
 * ラベルウィジェットの可視/不可視の機能を示すため、ボタンウィジェットを使用する。
 */
public class LabelDemo extends Panel {
    Label lbl1, lbl2;
    Button btn1;
    int count = 0;

    LabelDemo() {
        lbl1 = new Label("Label1");
        lbl2 = new Label("This is a Label2");
        btn1 = new Button("Button1");
        lbl1.setSize(25,15); // ラベルのサイズを設定する。

        this.add(btn1);
        this.add(lbl1);
        this.add(lbl2);
        ListenerClass lclass = new ListenerClass();
        this.setComponentListener(lclass);
    }
}
```

```

}

/**
 * ボタンイベントを処理する内部クラス
 */
class ListenerClass implements ComponentListener {
/**
 * イベントが Component c で発生したことを示す。
 *
 * @param c このイベントが発生した Component
 * @param type イベントのタイプ
 * @param param イベントのパラメータ
 */
    public void componentAction(Component c, int type, int param) {
        /**
         * コンポーネントアクションが行われた。コンポーネントがボタンの場合、カウントが 0 なら、
         * このパネルの最初のラベルが不可視に、カウントが 1 なら、同じラベルを可視にそれぞれ
         * 設定する。
         */
        if (c == btn1) {
            if (count == 0) {
                lbl1.setVisible(false);
                count++;
            }
            else {
                lbl1.setVisible(true);
                count=0;
            }
        }
    }
}

```

注意事項：

- TextBox および ListBox は、それらのクラスが備えるメソッドで TextBox に設定されているテキストや ListBox の選択状態を変更することができます。このように、ユーザーの操作ではなくメソッドを使用して状態の変更を行った場合もコンポーネントイベントが通知されます。

4.4.2 SoftKeyListener

SoftKeyListener インタフェースは、4つのファンクションキーが操作されるのを待ち受けるときに使用します。このインタフェースで定義されているメソッドは、**softKeyPressed()** と **softKeyReleased()** の2つです。ファンクションキーリスナーの定義と登録は、以下のようになります。

- リスナーとして使用するクラスに、SoftKeyListener インタフェースおよび **softKeyPressed()**、**softKeyReleased()** の2つのメソッドを実装します。ファンクションキーが押されると **softKeyPressed()** メソッドが、放されると **softKeyReleased()** メソッドが呼び出されます。
- Panel オブジェクトに対して **setSoftKeyListener()** メソッドを呼び出し、上記で実装したリスナークラスのインスタンスを登録します。

注意事項：

- ファンクションキーは、Canvas から也可以使用することができます。その場合は、低レベルイベントの処理方法に従ってファンクションキーリスナーの代わりに **Canvas.processEvent()** を使用します（通常のキー操作と同様に、**KEY_PRESSED_EVENT** または **KEY_RELEASED_EVENT** が通知されます）。

4.4.3 KeyListener

KeyListener インタフェースは、ファンクションキー以外のいずれかのキーが操作されるのを待ち受けるときに使用します。このインタフェースで定義されているメソッドは、**keyPressed()**と**keyReleased()**の2つです。キーリスナーの定義と登録は、以下のように行います。

- リスナーとして使用するクラスに、**KeyListener** インタフェースおよび **keyPressed()**、**keyReleased()**の2つのメソッドを実装します。キーが押されると **keyPressed()**メソッドが、放されると **keyReleased()**メソッドが呼び出されます。
- Panel** オブジェクトに対して **setKeyListener()**メソッドを呼び出し、上記で実装したリスナークラスのインスタンスを登録します。

なお、**Panel** では、画面のスクロールやコンポーネントの操作などのために方向キーおよび選択キーを使用します。高レベル **API** 利用時におけるこれらのキーのキーイベントについて以下のように規定されています。

- 方向（上下左右）キーはキーイベントを発生しません。これらのキーは、スクロールおよびフォーカス制御のために常に **Panel** で処理されます。
- 選択（決定）キーは、フォーカスを得ているコンポーネントがディスプレイ上に表示されている場合に限りキーイベントを発生しません。フォーカスを得ているコンポーネントがディスプレイ上に表示されている状態では、選択キーはそのコンポーネントの操作のために **Panel** で処理されます。

4.4.4 MediaListener

MediaListener インタフェースは、メディアデータ再生においてメディアイベント（**AudioPresenter** または **VisualPresenter** が通知するイベント）を待ち受けるときに使用します。このインタフェースで定義されているメソッドは **mediaAction()**だけです。メディアリスナーの定義と登録は、以下のように行います。

- リスナーとして使用するクラスに、**MediaListener** インタフェースおよび **mediaAction()**メソッドを実装します。メディアイベントが発生すると、このメソッドが呼び出されます。
- MediaPresenter** オブジェクトに対して **setMediaListener()**メソッドを呼び出し、上記で実装したリスナークラスのインスタンスを登録します。

AudioPresenter には、次の7つのイベントタイプが定義されています。

- AUDIO_COMPLETE** : サウンドメディアデータの再生完了イベント
- AUDIO_PLAYING** : サウンドメディアデータの再生開始イベント
- AUDIO_STOPPED** : サウンドメディアデータの再生停止イベント
- AUDIO_PAUSED** : サウンドメディアデータの再生一時停止イベント
- AUDIO_RESTARTED** : サウンドメディアデータの再生再開イベント
- AUDIO_SYNC** : サウンドメディアデータの同期イベント
- AUDIO_LOOPED** : サウンドメディアデータのループ再生イベント

VisualPresenter には、次の3つのイベントタイプが定義されています。

- VISUAL_COMPLETE** : 画像メディアデータの再生完了イベント
- VISUAL_PLAYING** : 画像メディアデータの再生開始イベント
- VISUAL_STOPPED** : 画像メディアデータの再生停止イベント

なお、メディアデータの詳細については 4.5 項を参照してください。

4.4.5 TimerListener

TimerListener インタフェースは、タイマー終了イベントを待ち受けるときに使用します。このインタフェースで定義されているメソッドは `timerExpired()` だけです。タイマーリスナーの定義と登録は、以下のようになります。

- リスナーとして使用するクラスに、TimerListener インタフェースおよび `timerExpired()` メソッドを実装します。タイマーが終了すると、このメソッドが呼び出されます。
- Timer オブジェクトに対して `setListener()` メソッドを呼び出し、上記で実装したリスナークラスのインスタンスを登録します。

4.5 マルチメディアデータの使用

4.5.1 i アプリが利用可能なマルチメディアデータ

i アプリ実行環境には、マルチメディアデータとして画像（静止画像および動画像）やサウンドを取り扱うための機能が備えられています。以下に、i アプリ実行環境で標準的に取り扱うことのできるマルチメディアデータのフォーマットを記載します。

● GIF 形式画像データ：

GIF87a、GIF89a をサポートします。通常の画像の他にインターレース GIF、透過 GIF、アニメーション GIF を取り扱うことができます。アニメーション GIF は、動画像として **VisualPresenter** クラスで再生します（4.5.2 項参照）。

● JPEG 形式画像データ：

各機種で共通的に使用可能な形式は JFIF Baseline エンコーディングです。メーカーによっては EXIF を取り扱えるものもあります。拡張 DCT 方式など非対応のエンコーディングを用いたデータを使用すると例外が発生する場合があります。

● BMP 形式画像データ：

以下の条件を満たす BMP 形式データを使用することができます。

- Windows BMP 形式で、bottom-up DIB であること
- ピクセルあたりのビット数は 1, 4, 8, 16, 24 のいずれかであること
- 無圧縮であること（圧縮形式が BI_RGB であること ただし 1 ピクセル 16 ビットの場合は BI_BITFIELDS (RGB 5-5-5 および RGB 5-6-5) にも対応)

なお、BMP 形式のデータは i アプリ内でのみ使用可能であり、それらをネイティブの画像保存領域（マイピクチャ）に保存することはできません。

● PNG 形式画像データ：

PNG 形式データをサポートします。また、カラータイプが 4 あるいは 6 の場合はアルファ値指定、カラータイプが 0、2、3 の場合は tRNS チャンクにより透過指定された PNG 形式データもサポートしています。

なお、PNG 形式のデータは i アプリ内でのみ使用可能であり、それらをネイティブの画像保存領域（マイピクチャ）に保存することはできません。

● MFi 形式サウンドデータ：

i モードにおける標準的なサウンドデータである、MFi (Melody For i-mode) 形式のサウンドデータをサポートします。サウンドの基本的な再生方法については 4.5.2 項を参照してください。MFi 形式サウンドデータでは、ADPCM や 3D サウンドなど多彩な拡張機能を利用することもできます。

- SMF 形式サウンドデータ :

FOMA 携帯電話では、サウンドデータとして MFi 形式の他に SMF (Standard MIDI File) 形式のサウンドデータを使用することもできます。サウンドの基本的な再生方法については 4.5.2 項を参照してください。

- i モーション形式動画データ :

i モーション形式動画データを取り扱うことができます (4.5.5 項参照)。なお、i モーション形式動画データには複数のバージョンがあり、各機種についてそれぞれに適したものを使用する必要があります。どの機種がどのバージョンのものをサポートしているかについては、別途 NTT ドコモより公開されます。

- SWF 形式データ

携帯電話の中には SWF 形式データを取り扱うことができるものがあります。詳細は「i アプリコンテンツ開発ガイド i アプリオプション・i アプリ拡張編」を参照してください。

【Star-2.0】

PNG 形式画像データのサポートは Star-2.0 プロファイルで追加されました。

マルチメディアデータの共通インタフェースである **MediaResource** には以下の機能があります。

- **isRedistributable()**
メディアデータが再配布可能に設定されているかどうかを取得します。再配布可否が設定できるかどうか、およびその設定方法は、メディアデータの種類および携帯電話プラットフォームにより異なります。再配布不可としてネイティブの保存領域に保存されたデータは、メール添付や赤外線機能などを使用して携帯電話外部に持ち出すことはできません。
- **setRedistributable()**
メディアデータに、再配布可/不可の設定を行います。
- **getProperty()**
メディアデータのプロパティ値を取得します。サポートされるプロパティの種別は、メディアデータの種別ごとに定められています。現在、以下のプロパティがサポートされています。
 - **MediaSound.AUDIO_3D_RESOURCES**
メディアサウンドのデータに 3D サウンド制御情報 (定位情報) が埋め込まれている場合、それを使用するためには 3D サウンド制御リソースをいくつ必要とするかを取得します。3D サウンド制御および 3D サウンド制御リソースについては第 13 章を参照してください。
 - **MediaMovie.MP4_AUDIOTRACK**
メディアムービーのデータが i モーション形式データである場合、データに含まれるオーディオトラックの数 (数を表す数字文字列) を取得します。
 - **MediaMovie.MP4_TEXTTRACK**
メディアムービーのデータが i モーション形式データである場合、データに含まれるテキストトラックの数 (数を表す数字文字列) を取得します。
 - **MediaMovie.MP4_VIDEOTRACK**
メディアムービーのデータが i モーション形式データである場合、データに含まれるビデオトラックの数 (数を表す数字文字列) を取得します。

なお、**MediaManager** クラスでは、複数の **MediaImage** オブジェクト、または複数の **MediaSound** オブジェクトを一括して利用可能状態にするための **static** メソッド **use()** が設けられています。この **use()** メソッドでは、メディアデータオブジェクトの利用を一度きりとするかどうかを指定することができます。メディアデータオブジェクトにおける「一度きり」の利用指定については、4.5.7 項も参照してください。

4.5.2 プレゼンタの利用

i アプリ実行環境では、アニメーションを含む画像の再生やサウンドの再生を行う機能が提供されます。画像を再生するクラスとサウンドを再生するクラスとしてそれぞれ、**VisualPresenter** と **AudioPresenter** があります。

注意事項：

- **VisualPresenter** はコンポーネントであり、高レベル API にて **Panel** オブジェクトに追加して使用します。**VisualPresenter** では、標準的に再生可能な画像として GIF アニメーションをサポートしています。i モーション形式データを再生する場合は、**NativeMoviePlayer** を使用してください。(4.5.5 項参照)。
- **AudioPresenter** はコンポーネントではありません。**AudioPresenter** は **Panel** 表示時、**Canvas** 表示時のいずれの場合も使用することができます。
- NTT ドコモの提供する i アプリ開発環境では、**AudioPresenter** は MFi 形式データと MIDI 形式データをサポートしています。i アプリ開発環境を使用した実際の開発手順などについては第 16 章を参照してください。
- FOMA 携帯電話の中には、i アプリやブラウザ、通話などを、タスクを切り替えながら並列に起動して利用できるものがあります。このような携帯電話において、別のタスクでサウンドや音声を取り扱う機能が起動されている状態では、i アプリからサウンドや音声を取り扱う機能 (**AudioPresenter** によるサウンド再生など) を使用すると例外 (**UIException**) が発生します。
i アプリのサウンド・音声関連機能と同時に使用できない機能には、通話機能や音楽プレーヤーなどがあります (機種によっては音楽プレーヤーを実行中に i アプリがサウンド再生を行っても例外を発生させず、音楽プレーヤーと i アプリのサウンドの切り替え制御などを行うものもあります)。
- メディアデータがセットされていないメディアプレゼンタに対し **stop()** メソッドを呼び出すと例外 (**UIException**) が発生します。メディアプレゼンタにメディアデータがセットされているかどうかは **MediaPresenter.getMediaResource()** メソッドを使用して判定してください。

次の各項では、**VisualPresenter** と **AudioPresenter** について説明します。

VisualPresenter

VisualPresenter には以下の項目を調整するために属性があります。

- イメージ表示位置の X 座標 (**VisualPresenter.IMAGE_XPOS**)
- イメージ表示位置の Y 座標 (**VisualPresenter.IMAGE_YPOS**)

以下に、**VisualPresenter** クラスを使って画像を再生する例を示します。

この例では、画像ファイルとして、i アプリの JAR ファイルに同梱されたリソースを使用しています ("**resource:///**" で始まる URL の使用)。リソースについては、第 7 章を参照してください。なお、この URL にはリソースだけではなく、Web サーバー上のファイルを示す HTTP の URL や ScratchPad 上の位置を示す URL、さらにはデータファイルのバイトイメージを指定することもできます。バイトイメージは **byte** 配列または **InputStream** の形式で与えることができます。

例： 画像ファイルの表示

```
public class VisualPresenterDemo extends Panel {
    int          px, py;          // 現在の画像の表示位置 (VisualPresenter 内)
    VisualPresenter theVP;        // プレゼンター
    Button        startButton;    // GIF アニメーション再生ボタン
```

```

Button      stopButton;    // GIF アニメーション停止ボタン
ListBox     repeatChoice;  // 繰り返し再生指示用チェックボックス
Ticker      ticker;        // キーマップ説明用ティッカーテープ

static String instr = "Control ImagePos: Key 2-Down 8-Up 4-Left 6-Right";

/**
 * VisualPresenterDemo コンストラクタ
 * 必要な画像データを取得して、VisualPresenter に引き渡す。
 */
VisualPresenterDemo() {
    // 画像データを取得する。
    MediaImage mi =
        MediaManager.getImage("resource:///uidemo/img/singleloop.gif");

    // use()メソッドを呼び出すまで画像データを利用することはできない。
    try {
        mi.use();
    } catch (ConnectionException ce) {
        // ここで入出力の例外を処理する。
    } catch (MediaException me) {
        // ここで Media の例外を処理する。
    }

    px      = 0;
    py      = 0;

    theVP      = new VisualPresenter();
    theVP.setImage(mi);

    ticker      = new Ticker(instr);

    startButton = new Button("start");
    stopButton  = new Button("stop ");

    repeatChoice = new ListBox(ListBox.CHECK_BOX);
    repeatChoice.append("Repeat");

    add(theVP);
    add(ticker);
    add(startButton);
    add(stopButton);
    add(repeatChoice);

    // メディアリスナーを設定する。
    theVP.setMediaListener(new MediaListenerClass());

    setTitle("Status Line");
    this.setComponentListener(new ListenerClass());
    this.setKeyListener(new KeyListenerClass());
}

class KeyListenerClass implements KeyListener {
    /**
     * キーリリースイベントは無視する。
     */
    public void keyReleased(Pane1 p, int key) {}

    /**
     * キープレスイベントにより画像表示位置の調整を行う
     *
     * @param p イベントの発生した Pane1
     * @param key 押されたキーの値
     */
    public void keyPressed(Pane1 p, int param1) {
        if (VisualPresenterDemo.this == p) {

```

```

        if (param1 == Display.KEY_8) {                // 画像の位置 (下)
            py--;
        } else if (param1 == Display.KEY_2) {         // 画像の位置 (上)
            py++;
        } else if (param1 == Display.KEY_4) {         // 画像の位置 (左)
            px++;
        } else if (param1 == Display.KEY_6) {         // 画像の位置 (右)
            px--;
        }

        // VisualPresenter の setAttribute()メソッド呼び出しにより
        // 画像表示位置を設定する。
        theVP.setAttribute(VisualPresenter.IMAGE_XPOS, px);
        theVP.setAttribute(VisualPresenter.IMAGE_YPOS, py);
    }
}

class ListenerClass implements ComponentListener {

    /**
     * コンポーネントイベントにより再生の開始、停止を制御する
     */
    * @param c        イベントの発生したコンポーネント
    * @param type      イベントタイプ
    * @param param     イベントパラメータ
    */
    public void componentAction(Component c, int type, int param) {
        if (c == startButton) {                        // 再生ボタンを押すと再生開始
            theVP.play();
        } else if (c == stopButton) {                  // 停止ボタンを押すと再生停止
            theVP.stop();
        }
    }
}

class MediaListenerClass implements MediaListener {

    /**
     * VisualPresenter からのイベントを受け取り、必要に応じて再度再生を行うための
     * メディアリスナー。ループ再生機能はここで実現される。
     */
    public void mediaAction(MediaPresenter source, int type, int param) {

        if (source == theVP) {
            switch (type) {

                // 画像の再生完了イベントはここで処理される。Panel タイトルに
                // 再生を完了したことを表示し、必要に応じて再度再生を繰り返す。
                // この処理は非ループ再生型 GIF アニメーションでのみ有用。
                case VisualPresenter.VISUAL_COMPLETE :
                    setTitle("Complete");

                    if ((repeatChoice.isIndexSelected(0))) {
                        ((VisualPresenter) source).play();
                        setTitle("[Comp]Play again");
                    }
                    break;

                // 画像の再生開始イベントはここで処理される。
                // Panel タイトルに再生を開始したことを表示する。
                case VisualPresenter.VISUAL_PLAYING :
                    setTitle("Playing");
                    break;

                // 画像の再生停止イベントはここで処理される。
                // Panel タイトルに再生を停止したことを表示する。
                case VisualPresenter.VISUAL_STOPPED :
                    setTitle("Stopped");
            }
        }
    }
}

```

```

        break;
    }
}
}

/**
 * MediaImage は不要になったら破棄 (unuse()および dispose()) すべきである。
 * 破棄することにより資源の解放が行われる。ここでは MediaImage を再利用する可能性
 * があるため破棄は行わない。
 */
public void leave() {
}
}

```

注意事項：

- VisualPresenter によるアニメーション GIF 再生について以下の規定があります。
 - 再生可能なフレーム数は、最低限 8 フレームが保証されます。
 - ループ型アニメーション GIF を再生した場合、1 回のループが終了する毎に再生完了イベントが発生します。
 - ループ型アニメーション GIF の最大ループ回数は 16 回です。これを超えたループ回数が指定されている場合でも、16 回ループした時点で再生は停止します。
- メーカーによっては MediaManager.getImage() メソッドの引数に Flash コンテンツのデータを指定して呼び出すことで、MediaImage オブジェクトを取得することをサポートすることがあります。ただし、Flash コンテンツから生成された MediaImage オブジェクトは、VisualPresenter で再生することはできません。

AudioPresenter

AudioPresenter には以下の項目を調整するために属性があります。

- テンポ (AudioPresenter.CHANGE_TEMPO)
- ボリューム (AudioPresenter.SET_VOLUME)
- キー (AudioPresenter.TRANSPOSE_KEY)
- 再生回数 (AudioPresenter.LOOP_COUNT)

以下に、AudioPresenter クラスを使ってサウンドを再生する例を示します。

この例では、サウンドファイルとして、i アプリの JAR ファイルに同梱されたリソースを使用しています ("resource:/// "で始まる URL の使用)。リソースについては、第 7 章を参照してください。なお、この URL にはリソースだけではなく、VisualPresenter における画像ファイルの URL と同様に、Web サーバー上のファイルを示す HTTP の URL や ScratchPad 上の位置を示す URL、さらにはデータファイルのバイトイメージを指定することもできます。バイトイメージは byte 配列または InputStream の形式で与えることができます。

例: サウンドファイルの再生

```

public class AudioPresenterDemo extends Panel {
    ListBox      loopChoice;    // ループ再生フラグ
    Button       playButton;    // サウンド再生ボタン
    Button       stopButton;    // サウンド停止ボタン
    Label        statusLabel;   // ステータス表示ラベル
    AudioPresenter theAP;       // AudioPresenter
    MediaSound    theMediaSound; // MediaSound

    /**

```

```

* AudioPresenterDemo コンストラクタ
* 再生/停止用の 2 つのボタンと、ループ再生を選択するためのチェックボックスを作成する。
*/
AudioPresenterDemo() {
    loopChoice = new ListBox(ListBox.CHECK_BOX);
    loopChoice.append("loop");
    playButton = new Button("play");
    stopButton = new Button("stop");
    statusLabel = new Label("status line");

    theMediaSound =
        MediaManager.getSound("resource:///uidemo/img/music.mld");

    // use()メソッドを呼び出すまでサウンドデータを利用することはできない。
    try {
        theMediaSound.use();
    } catch (ConnectionException ce) {
        // ここで入出力の例外を処理する。
    } catch (MediaException me) {
        // ここで Media の例外を処理する。
    }

    theAP = AudioPresenter.getAudioPresenter();
    theAP.setSound(theMediaSound);

    this.add(playButton);
    this.add(stopButton);
    this.add(loopChoice);
    this.add(statusLabel);
    this.setComponentListener(new ListenerClass());
    theAP.setMediaListener(new MediaListenerClass());
}

class ListenerClass implements ComponentListener {
    /**
     * サウンドデータの再生/停止を制御するためのボタンのイベントを受け取る
     * リスナークラス。
     */
    public void componentAction(Component c, int type, int param) {
        if (c == playButton) {
            theAP.play();
        } else if (c == stopButton) {
            theAP.stop();
        }
    }
}

class MediaListenerClass implements MediaListener {
    /**
     * AudioPresenter からのイベントを受け取り、必要に応じて再度再生を
     * 行うためのメディアリスナー。ループ再生機能はここで実現される。
     */
    public void mediaAction(MediaPresenter source, int type, int param) {
        if (source == theAP) {
            switch(type) {
                // サウンドの再生完了イベントはここで処理される。
                // ステータス表示ラベルに再生を完了したことを表示し、
                // 必要に応じて再度再生を繰り返す。
                case AudioPresenter.AUDIO_COMPLETE:
                    statusLabel.setText("complete");
                    if (loopChoice.isIndexSelected(0))
                        ((AudioPresenter)source).play();
                    break;

                // サウンドの再生開始イベントはここで処理される。
                // ステータス表示ラベルに再生を開始したことを表示する。
                case AudioPresenter.AUDIO_PLAYING:
                    statusLabel.setText("playing ");
                    break;
            }
        }
    }
}

```

```

// サウンドの再生停止イベントはここで処理される。
// ステータス表示ラベルに再生を停止したことを表示する。
        case AudioPresenter.AUDIO_STOPPED:
            statusLabel.setText("stopped ");
            break;
    }
}
}

/**
 * この画面が非表示になる際に、サウンドの再生を停止するため、AudioPresenter.stop()メソッド
 * の呼び出しを行う。MediaSound をこれ以上使用しない場合は、MediaSound.unuse()メソッド
 * および MediaSound.dispose()メソッドも呼び出すべきである（各種資源の解放が行われる）。
 * ここでは MediaSound を再度利用する可能性があるため、これらのメソッドは呼び出さない。
 */
public void leave() {
    theAP.stop();
}
}

```

注意事項：

- AudioPresenter クラスの i モーションのオーディオトラックの再生
(getAudioTrackPresenter()メソッド) 機能はオプションであるため、サポートの有無はメーカーにより異なります。

【Star-2.0】

Star-2.0 プロファイルにて、AudioPresenter クラスの i モーションのオーディオトラックの再生機能は i アプリ基本 API となりました。

4.5.3 複数の AudioPresenter の同時再生

アプリケーションプログラムは、AudioPresenter.getAudioPresenter()メソッドを複数呼び出すことで、複数の AudioPresenter を取得し、それらに対して同時に再生指示を与えることができます。

アプリケーションプログラムから同時再生可能なプレゼンタの数を越えて再生指示を行った場合、どのプレゼンタが有効なものとして再生されるかは、プレゼンタに設定された優先順位属性 (AudioPresenter.PRIORITY) の値により決定されます。同時に 2 つのプレゼンタが再生可能な携帯電話において、中優先度、低優先度の 2 つのプレゼンタを再生中に高優先度のプレゼンタの再生が指示された場合、低優先度のプレゼンタは停止して新たに指示されたプレゼンタの再生が開始されます（中優先度のプレゼンタは継続して再生されつづけます）。なお、停止すべき優先度のプレゼンタが複数存在する場合、それらのうちどのプレゼンタが停止するかはメーカーにより異なります。

優先順位属性をサポートしていない携帯電話では、同時再生可能なプレゼンタの数を越えて再生指示を行った場合、後に再生したプレゼンタの方が再生中のプレゼンタより優先度が高いものとして扱われます。

携帯電話の能力の範囲（同時発音可能数や使用可能なチャンネル数、プレゼンタ数）であれば、同時に再生した音はプレゼンタの優先順位に関係なく全て聞こえます。携帯電話の能力を超えた再生指示が行われた場合には、同時再生可能なプレゼンタの数の範囲内であっても全ての音が聞こえるとは限りません。

AudioPresenter の同時再生に関してポートの概念があります。ポートとは、同時再生可能な **AudioPresenter** の数分用意された仮想的な再生器です。アプリケーションプログラムは、**AudioPresenter.getAudioPresenter()** メソッドのオプションの引数により、個々の **AudioPresenter** をどのポートに割り付けるかを明示的に宣言することができます。ポート番号に割り付けられた **AudioPresenter** と割り付けられていない **AudioPresenter** を同時に再生することはできません。

AudioPresenter にポート指定を行う場合については、各機種で共通に保証される **AudioPresenter** の同時再生可能数は最低 4 つです。

ポート指定を行わない場合の **AudioPresenter** の同時再生可能数は最低 2 つです。

注意事項：

- i アプリではサウンドデータとして **MP3** 形式フォーマットおよび **SMF** 形式フォーマットをサポートしていますが、両者を混在させて同時に再生できるかどうかはメーカーにより異なります。複数のサウンドを同時再生する場合は、サウンドデータのフォーマットをいずれか一方に統一するようにしてください。

4.5.4 サウンド再生の同期イベント

AudioPresenter では、サウンド再生における同期イベント発生機能をサポートしています。同期イベントとは、サウンドの再生進行に同期してメディアリスナーに通知されるメディアイベントです。**AudioPresenter.setAttribute()** メソッドを使用して、同期イベント発生可否属性 (**AudioPresenter.SYNC_MODE**) に対しオン (**AudioPresenter.ATTR_SYNC_ON**) またはオフ (**AudioPresenter.ATTR_SYNC_OFF**) を指定することでイベントの発生有無を制御することができます。

同期イベント機能を利用する場合、サウンドデータを構成するうちの任意の 1 チャンネルをイベント発生源として指定します。指定されたチャンネルの指定されたノートメッセージに連動して、メディアリスナーに同期イベントが通知されます。

なお、イベント発生源として指定されたチャンネルも発音の対象となります。発音が不要な場合、サウンドデータ作成時においてそのチャンネルに対し消音指定を行うようにしてください。

4.5.5 NativeMoviePlayer での i モーションの再生

i アプリでは、i モーション形式データを **com.docomostar.system.NativeMoviePlayer** で再生することができます。i モーション形式データを再生するには **MediaManager.getMovie()** メソッドを呼び出して **MediaMovie** オブジェクトを生成し、それを引数にして **NativeMoviePlayer.play()** メソッドを呼び出します。また、**NativeMoviePlayer.play(String)** の引数に HTTP の URL を指定することで再生することもできます。

再生指示 (**NativeMoviePlayer.play()**) により i アプリは一時中断し、ネイティブの i モーションプレイヤーが起動して、指定した i モーションが再生されます。

再生が終了した理由は、`NativeMoviePlayer.play()`の戻り値により判断することができます。`NativeMoviePlayer.play()`の戻り値に関して以下の値があります。

- `STATUS_COMPLETED` コンテンツの最後まで再生が完了した。
- `STATUS_FAILED` その他の理由で動画の再生が継続できなかった。
- `STATUS_STOPPED` ユーザー操作によって再生が終了した。
- `STATUS_SUSPENDED` 再生中の着信などにより再生が中断された。
- `STATUS_UNSUPPORTED_FORMAT` 端末で再生できないような動画が指定された。

`NativeMoviePlayer.setAttribute(int,int)`を呼び出すことにより i モーション再生時の画面表示モード、およびユーザーインタフェースモードの設定ができます。

• 画面表示モードの指定

画面表示モードは、動画再生時の画面の表示方法を示すモードで、縦画面表示モードと全画面再生モードがあります。どの機種でも使用可能な画面表示モードは縦画面表示モードです。全画面再生モードはオプションであり、使用できるかどうかはメーカーにより異なります。また、i アプリから全画面再生モードを使用した場合のプレーヤーの表示（見映え）は、ネイティブプレーヤーで全画面再生モードを使用した場合のプレーヤーの表示に準じます。

なお、全画面再生モードでテキストトラックを含む i モーションデータを再生できるかどうかはメーカーにより異なります。

画面表示モードは、`NativeMoviePlayer` クラスの `setAttribute()` メソッドにより `SCREEN_MODE` 属性の設定を行なうことにより選択することができます。

• ユーザーインタフェースモードの指定

ユーザーインタフェースモードは、i モーション再生中に、再生されている動画に対して、ユーザーの操作を可能とするかどうかを示すモードです。ユーザー操作の禁止はオプションであり、使用できるかどうかはメーカーにより異なります。ユーザー操作が禁止されている場合、再生中の動画に対してユーザーは早送り等の操作を行うことはできません。ただし、音量調節、オンフックキーはモードに拠らず操作することが可能です。また、タスクキー操作時もモードに拠らず操作することが可能ですが、タスクキー操作時の振る舞いはメーカーによって異なります。

ユーザーインタフェースモードは、`NativeMoviePlayer` クラスの `setAttribute()` メソッドにより `UI_INTERFACE_MODE` 属性の設定を行なうことにより選択することができます。

また、i モーションのダウンロードには、全てのデータを携帯電話上にダウンロードし終わった段階で再生が可能となる通常のダウンロード方法の他に、一定量のデータを取得した時点で再生を開始し、それ以降はデータのダウンロードと再生を並行して行うプログレッシブダウンロードがあります。

`NativeMoviePlayer.play(String)` メソッドで再生指示が行なわれるとデータのダウンロードと再生が並行して行われます。

なお、メーカーによっては、i アプリにおけるプログレッシブダウンロードをサポートしない場合があります。そのような機種では `NativeMoviePlayer.play(String)` メソッドを使用した場合にもプログレッシブダウンロードとはならず、コンテンツ全体がダウンロードされた後に再生が行われます。

i アプリから再生可能な i モーションのデータサイズは、`MediaManager.getMovie()` メソッドや `NativeMoviePlayer.play()` メソッドに HTTP スキームの i モーション URL を指定する場合には最大 150K バイトまでです（本書 5.1 項に記載の HTTP(S)通信サイズの制約を受けることによります）。また、分割して入手した i モーションデータを `ScratchPad` や実行メモリ上で結合するなどの方法により大きな i モーションデータを i アプリ内で生成するような場合

は、500K バイトまでの i モーションデータを再生することができます（メーカーによっては、より大きなサイズの i モーションデータに対応する場合があります）。

【Star-2.0】

Star-2.0 プロファイルに対応する携帯電話では、i アプリから再生可能な i モーションのデータサイズは、**MediaManager.getMovie()** メソッドや **NativeMoviePlayer.play()** メソッドに HTTP スキームに i モーション URL を指定する場合には最大 1024K バイトに拡張されました。また、分割して入手した i モーションデータを ScratchPad や実行メモリ上で結合するなどの方法により大きな i モーションデータを i アプリ内で生成するような場合の再生可能なサイズは 3M バイトに拡張されました（メーカーによっては、より大きなサイズの i モーションデータに対応する場合があります）。

注意事項：

- フル楽曲コンテンツとしての設定が行われている i モーションデータは、**NativeMoviePlayer** クラスで再生することはできません。

4.5.6 マルチメディアデータの外部メモリへの出力制御

i アプリで取り扱うマルチメディアデータ（**MediaResource** 継承インタフェースの実装オブジェクト）は、本書の第 11 章で解説されるようなアプリケーション連携機能（**ImageStore** クラスや **MovieStore** クラスなど）を利用して携帯電話ネイティブのデータ保存領域（以下、内蔵メモリと呼びます）に保存することができます。

また、外部メモリの利用をサポートする機種の中にはコンテンツ保護技術を活用して内蔵メモリのコンテンツを外部メモリに移動したり、さらにそれを他の携帯電話の内蔵メモリに移動することができるものもあります。

i アプリから内蔵メモリへのマルチメディアデータの保存に関しては、保存されたマルチメディアデータがユーザー操作で外部メモリに出力されるような状況において、そのコンテンツに対する保護のレベルを制御することが可能となっています。

以下に、i アプリから内蔵メモリに保存されたマルチメディアデータの、コンテンツの保護レベルの取り扱いについて記載します。

再配布不可情報が設定されていない（再配布が許可された）コンテンツ

再配布不可情報が設定されていないコンテンツは、外部メモリの持つ著作権保護機構の対象外となります。従って、i アプリが再配布不可情報の設定されていないマルチメディアデータを内蔵メモリに保存し、そのデータがユーザー操作によって外部メモリに移される際、そのコンテンツの内容は暗号化されていない平文の状態で外部メモリに記録されます。そのため、このようにして外部メモリに移されたデータは PC などのビューアなどでも閲覧することができます。

再配布不可情報が設定された（再配布が許可されない）コンテンツ

再配布不可情報が設定されているコンテンツは外部メモリの持つ著作権保護機構の対象となり、外部メモリに出力される際に暗号化されます。通常、i アプリから内蔵メモリに保存したコンテンツを外部メモリに移すと、そのコンテンツ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。なお、i アプリが内蔵メモリに保存しようとしているデータについて、保存前に **MediaResource.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 **ImageStore** クラスなどを使用してそのデータを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 (**MediaManager** クラスを使用してマルチメディアデータをオブジェクト化した際には、これらのプロパティは未設定の状態となります)。

4.5.7 メディアデータ利用時のメモリ管理

i アプリが画像やサウンドなどのメディアデータを利用する際には、比較的多くのアプリケーション実行メモリ（ヒープメモリ）が消費されます。一般的な実装ではメディアデータを取り扱う際には、GIF や JPEG などデータ形式毎に固有のオリジナルデータを保持するためのメモリや、それらを内部データ形式に変換したデータを保持するためのメモリが使用されます。

i アプリ実行環境ではメディアデータ使用時に行うメモリ管理を以下のようにモデル化し、i アプリからそれらに対して一定の制御を行うことができます。

オリジナルデータ領域：

GIF や JPEG などのオリジナルデータを保持するために使用されるメモリです。原則として、メディアデータのオブジェクトが生成される時点で確保され、そのオブジェクトに対して **dispose()** メソッドが呼び出されるまで保持されます。

内部データ領域：

オリジナルデータを、システム内部で取り扱うことのできる形式に変換したものを保持するために使用されるメモリです。例えば画像の場合は、GIF や JPEG などのデータを画素毎のデータに展開したものが相当します。原則として、メディアデータのオブジェクトに対して **use()** メソッドが呼び出される時点で確保され、そのオブジェクトに対して **unuse()** メソッドが呼び出されるまで保持されます。

例えば、アプリケーションの起動から終了までの間に複数回 **use()**、**unuse()** を呼び出さない（利用可能状態と利用不可能状態を何度も行き来しない）ことが確実であるメディアデータに対しては、メディアデータを利用可能状態とする際にオリジナルデータを破棄し、その分のメモリを解放することを指示することができます（「一度きり」の利用指定）。

また、短期間だけ利用可能状態とすればよいメディアデータを複数取り扱うような場合には、複数のメディアデータのオブジェクトの間に1つの内部データ領域を引き継ぎながら使用するということを指示することもできます（内部データ領域の再利用）。

これらの機能は、各メディアデータインタフェースなどに規定されている **use()** メソッドの拡張として提供されています。

さらに、イメージリファレンスを用いて描画を行うことで、オリジナルデータを展開しておく領域も節約することができます。イメージリファレンスについては、4.2.7 項も参照してください。

注意事項：

- i アプリ実行環境の実装によっては、本項で解説するメモリ管理のモデルがそのまま適用できない場合があります。そのような実装ではi アプリ側でメモリ利用に関する制御を行っても、効果が少ないか、もしくは効果が現れない場合があります。
- 取り扱うメディアデータの種類によっては、内部データ領域の再利用は適用できない場合があります。内部データ領域の再利用が適用可能なメディアデータは、静止画像（アニメーション GIF は含まれません）とサウンド（MFi と SMF）です。
- 「一度きり」の利用指定が行われたメディアデータに対して複数回 **use()** メソッドを呼び出すと例外が発生します。また、この指定が行われたメディアデータオブジェクトはオリジナルデータを保持していないため、**ImageStore** などによるネイティブのデータ領域への保存はできないことに注意してください。

- 内部データ領域の再利用では、一度確保されたメモリを再利用時に拡張することはできません。内部データ領域が最初に確保される際に、再利用時にも十分なサイズとなる領域が確保されるように考慮しなければなりません。内部データ領域のサイズの考え方は、メディアデータの種別によって異なります。これらの詳細は、API リファレンスにおける各メディアデータインタフェースの `use()` メソッドの項を参照してください。
- `MediaImage` オブジェクトから取得 (`getImage()` メソッド) した `Image` オブジェクトは、内部的に `MediaImage` オブジェクトの持つ内部データ領域を参照しています。従って、このようにして取得した `Image` オブジェクトを使用している間は、対応する `MediaImage` オブジェクトの内部データ領域が破棄されないようにしなければなりません。
- メディアデータの生成時にストリームを指定した場合もオリジナルデータ領域が使用されます。ストリームが指定されたメディアデータのオブジェクトに対して `use()`、`unuse()` メソッドを繰り返し呼び出した場合は、常に初回に読み込んだオリジナルデータから内部データが作成されます。
- PNG 形式データで内部データ領域を再利用する場合、その内部データ領域をその他の形式のメディアデータで利用することはできません。

4.6 イメージ処理

i アプリ実行環境では、イメージ処理（イメージのエンコードや加工など）に関する機能が提供されています。この項では、それらの機能の概要について解説します。

4.6.1 イメージエンコーダ

イメージエンコーダは、`Image` や `Canvas` に描画された内容をキャプチャーして、特定の画像フォーマットのバイトイメージにエンコードする機能を提供します。この機能を利用することで、画像ファイルの内容を `Image` 上や `Canvas` 上で加工し、その結果を再度画像ファイルに変換して `ScratchPad` に保存したりサーバーに送信したりすることができます。

イメージのエンコードを行うには、`com.docomostar.ui.ImageEncoder` クラスを使用します。`static` メソッド `getEncoder()` の引数に画像フォーマット名を指定して呼び出すことで、その画像フォーマットに対応したエンコーダオブジェクトを取得することができます。実際のエンコード処理は、エンコーダオブジェクトの `encode()` メソッドを使用して行います。

`encode()` メソッドを呼び出してイメージのエンコードを行うと、その結果として `com.docomostar.media.EncodedImage` オブジェクトが返されます。`EncodedImage` はエンコードされた画像データをカプセル化したクラスであり、そのオブジェクトから画像のバイトイメージを取り出すことができます。

以下に、イメージのエンコード処理を行う例を示します。

例: イメージエンコーダの利用

```
class MainCanvas extends Canvas {

    public void paint(Graphics g) {
        // この Canvas に何かを描画する。
        ...
    }

    public void processEvent(int type, int param) {
        // ファンクションキーのイベントで、Canvas の描画内容をキャプチャーする。
        if (type==Display.KEY_RELEASED_EVENT && param==Display.KEY_SOFT1) {
            // 画像フォーマットは JPEG 形式とする。
            ImageEncoder ie = ImageEncoder.getEncoder("JPEG");
            // この Canvas の座標(0,0)から、幅 50 ドット、高さ 50 ドットの領域をキャプチャー
            // する。キャプチャーした結果は EncodedImage オブジェクトとして返される。
            EncodedImage ei = ie.encode(this, 0, 0, 50, 50);
            // 画像バイトイメージ（この場合 JPEG）を取り出すためのストリームを取得する。
        }
    }
}
```

```

InputStream is = ei.getInputStream();
...
    }
}
...
}

```

注意事項：

- イメージエンコーダでどの機種でも共通的にサポートされる画像フォーマットは JPEG 形式 (JFIF Baseline エンコーディング) です。JPEG 形式のイメージエンコーダを取得するには、`ImageEncoder.getEncoder()` メソッドの引数に "JPEG" を指定します。また、エンコード可能な画像のサイズは、縦横とも携帯電話の描画エリアのサイズまでです。このサイズを超えた画像をエンコードできるかどうかはメーカーにより異なりますので注意してください。
- **Canvas** をキャプチャーする場合、その **Canvas** オブジェクトはカレントフレームに設定されていなければなりません。また、その **Canvas** オブジェクトで IME が起動されている状態ではキャプチャーを行うことはできません。
- **Image** をキャプチャーする場合、その **Image** オブジェクトは `getGraphics()` メソッドを呼び出せるものでなければなりません。つまり、`Image.createImage()` メソッドにより作成された **Image** オブジェクトである必要があります。
- i アプリオプション API、i アプリ拡張 API に属する **Canvas**、**Image** 継承クラスではキャプチャーがサポートされない場合があります。

4.6.2 ピクセル操作

ピクセル操作とは、グラフィックスコンテキスト上に並んでいる各ピクセル（画素）に対してアプリケーションプログラムから直接アクセスを行うための機能です。アプリケーションプログラムは、ピクセル操作として **Graphics** クラスの以下の機能を使用することができます。

- グラフィックスコンテキスト上の、特定ピクセル位置の色情報取得 (`getPixel()`系メソッド)
- 特定ピクセル位置への点の描画 (`setPixel()`系メソッド)

注意事項：

- 色情報取得メソッド (`getPixel()`系メソッド) において、範囲外の座標やカレントフレームに設定されていない **Canvas** に対応付けられる **Graphics** オブジェクトを使用した場合は、黒を表す色情報が返されます。
- 色情報取得メソッドは、ダブルバッファリングが有効な状態ではバックバッファから情報を取得してアプリケーションプログラムに返します。このためダブルバッファリングが有効な状態では、ディスプレイで表示されている色とこれらのメソッドで取得する色が一致しているとは限りません。

4.6.3 イメージ回転・反転・拡大縮小表示

イメージの回転・反転表示を行うには、`Graphics.setFlipMode()` メソッドを呼び出します。**Graphics** オブジェクトに対して回転・反転方法を指定してこのメソッドを呼び出すと、以降のイメージ描画メソッド (`drawImage()` および `drawScaledImage()` メソッド) 呼び出しでは回転・反転されたイメージが描画されます。

指定可能な回転・反転方法には以下のものがあります。

- ・ `Graphics.FLIP_HORIZONTAL` : 横方向に鏡像反転することを示します。
- ・ `Graphics.FLIP_NONE` : 反転表示しないことを示します。
- ・ `Graphics.FLIP_ROTATE` : 縦横方向に鏡像反転 (180 度回転) することを示します。

- `Graphics.FLIP_ROTATE_LEFT` : 左に 90 度回転することを示します。
- `Graphics.FLIP_ROTATE_RIGHT` : 右に 90 度回転することを示します。
- `Graphics.FLIP_VERTICAL` : 縦方向に鏡像反転することを示します。
- `Graphics.FLIP_ROTATE_RIGHT_HORIZONTAL` : 右に 90 度回転し、横方向に鏡像反転することを示します。
- `Graphics.FLIP_ROTATE_RIGHT_VERTICAL` : 右に 90 度回転し、縦方向に鏡像反転することを示します。

また、イメージの拡大縮小表示を行うには、`Graphics.drawScaledImage()` メソッドを呼び出します。このメソッドの引数には、イメージとグラフィックスコンテキスト上の描画位置の他に以下を設定します。

- イメージ描画先グラフィックスコンテキスト上の矩形領域の幅と高さ
- 描画元イメージのうち、描画対象とする矩形領域の左上座標、および幅と高さ

これにより、描画元の矩形領域内の内容が、描画先の矩形領域にちょうど収まるように拡大または縮小されて表示されます。

なお、`Graphics` クラスには、画像を単純に拡大・縮小する `drawScaledImage()` メソッドの他に、イメージの回転や拡大縮小を任意のアフィン変換行列で表現し、指定することができる `drawImage()` メソッドが用意されています。このメソッドでは、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、 $m00 \sim m12$ 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。

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

4.6.4 イメージマップ

イメージマップ (`com.docomostar.ui.ImageMap` クラス) は、小さなイメージを縦横に並べて見かけ上大きなイメージを作成する機能を提供します。

イメージマップにイメージを割り当てる単位（イメージマップ上の升目）をセルと呼びます。イメージマップには、イメージマップを構成するイメージの集合（配列）と、個々のセルにどのイメージを割り当てるかを示すマップデータを指定します。以下は、6×6 個のセルから構成されるイメージマップにイメージを割り当てる様子を示しています。

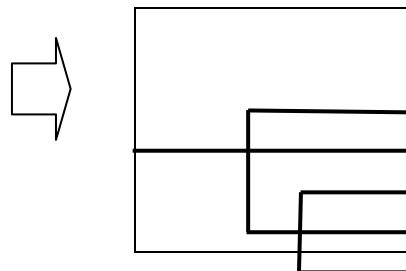
<セルへのイメージ割り当て>

	0		→		5
0	0	0	0	0	0
	0	0	4	1	1
	1	1	3	1	1
↓	0	0	2	4	1
	0	0	5	3	1
5	0	0	0	5	1

<イメージ配列>

0	
1	—
2	
3	+
4	⌈
5	⌋

<イメージマップの内容>



※升目内の数字がイメージ配列要素を示します。

このようにイメージマップは、数種類の小さな画像の組み合わせで表現される大きな画像を最小限のデータ量で取り扱いたい場合に適しています。

イメージマップは **Graphics.drawImageMap()** メソッドを使用してグラフィックスコンテキスト上に描画します。イメージマップはその全体を描画対象とすることもできますが、イメージマップ上にウィンドウと呼ぶ矩形領域を設定することにより、その領域だけを描画対象とすることもできます。ウィンドウの位置はイメージマップ全体のサイズの範囲内で自由に変更することができるため、ディスプレイ内で大きなイメージマップがスクロールするようなアプリケーションを容易に記述することができます。

なお、イメージマップでは、マップを構成するイメージを、個々の構成画像を連結した大きなイメージの形で与えることもできます。このようにすることで、データ量をより小さく抑えることができる場合があります。

10 枚の構成画像を個別に用意する例

0	1	2	3	4
5	6	7	8	9

10 枚の構成画像を 1 枚の画像にまとめて用意する例

0	1	2	3	4
5	6	7	8	9

4.6.5 スプライト

i アプリ実行環境では、イメージ処理機能の 1 つとして簡易なスプライト表示機能をサポートしています。スプライト機能は **com.docomostar.ui.Sprite** クラスと **com.docomostar.ui.SpriteSet** クラスから構成されます。

Sprite クラスは、画面上を移動する 1 つ 1 つのスプライトを表現し、属性としてイメージオブジェクトと表示位置を持ちます。また **Sprite** クラスの持つイメージを実際にディスプレイに表示する際には、以下の操作を行うことができます

- ・ 可視・不可視設定
- ・ 反転表示・回転表示設定

多くの場合、アプリケーションは 1 つの画面上にスプライトを複数表示します。このため、それら複数のスプライトを集合管理するための **SpriteSet** クラスを導入しています。アプリケーションは、同時に使用するスプライトを全てスプライトセットに登録します。画面へのスプライトの表示は、**Graphics.drawSpriteSet()** メソッドにスプライトセットを指定することで行います。**Sprite** オブジェクトをそのまま描画することはできません。

スプライトセットでは、内部で保持しているスプライト間の衝突判定を行うこともできます。衝突判定は、各々のスプライトの位置およびスプライトに登録されているイメージの矩形のサイズに基づいて行われます。

4.6.6 パレット付きイメージ

パレット付きイメージ (**com.docomostar.ui.PalettedImage** クラスと **com.docomostar.ui.Palette** クラス) とは、アプリケーションプログラムから色パレットを指定することのできるイメージです。

パレット付きイメージは static メソッド **PalettedImage.createPalettedImage()** を、パレット付きイメージとして取り扱いたい GIF データのバイト配列を引数に指定して呼び出すことにより取得します (GIF データ以外に 1、4、8 ビットの Windows BMP 画像を使用することもできます)。また、**createPalettedImage()** メソッドに縦横サイズのみ指定し、空のイメージを生成することもできます。一度生成した **PalettedImage** オブジェクトに対しては、**changeData()** メソッドを使用して画像データの内容を差し替えることが可能です。

パレット付きイメージからは、`getPalette()`メソッドおよび`setPalette()`メソッドを使用してパレットの参照や設定を行うことができます。パレットの色情報には、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。

`PalettedImage` クラスは `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 `PalettedImage` 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。

なお、`PalettedImage` クラスでも透過色指定を行うことができますが、`PalettedImage` クラスでの透過色指定は、親クラスである `Image` クラスの透過色指定（次項参照）と異なり、`setTransparentIndex()`メソッドにパレットのインデックス値を指定することにより行うことに注意してください。

4.6.7 イメージに対する透過色指定と半透明描画指定

`Image` クラスでは、イメージに対する透過色指定、および半透明描画指定をサポートしています。以下にこれらの機能について解説します。

(1) イメージに対する透過色指定

イメージの描画時に、アプリケーションプログラムが指定した特定の色を透過色として取り扱い、イメージ上その色が配色されている部分を描画対象外とする機能です。

イメージに透過色を指定して描画するには、`Image` オブジェクトに対して `setTransparentColor()`メソッドを呼び出すことで透過させたい色を指定した後に、`setTransparentEnabled()`メソッドに引数 `true` を指定して呼び出すことでその透過色を有効化します。`setTransparentColor()`メソッドの引数に指定する色情報には、`Graphics.getColorOfName()`メソッドや `Graphics.getColorOfRGB()`メソッドで返される、メーカー固有の色値を使用します。

`setTransparentEnabled()`メソッドは、`setTransparentColor()`メソッドを使用して透過色を変更するたびに（透過色を変更した後に）呼び出す必要があることに注意してください。

現在透過色として指定されている色は、`getTransparentColor()`メソッドで確認することができます。また、`setTransparentEnabled()`メソッドに引数 `false` を指定して呼び出すことで、有効となっている透過色指定を無効化することができます。

注意事項：

- `setTransparentColor()`メソッドに渡す色値にアルファ値の情報が含まれている場合であってもその情報は無視され、常にアルファ値として 255 が指定されているものとして取り扱われます。
- 透過 GIF から作成された `Image` オブジェクトでは、`setTransparentEnabled()`メソッドに引数 `true` を指定して呼び出すまでは GIF データから引き継いだ透過色が有効です。この GIF データから引き継いだ透過色情報は、`setTransparentEnabled()`に引数 `true` を指定して呼び出すと失われます。
- 透過 PNG から生成された `Image` オブジェクトでは、`setTransparentEnabled()`メソッドに引数 `true` を指定して呼び出すまでは PNG データから引き継いだ透明色が有効です。この PNG データから引き継いだ透過色情報は、`setTransparentEnabled()`に引数 `true` を指定して呼び出すと失われます。
なお、PNG のカラータイプによっては、`setTransparentEnabled()`メソッドを呼び出すと例外が発生します。詳細は API リファレンスの `Image` クラス説明を参照してください。

(2) イメージに対する半透明描画指定

イメージの描画において、そのイメージに作用させるアルファ値を指定する機能です。本機能を使用することで、イメージの描画時に描画元（イメージ）と描画先の各ピクセルについての透過演算を行わせることができます。

半透明描画を行うには、`Image` オブジェクトに対して `setAlpha()`メソッドを使用して、アルファ値の設定を行います。このメソッドの引数には 0～255 の範囲の値が設定できます。255 を指定した場合にはそのイメージを完全に不透明なものとして扱い、0 を指定した場合には完全に透明なものとして扱います。

第 5 章

通信の制御

i アプリ実行環境では、ネットワーキング API が提供されます。これは、Web サーバー側コンポーネントと通信するときの開発者が使用する一連のクラスです。これらのクラスの目的は、通信クライアントとしての i アプリの開発をできるだけ簡単にすることにあります（第 2 章の図 5「Generic Connection クラス階層」を参照）。i アプリ実行環境では、アプリケーションプログラムは HTTP(S) プロトコルを使用して Web サーバーと通信を行うことができます。Java 言語標準ライブラリには、記憶装置との入出力や通信を処理する機能が豊富に揃っています。i アプリ実行環境では、2.2.2 項で述べた Generic Connection フレームワークに基づいて携帯電話と Web サーバーを接続する基本的な機能が提供されます。以下の各節では、HTTP(S) 接続インタフェースを使ってサーバーと通信する手順について説明します。

i アプリ実行環境のネットワーキング API と Java 言語標準ライブラリの `java.net.HttpURLConnection` は、次の点で大きく異なります。

- メッセージコンテンツのタイプを認識し、コンテンツを分析してオブジェクト形式に変換するハンドラをプラグインするためのコンテンツハンドラファクトリという概念はありません。
- メモリサイズを削減するため、サポートするコンテンツタイプを絞り込んでいます。一般的なアプリケーションでは、使用するコンテンツタイプは限られているため、多様なコンテンツタイプをサポートする総合的なフレームワークは採用していません。
- 汎用的な `URLConnection` (`java.net.URLConnection` クラス) はありません。クラス数を最小限に抑えるため、汎用的な `URLConnection` API は `HttpConnection` インタフェースに統合されています。これによってメモリサイズが減少し、開発者にとっても使用するべき API が簡素になるというメリットがあります。
- `URLStreamHandlerFactory` や `URLStreamHandler` API はありません。i アプリ実行環境では、使用できるネットワークプロトコルは HTTP、HTTPS に限定されており、多様な通信プロトコルに対応するための総合的なフレームワークは採用していません。これによってメモリサイズが減少し、セキュリティの問題も取り除かれます。
- i アプリ API の `HttpConnection` を使用する上では、Generic Connection フレームワーク API の厳密な呼び出しシーケンスが定義されています。詳細は、5.1 項を参照してください。

注意事項：

- HTTP(S) プロトコルの利用にあたり、i アプリ実行環境では HEAD メソッドはサポートされず、GET、POST の各メソッドのみ使用可能です。
- HTTP プロトコルのバージョンは 1.1 となります。
- 1 つの i アプリから同時に確立可能な HTTP の通信リンクは 1 つだけです。また、i アプリのバックグラウンドで動作しているネイティブ機能を含め、HTTP 通信が行われている状況では、i アプリから HTTP 通信機能を利用することはできません。

5.1 クライアント/サーバープログラミング

多くのプログラムは、クライアントとサーバーが対になって動作します。i アプリサービスでは、サーバーはインターネット経由でアクセスするリモートホストシステムであり、クライアントは携帯電話上で動作する i アプリ自身です。クライアントとサーバーは、通常、IP アドレス（サーバー固有の識別子）とポート番号（接続を特定のプロセスにルーティングするための識別子）を知ることによって相互に通信します。構造をシンプルにするため、ネットワーク接続には通常の入出力ストリームが使用されます。データの読み書きは、一般的な Java 言語標準ライブラリのストリームクラスおよびメソッドを使って行います。そのため、通信対象への接続から入力ストリームを取得するメソッドと出力ストリームを取得するメソッドがそれぞれ用意されています。これによって、クライアントとサーバーは相互に通信することができます。クライアントは、HTTP サーバーから Web ページの内容を入手するなどのインターネットサービスが必要になると、URL を使ってこのサービスにアクセスします。i アプリ実行環境では、Java 言語標準ライブラリにおける `HttpURLConnection` のサブセット機能をサポートする `HttpConnection` インタフェースが提供されます。次のコードは、`HttpConnection` インタフェース（の実装オブジェクト）の設定と使用の例を示したものです。

例: 接続の確立と動作

```
private String[] getScoreList(int score) {
    try {
        // HTTP 接続 (HttpConnection) オブジェクトを取得する。
        HttpConnection conn = (HttpConnection)Connector.open(
            "http://backflip.sfbay:8080/scores", Connector.READ_WRITE,
            true);

        // リクエストメソッドとコンテンツタイプを設定する。
        conn.setRequestMethod(HttpConnection.POST);
        conn.setRequestProperty("Content-Type", "text/plain");

        // 出力ストリームを取り出す。
        OutputStream out = conn.getOutputStream();

        // データを書き込む（実際の接続がまだ確立されていないため、
        // 書き込まれたデータは一時的にバッファに入れられる）。
        out.write("Highscore-Game: Space Invaders\n".getBytes());
        out.write(("Highscore-Name: " + SpaceInvaders.USER.getName() +
            "\n").getBytes());
        out.write(("Highscore-Value: " + score + "\n").getBytes());

        // 出力ストリームをクローズする。
        out.close();

        // リモート資源に実際に接続する。
        conn.connect();

        String[] result = new String[HighScoreScreen.MAX_ENTRIES];

        // 入力ストリームを取り出す。
        InputStream in = conn.getInputStream();
        if (in == null)
            throw new IOException("high score servlet unreachable");

        // データを読み取る。
        for (int i=0; i<HighScoreScreen.MAX_ENTRIES; i++) {
            result[i] = myReadLineMethod(in);
        }

        // 入力ストリームをクローズする。
        in.close();

        // HTTP 接続をクローズする。
        conn.close();
    }
}
```

```

        return result;
    } catch (Exception x) {
        return new String[] {"Scores", "currently", "unavailable."};
    }
}

```

HttpConnection は、HTTP や HTTPS プロトコルのネットワーク資源にアクセスするために使用されるインタフェースです。このインタフェースを実装したクラスのインスタンスは、URL で指定する資源の読み取りと書き込みの両方に使用できます。一般に、URL との接続は、次の手順に従って行います。

1. **Connector.open()** メソッドを呼び出して接続オブジェクトを作成する。オープンモードとリクエストメソッドは以下のように対応する。

Connector.READ	: GET メソッドに対応
Connector.READ_WRITE	: POST メソッドに対応

2. セットアップパラメータを設定する（リクエストメソッドとリクエストヘッダ）。アプリケーションプログラムから設定可能なリクエストヘッダは、**If-Modified-Since** と **Content-Type** の2つです。
3. **HttpConnection.openOutputStream()** メソッドを呼び出して、この接続の出力ストリームを取得する。
4. **OutputStream.write()** メソッドを使って出力ストリームに書き込む。
5. **OutputStream.close()** メソッドを使って出力ストリームをクローズする。
6. **HttpConnection.connect()** メソッドを使ってリモートオブジェクトに実際に接続する。この後、必要に応じてレスポンスヘッダを参照することができます。アプリケーションプログラムから参照可能なレスポンスヘッダについては、API リファレンス (**HttpConnection.getHeaderField()** メソッド) を参照してください。
7. **HttpConnection.openInputStream()** メソッドを使って入力ストリームを取得する。
8. **InputStream.read()** メソッドを使って入力ストリームから読み取る。
9. **InputStream.close()** メソッドを使って入力ストリームをクローズする。
10. **HttpConnection.close()** メソッドを使って接続をクローズする。

上記の3～5は、POST メソッド使用時における HTTP リクエストボディの書き込み処理に相当します。また7～9は、POST、GET 両メソッドにおける HTTP レスポンスボディの読み込み処理に相当します。

これらの処理はそれぞれ必須ではなく必要に応じて行うものですが、処理を行う場合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 **HttpConnection.connect()** を呼び出した後で **HttpConnection.openOutputStream()** を呼び出すと、機種によっては例外が発生することがあります。

次の図に、接続の始めから終わりまでの状態遷移を示します。

URLConnection の状態遷移図

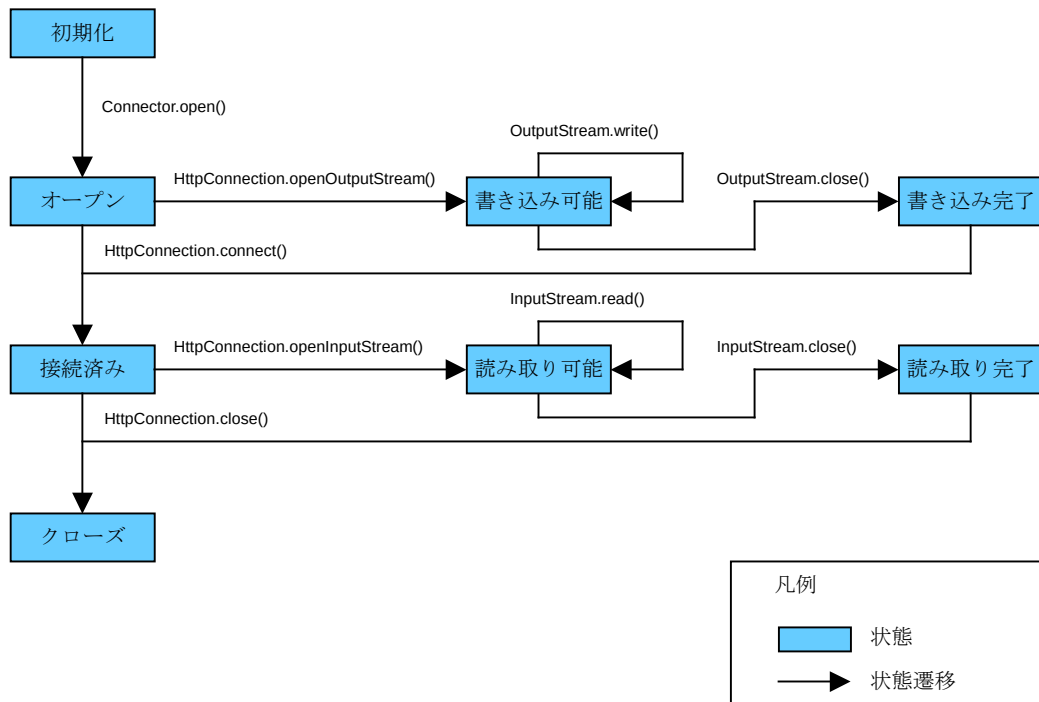


図 10 HttpURLConnection の状態遷移

注意事項：

- HttpURLConnection に渡す URL には、次の制約があります。
 - IP アドレスはシンボリックでなければなりません ("www.nttdocomo.co.jp" など)。HttpURLConnection では、数値 IP アドレス ("127.0.0.1" など) は使用できません。
 - HTTP(S) 通信で指定する URL の長さとして、スキーム/パス部に 255 バイト、クエリー部に 512 バイトまで指定することができます。
 - HttpURLConnection に渡す URL のプロトコル、ホスト名、ポートは、i アプリのダウンロード元となった URL (ADF の PackageURL キーでの指定) と同じでなければなりません。
- i アプリサービスにおけるクライアント/サーバーシステムでは、クライアント (携帯電話) とサーバー (Web サーバー) の間に i モードサーバーが介在しています。i モードサーバーでは、以下のような異常検出時にレスポンスステータスを 200 としてメッセージを返すことがあります。このような場合、クライアント側ではレスポンスステータスに基づいた正常性確認を行うことができません。クライアント側で厳密な正常性確認を行うには、独自の HTTP レスポンスヘッダの付加や HTTP レスポンスボディの内容による判断を行うようにしてください。
 - i モードサーバーのインターネット向けプロキシサーバー輻輳時
 - 公式メニューサイトのメンテナンス時 (公式メニューサイトのコンテンツプロバイダより申請があった場合)

なお、異常を示すレスポンスステータスが返却された場合、i アプリ実行環境は例外をスローします。

- HTTP リクエスト/レスポンスボディについて x-www-form-urlencoded 形式のエンコード、デコードを行うには、それぞれ以下のユーティリティクラスを使用してください。
 - com.docomostar.net.URLEncoder
 - com.docomostar.net.URLDecoder

なお Star-1.2 プロファイルにおいて、上記それぞれのクラスのエンコード、デコードメソッドに、引数に文字エンコーディング種別を指定できるバリエーションが追加されました。

- HTTP(S)通信で送信可能なリクエストボディのサイズ、および受信可能なレスポンスボディのサイズには、プロファイルの世代により以下の制限があります。

プロファイル	FOMA 携帯電話	
	リクエスト	レスポンス
Star-1.x	80K バイト	150K バイト
Star-2.x	80K バイト	1024K バイト (*1)

(*1) ミニアプリからの HTTP(S)通信で受信可能なレスポンスボディのサイズは 150K バイトです。

この制限は、アプリケーションプログラムが直接 **HttpConnection** を操作する場合だけでなく、メディアデータを HTTP 通信を介して取得する際にも適用されます。

- i アプリがアクセスする URL には、サーバー側で基本認証 (BASIC 認証) の設定が行われている場合があります。このような URL にアクセスした場合、携帯電話は自動的にユーザーID とパスワードの入力を促すダイアログを表示します。アプリケーションプログラムが独自の処理で基本認証の情報 (Authorization リクエストヘッダなど) をサーバーに送信することはできません。
- i アプリが通信を行っている最中に通話着信があった場合、その動作は携帯電話の i モード着信設定に従います。
- Web サーバーに POST メソッドで HTTP リクエストを送信する場合、HTTP リクエストボディの設定内容に適したコンテンツタイプを指定するようにしてください。コンテンツタイプ (Content-Type ヘッダ) の設定には **HttpConnection.setRequestProperty()** メソッドを使用します。また、リクエストボディに書き込むデータが存在しない場合、POST メソッドではなく GET メソッドを使用します。GET メソッド使用時はコンテンツタイプの設定は行わないようにしてください。
- i モードサービスでは、Web サーバーからのレスポンスにはコンテンツ長 (Content-Length ヘッダ) の設定を必須としています。i アプリからの HTTP(S)通信においても、Web サーバーからのレスポンスにはコンテンツ長の設定を行うようにしてください。
- i アプリが通信を行っている状態で i アプリの中断 (3.6 項参照) が発生すると、通信は中断されます。この場合、i アプリの実行が再開された時点で例外が発生します。
- **HttpConnection** インタフェースには、HTTP レスポンスに含まれるレスポンスヘッダの内容を参照するための **getHeaderField()** メソッドがあります。このメソッドでは "x-" で始まるユーザー定義ヘッダも取得することができますが、i モードにおいては i モードサーバーが携帯電話との通信を制御するために独自のユーザー定義ヘッダを使用する場合があるため、i アプリではユーザー定義ヘッダは使わないことを強く推奨します。i モードサーバーが使用しているものと同名のヘッダが i アプリの HTTP 通信で使用された場合、i モードサーバーはそのヘッダを削除する場合があります。
- 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、i アプリの起動を行うことはできますが起動された i アプリは通信を行うことはできません。

【Star-1.2】

通信時やその入出力処理で何らかの異常が発生した場合、**ConnectionException** の例外が発生する場合があります。この例外クラスにはその発生原因を表すステータス値が用意されていますが、Star-1.2 プロファイル以降、i モードロック中を示すステータス値である **IMODE_LOCKED** は非推奨となりました。このステータス値が実際にシステムから使用されることはありません。

5.1.1 接続の終了

サーバーとクライアント間の接続は、**HttpConnection** インタフェースの **close()** メソッドを使用してクローズします。ただし、このメソッドが実際の物理的な資源を解放するのは、**HttpConnection** から取得されたすべての入出力ストリームがクローズされた後になります。

なお、`connect()`メソッドが通信を行っている状態で、他のスレッドからその `HttpConnection` インスタンスに対し `close()`メソッドを呼び出すことで通信を切断させることができます。

注意事項：

- `Connector.open()`を呼び出して取り出すすべての接続（`Connection` インタフェースの実装オブジェクト）と、そこから取得されたすべての入出力ストリームは明示的にクローズする必要があります。これは、`Connector.open()`メソッドで取得した `Connection` 実装オブジェクトを使用した結果として、通信エラーなどにより例外が発生した場合でも同様です。クローズしない場合、資源のリークが発生して新しい接続をオープンすることができなくなる場合がありますので注意してください。

5.2 セッション管理

HTTP プロトコルでは、永続的な接続は維持されません。接続は、クライアントがサーバーとデータを交換する必要があるときだけ確立されます。一方、特定のユーザーにのみ使用を許可するサーバーではユーザーの認証が必要になりますが、接続が行われるたびにユーザーがサーバーにログインすることは現実的ではありません。クライアントが繰り返しサーバーとデータを交換するためには、ユーザーが継続してシステムにログインした状態を保つことが必要となります。

この問題を解決するための手段がセッション管理と呼ばれている手法です。セッション管理の手法では、ユーザーがシステムにログインした際にセッションを一意的に識別するための文字列（セッション ID）を作成します。そしてそれ以降は、サーバーとクライアントの間でセッション ID をやり取りして、個々の HTTP 接続の持ち主（ログインユーザー）を識別します。セキュリティを保つために、一定時間アクセスのなかったセッション ID は無効化されます。無効化されたセッション ID の持ち主は、再度システムを利用する際にログインする必要があります。

一般的には、セッション管理の具体的な実装例としてクッキーが広く利用されています。クッキーは HTTP リクエスト/レスポンスヘッダの特定のヘッダにユーザー情報（この場合セッション ID）を設定してクライアントとサーバーの間でセッションを保ちます。しかし i アプリ実行環境では HTTP リクエストヘッダに設定できる情報は **Content-Type** と **If-Modified-Since** の 2 つに制限されており、クッキーを使用することはできません。i アプリ実行環境ではクッキーを使用する代わりに、リクエストの対象 URL か HTTP のボディ部分にセッション ID を設定するようにしてください。

次の例では、クッキーを使用する代わりに HTTP リクエスト/レスポンスボディを使用してセッション ID のやり取りを行っています。

例: セッション管理の実装例

```
/**
 * このクラスにはバンクサーバーとの通信処理が実装されている。
 */
public class BankingClient {

    private static final String UNAUTHORIZED =
        "verification of account data failed";
    private String url;
    private String cookie;
    private HttpConnection conn;
    private InputStream in;

    /**
     * このクラスをインスタンス化する。その際、入力された ID とパスワードをチェックする
     * ためにサーバーに初期接続を行う。
     */
}
```

```

* @param url バンクサーバーへの URL
* @param acct ユーザーのアカウント名
* @param pin ユーザーのアカウントパスワード
* @exception サーバーに接続できないか入力された ID とパスワードが拒否された場合
* に IOException がスローされる。
*/
public BankingClient(String url, String acct, String pin)
    throws IOException {
    this.url = url;
    conn = (HttpURLConnection)Connector.open(url,
        Connector.READ_WRITE, true);
    conn.setRequestMethod(HttpURLConnection.POST);
    conn.setRequestProperty("Content-Type", "text/plain");
    OutputStream out = conn.getOutputStream();

    // HTTP リクエストボディにアカウント情報を設定する。
    out.write((VERIFY + " " + acct + " " + pin).getBytes());
    out.close();

    conn.connect();

    in = conn.openInputStream();
    if (getCookie(myReadLineMethod(in)) == null) {
        throw new IllegalArgumentException(UNAUTHORIZED);
    }
}

private String getCookie(String line) {
    String header = "Cookie: ";
    // HTTP レスポンスボディから読み込んだ行よりセッション ID を取得する。
    if (!line.startsWith(header) || line.length() <= header.length())
        return null;
    return line.substring(header.length());
}

private String contactServer(String cmd) {
    try {
        // 古い接続が残っていたらそれをクローズする。
        if (in != null) in.close();
        if (conn != null) conn.close();

        // サーバーとの接続を再度オープンする。その際に、この
        // インスタンスのコンストラクタによる初期接続で得られた
        // セッション ID を HTTP リクエストボディに設定する。
        conn = (HttpURLConnection)Connector.open(url,
            Connector.READ_WRITE, true);
        conn.setRequestMethod(HttpURLConnection.POST);
        conn.setRequestProperty("Content-Type", "text/plain");
        OutputStream out = conn.getOutputStream();
        out.write((cmd + "¥n").getBytes());
        out.write(("Cookie: " + cookie + "¥n").getBytes());
        out.close();

        conn.connect();

        in = conn.openInputStream();
        if (in == null) {
            throw new IOException(UNAUTHORIZED);
        }

        return myReadLineMethod(in);

        // 以降の処理でこの接続からさらにデータの読み込みが行われるので、
        // ここでは入力ストリームと接続のクローズは行わない。
        // 接続のクローズは、このメソッドが次に呼び出されるときか
        // このクラスの close() メソッドを呼び出すことによって行われる。
    } catch (IOException ioe) {
        return null;
    }
}
}

```

}

5.3 HTTPS による安全な通信

安全な接続が必要なクライアント/サーバーアプリケーションでは、HTTPS プロトコルを使用します。HTTPS サーバーに接続するクライアントプログラムのコードは、URL 文字列のプロトコルセクションに“http”の代りに“https”と指定する以外、HTTP プロトコルの場合と全く同じです。たとえば、クライアントとサーバーの間でスペースインベータのハイスコアと名前を安全に送受信する場合、必要な変更は URL 文字列だけです。つまり、サーバー

(backflip.sfbay) のポート 8080 番において HTTP プロトコルを使用したサービスが提供されており、またポート 7070 番で HTTPS プロトコルを使用した同様のサービスが提供されている場合、URL “http://backflip.sfbay:8080/”

を “https://backflip.sfbay:7070/” で置き換えます。

注意事項：

- **URLConnection** に渡される URL は i アプリがダウンロードされたときの URL と同じプロトコル、ホスト、ポート番号が指定されていなければなりません。したがって、HTTPS を介してダウンロードされた i アプリでなければ HTTPS プロトコルを使用することはできません。これは、HTTPS を介してダウンロードされた i アプリは、HTTP プロトコルは使用できないということでもあります。
- HTTPS 通信で利用する Web サーバーには、i アプリ対応携帯電話が対応している認証局によって署名されたサーバー証明書をインストールしておく必要があります。i アプリ対応携帯電話が対応する認証局のルート証明書については、以下の URL を参照してください。
<http://www.nttdocomo.co.jp/service/developer/make/content/ssl/spec/index.html>

第 6 章 テキスト処理

一般にアプリケーションのテキスト処理では、ASCII 文字セット以外に各国固有の文字セットも多く使用されます。i モードサービスでは日本語テキストの符号化方式として Shift-JIS を採用していますが、i アプリ実行環境では国際化されたストリーム入出力機構を利用して、Shift-JIS の日本語テキストを容易に取り扱うことができます。

6.1 テキスト処理

仮想マシンでは、すべてのテキスト処理は Unicode で行われます。このため、Unicode 以外の形式で符号化されたテキストデータを文字ストリームとして使用するには、これを Unicode に変換する必要があります。i アプリ実行環境には国際化されたストリーム入出力機構が備えられており、Unicode 文字ストリームとそれ以外のテキストのバイトストリームとの間の文字コード変換を行うクラスが `java.io` パッケージで提供されています。`InputStreamReader` クラスはバイトストリームを文字ストリームに、`OutputStreamWriter` クラスは文字ストリームをバイトストリームにそれぞれ変換します。次に、この変換プロセスを示します。

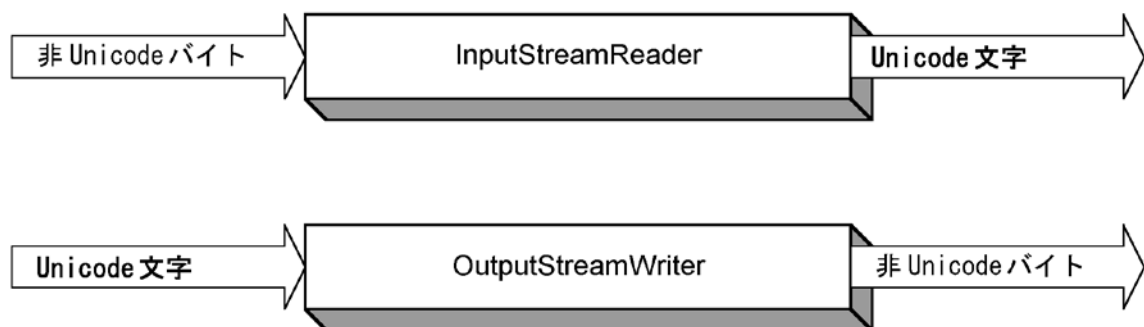


図 11 入出力ストリームの変換

アプリケーションプログラムでは、`InputStreamReader` や `OutputStreamWriter` クラスを使ってネットワークから入出力するテキストを適切に変換する必要があります。なお、i モードサービスでは日本語テキストの符号化方式として Shift-JIS (SJIS) を採用しており、これらのクラスのデフォルトエンコーディングも SJIS となっています。

次に、これらの変換方法を使用するコード例を示します。

例: SJIS バイトデータを Unicode 文字に変換

```
HttpConnection conn = (HttpConnection)Connector.open(url, Connector.READ, true);
conn.setRequestMethod(HttpConnection.GET);
conn.connect();
```

```
// SJIS バイトデータを Unicode 文字に変換する。(HTTP レスポンスボディ)
InputStreamReader isr = new InputStreamReader (conn.openInputStream()); //デフォルト
の符号化は SJIS
...
```

例: Unicode 文字を SJIS バイトデータに変換

```
HttpConnection conn = (HttpConnection)Connector.open(url, Connector.READ_WRITE,
true);
conn.setRequestMethod(HttpConnection.POST);

// Unicode 文字を SJIS バイトデータに変換する。(HTTP リクエストボディ)
InputStreamReader osw = new OutputStreamWriter (conn.openOutputStream()); //デフォ
ルトの符号化は SJIS
...
conn.connect();
```

com.docomostar.io パッケージの、以下の 2 つのクラスは、Java 言語標準ライブラリの java.io パッケージに含まれる同名のクラスのサブセットに相当する機能を提供します。

- ・ バッファ付きリーダ (BufferedReader)
- ・ フォーマット出力用ライタ (PrintWriter)

6.2 i アプリ対応携帯電話の絵文字

i アプリ対応携帯電話では、従来の i モードコンテンツでもサポートされている絵文字 (i モード用外字セット) を利用することができます。以下に NTT ドコモが定義した、各携帯電話に搭載されている絵文字のコード表 (SJIS および Unicode の対応表) を示します。各コードの値は 16 進数の数値です。

なお、ここに記載したフォントは、実際の携帯電話に搭載されたフォントとは若干異なりますのでご注意ください。一般的に、フォントを構成するドット数の制約により、実際の携帯電話に搭載されるフォントはここに記載されたものより粗く表示されます。

< 基本絵文字 >

文字	SJIS	Unicode	タイトル	文字	SJIS	Unicode	タイトル
☐	F89F	E63E	晴れ	☐	F8CA	E669	ホテル
☐	F8A0	E63F	曇り	☐	F8CB	E66A	コンビニ
☐	F8A1	E640	雨	☐	F8CC	E66B	ガソリンスタンド
☐	F8A2	E641	雪	☐	F8CD	E66C	駐車場
☐	F8A3	E642	雷	☐	F8CE	E66D	信号
☐	F8A4	E643	台風	☐	F8CF	E66E	トイレ
☐	F8A5	E644	霧	☐	F8D0	E66F	レストラン
☐	F8A6	E645	小雨	☐	F8D1	E670	喫茶店
☐	F8A7	E646	牡羊座	☐	F8D2	E671	バー
☐	F8A8	E647	牡牛座	☐	F8D3	E672	ビール
☐	F8A9	E648	双子座	☐	F8D4	E673	ファーストフード
☐	F8AA	E649	蟹座	☐	F8D5	E674	ブティック
☐	F8AB	E64A	獅子座	☐	F8D6	E675	美容院
☐	F8AC	E64B	乙女座	☐	F8D7	E676	カラオケ
☐	F8AD	E64C	天秤座	☐	F8D8	E677	映画
☐	F8AE	E64D	蠍座	☐	F8D9	E678	右斜め上
☐	F8AF	E64E	射手座	☐	F8DA	E679	遊園地
☐	F8B0	E64F	山羊座	☐	F8DB	E67A	音楽
☐	F8B1	E650	水瓶座	☐	F8DC	E67B	アート
☐	F8B2	E651	魚座	☐	F8DD	E67C	演劇
☐	F8B3	E652	スポーツ	☐	F8DE	E67D	イベント
☐	F8B4	E653	野球	☐	F8DF	E67E	チケット
☐	F8B5	E654	ゴルフ	☐	F8E0	E67F	喫煙
☐	F8B6	E655	テニス	☐	F8E1	E680	禁煙
☐	F8B7	E656	サッカー	☐	F8E2	E681	カメラ
☐	F8B8	E657	スキー	☐	F8E3	E682	カバン
☐	F8B9	E658	バスケットボール	☐	F8E4	E683	本
☐	F8BA	E659	モータースポーツ	☐	F8E5	E684	リボン
☐	F8BB	E65A	ポケットベル	☐	F8E6	E685	プレゼント
☐	F8BC	E65B	電車	☐	F8E7	E686	バースデー
☐	F8BD	E65C	地下鉄	☐	F8E8	E687	電話
☐	F8BE	E65D	新幹線	☐	F8E9	E688	携帯電話
☐	F8BF	E65E	車（セダン）	☐	F8EA	E689	メモ
☐	F8C0	E65F	車（RV）	☐	F8EB	E68A	TV
☐	F8C1	E660	バス	☐	F8EC	E68B	ゲーム
☐	F8C2	E661	船	☐	F8ED	E68C	CD
☐	F8C3	E662	飛行機	☐	F8EE	E68D	ハート
☐	F8C4	E663	家	☐	F8EF	E68E	スペード
☐	F8C5	E664	ビル	☐	F8F0	E68F	ダイヤ
☐	F8C6	E665	郵便局	☐	F8F1	E690	クラブ
☐	F8C7	E666	病院	☐	F8F2	E691	目
☐	F8C8	E667	銀行	☐	F8F3	E692	耳
☐	F8C9	E668	ATM	☐	F8F4	E693	手（グー）

文字	SJIS	Unicode	タイトル	文字	SJIS	Unicode	タイトル
□	F8F5	E694	手 (チョキ)	□	F984	D6DF	フリーダイヤル
□	F8F6	E695	手 (パー)	□	F985	E6E0	シャープダイヤル
□	F8F7	E696	右斜め下	□	F986	E6E1	モバQ
□	F8F8	E697	左斜め上	□	F987	E6E2	1
□	F8F9	E698	足	□	F988	E6E3	2
□	F8FA	E699	くつ	□	F989	E6E4	3
□	F8FB	E69A	眼鏡	□	F98A	E6E5	4
□	F8FC	E69B	車椅子	□	F98B	E6E6	5
□	F940	E69C	新月	□	F98C	E6E7	6
□	F941	E69D	やや欠け月	□	F98D	E6E8	7
□	F942	E69E	半月	□	F98E	E6E9	8
□	F943	E69F	三日月	□	F98F	E6EA	9
□	F944	E6A0	満月	□	F990	E6EB	0
□	F945	E6A1	犬	□	F991	E6EC	黒ハート
□	F946	E6A2	猫	□	F992	E6ED	揺れるハート
□	F947	E6A3	リゾート	□	F993	E6EE	失恋
□	F948	E6A4	クリスマス	□	F994	E6EF	ハートたち (複数ハート)
□	F949	E6A5	左斜め下	□	F995	E6F0	わーい (嬉しい顔)
□	F950	E6AC	カチンコ	□	F996	E6F1	ちっ (怒った顔)
□	F951	E6AD	ふくろ	□	F997	E6F2	がく〜 (落胆した顔)
□	F952	E6AE	ペン	□	F998	E6F3	もうやだ〜 (悲しい顔)
□	F955	E6B1	人影	□	F999	E6F4	ふらふら
□	F956	E6B2	いす	□	F99A	E6F5	グッド (上向き矢印)
□	F957	E6B3	夜	□	F99B	E6F6	るんるん
□	F95B	E6B7	soon	□	F99C	E6F7	いい気分 (温泉)
□	F95C	E6B8	on	□	F99D	E6F8	かわいい
□	F95D	E6B9	end	□	F99E	E6F9	キスマーク
□	F95E	E6BA	時計	□	F99F	E6FA	ぴかぴか (新しい)
□	F972	E6CE	phone to	□	F9A0	E6FB	ひらめき
□	F973	E6CF	mail to	□	F9A1	E6FC	むかつ (怒り)
□	F974	E6D0	fax to	□	F9A2	E6FD	パンチ
□	F975	E6D1	i モード	□	F9A3	E6FE	爆弾
□	F976	E6D2	i モード (枠付き)	□	F9A4	E6FF	ムード
□	F977	E6D3	メール	□	F9A5	E700	バッド (下向き矢印)
□	F978	E6D4	ドコモ提供	□	F9A6	E701	眠い (睡眠)
□	F979	E6D5	ドコモポイント	□	F9A7	E702	exclamation
□	F97A	E6D6	有料	□	F9A8	E703	exclamation&question
□	F97B	E6D7	無料	□	F9A9	E704	exclamation×2
□	F97C	E6D8	ID	□	F9AA	E705	どんっ (衝撃)
□	F97D	E6D9	パスワード	□	F9AB	E706	あせあせ (飛び散る汗)
□	F97E	E6DA	次項有	□	F9AC	E707	たらーっ (汗)
□	F980	E6DB	クリア	□	F9AD	E708	ダッシュ (走り出すさま)
□	F981	E6DC	サーチ (調べる)	□	F9AE	E709	ー (長音記号 1)
□	F982	E6DD	NEW	□	F9AF	E70A	ー (長音記号 2)
□	F983	D6DE	位置情報	□	F9B0	E70B	決定

＜拡張絵文字＞

文字	SJIS	Unicode	タイトル	文字	SJIS	Unicode	タイトル
□	F9B1	E70C	i アプリ	□	F9D7	E732	トレードマーク
□	F9B2	E70D	i アプリ (枠付き)	□	F9D8	E733	走る人
□	F9B3	E70E	T シャツ (ボーダー)	□	F9D9	E734	マル秘
□	F9B4	E70F	がま口財布	□	F9DA	E735	リサイクル
□	F9B5	E710	化粧	□	F9DB	E736	レジスタードトレードマーク
□	F9B6	E711	ジーンズ	□	F9DC	E737	危険・警告
□	F9B7	E712	スノボ	□	F9DD	E738	禁止
□	F9B8	E713	チャペル	□	F9DE	E739	空室・空席・空車
□	F9B9	E714	ドア	□	F9DF	E73A	合格マーク
□	F9BA	E715	ドル袋	□	F9E0	E73B	満室・満席・満車
□	F9BB	E716	パソコン	□	F9E1	E73C	矢印左右
□	F9BC	E717	ラブレター	□	F9E2	E73D	矢印上下
□	F9BD	E718	レンチ	□	F9E3	E73E	学校
□	F9BE	E719	鉛筆	□	F9E4	E73F	波
□	F9BF	E71A	王冠	□	F9E5	E740	富士山
□	F9C0	E71B	指輪	□	F9E6	E741	クローバー
□	F9C1	E71C	砂時計	□	F9E7	E742	さくらんぼ
□	F9C2	E71D	自転車	□	F9E8	E743	チューリップ
□	F9C3	E71E	湯のみ	□	F9E9	E744	バナナ
□	F9C4	E71F	腕時計	□	F9EA	E745	りんご
□	F9C5	E720	考えてる顔	□	F9EB	E746	芽
□	F9C6	E721	ほっとした顔	□	F9EC	E747	もみじ
□	F9C7	E722	冷や汗	□	F9ED	E748	桜
□	F9C8	E723	冷や汗 2	□	F9EE	E749	おにぎり
□	F9C9	E724	ぷくっく顔	□	F9EF	E74A	ショートケーキ
□	F9CA	E725	ポケーとした顔	□	F9F0	E74B	とっくり (おちょこ付き)
□	F9CB	E726	目がハート	□	F9F1	E74C	どんぶり
□	F9CC	E727	指で OK	□	F9F2	E74D	パン
□	F9CD	E728	あっかんべー	□	F9F3	E74E	かたつむり
□	F9CE	E729	ウィンク	□	F9F4	E74F	ひよこ
□	F9CF	E72A	うれしい顔	□	F9F5	E750	ペンギン
□	F9D0	E72B	がまん顔	□	F9F6	E751	魚
□	F9D1	E72C	猫 2	□	F9F7	E752	うまい!
□	F9D2	E72D	泣き顔	□	F9F8	E753	ウッシッシ
□	F9D3	E72E	涙	□	F9F9	E754	ウマ
□	F9D4	E72F	NG	□	F9FA	E755	ブタ
□	F9D5	E730	クリップ	□	F9FB	E756	ワイングラス
□	F9D6	E731	コピーライト	□	F9FC	E757	げっそり

注意事項：

- 拡張絵文字をサポートしていない携帯電話では、拡張絵文字はスペースなどに置換されて表示されます。どのような文字に置換されるかは機種により異なります。

第 7 章

ScratchPad とリソースの使用

この章では、i アプリ対応携帯電話上でアクセス可能な ScratchPad およびリソースの使用方法について解説します。

ScratchPad とは、携帯電話上に i アプリで使用する永続データを保存するための記憶領域です。またリソースとは、i アプリの JAR ファイルに同梱されたデータファイル（画像ファイルやサウンドファイルなど）のことです。ScratchPad もリソースも、第 5 章で解説した HTTP 通信と同様に、Generic Connection フレームワークを使用してアクセスを行います。

7.1 ScratchPad に対する読み取りと書き込み

i アプリは、永続データ（長期間の保持・保存が必要なデータ）の主要な格納場所としてサーバーもしくは ScratchPad を使用します。i アプリ実行環境では、i アプリの実行終了後もデータを保存し続け、次にその i アプリが実行されたときにそれを再び利用できるようにするための、ScratchPad と呼ばれる特別な記憶領域が個々の i アプリに与えられます。使用可能な ScratchPad サイズには、3.2 項で解説した制限があります。さらに、ある i アプリに割り当てられた ScratchPad にアクセスできるのは、それを作成した i アプリだけです。

i アプリで ScratchPad を使用するには、ADF の **SPsize** キーを使ってその i アプリの使用する ScratchPad サイズを指定する必要があります。アプリケーションマネージャは、指定された ScratchPad サイズと JAR ファイルサイズの両面から、携帯電話にその i アプリをダウンロードできるスペースがあるかどうかを判定します。ADF の記述方法については、16.5.1 項を参照してください。

ScratchPad の記憶モデルには、トランザクションの概念の一部が導入されています。ScratchPad に対する変更は、必ず全部が書き込まれるか、全部が拒否されます。

アプリケーションマネージャは、ADF に ScratchPad サイズが指定されている i アプリごとに記憶領域を作成し、それを管理します。アプリケーションマネージャが記憶装置にデータを実際に書き込むタイミングは、メーカーのメモリやシステムモデルの最適化方法によって異なります。

ScratchPad へのアクセスは Generic Connection フレームワークを使用して行います。

ScratchPad の URL の書式は以下の通りです。

```
scratchpad:///<number> [;pos=<start-point> [, length=<access-length>] ]
```

アプリケーションに割り当てられた ScratchPad は、ADF の **SPsize** キーの指定により最大 16 個まで分割管理することができます。<number>には、何番目の ScratchPad にアクセスするかを 0～15 までの数字で指定します。

また、pos オプションの値<start-point>には指定された番号の ScratchPad におけるアクセス開始オフセットを、length オプションの値<access-length>にはアクセス開始オフセット位置からのバイト数を指定します。pos オプションを指定した場合は、その位置からデータの読み取りま

たは書き込みが行えます。また **pos** オプションとともに **length** オプションを指定した場合は、そのストリームの長さは **length** オプションで指定された長さとなります (**pos** オプションを指定せずに **length** オプションを指定することはできません)。**pos** オプション、**length** オプションとも、それらを適用した結果 **ScratchPad** の領域 (分割管理されている場合は分割された個々の領域) を逸脱する値を指定することはできません。

Connector.open() に対して **ScratchPad** の URL を指定すると、このメソッドは **StreamConnection** の実装インスタンスを返します。アプリケーションプログラムは、この **StreamConnection** インスタンスから入出力ストリームを取り出して **ScratchPad** へのアクセスを行います。

なお **ScratchPad** アクセスは、性能やリソースに制約のある携帯電話にとってコストの高い処理です。大きな **ScratchPad** 領域を使用する i アプリでは、**ScratchPad** アクセス時の URL に **pos** オプションおよび **length** オプションを適切に指定することで、必要なエリアだけを最低限のメモリ使用量でアクセスできるようにしてください。機種によっては、メディアデータを **ScratchPad** から読み込む場合 (**MediaManager** クラスのメディアデータ取得メソッドに **ScratchPad** の URL を指定する場合)、その URL に適切な **length** オプションを設定することでアプリケーションの実行パフォーマンスが向上する場合があります。

次の 2 つの例は、**ScratchPad** への書き込みと **ScratchPad** からの読み取りをどのように行うかを示したものです。

例: **ScratchPad** 領域への書き込み

```
public void save() {
    try {
        OutputStream out = Connector.openOutputStream("scratchpad:///0")
        out.write((hi_score >> 24) & 0xff);
        out.write((hi_score >> 16) & 0xff);
        out.write((hi_score >> 8) & 0xff);
        out.write(hi_score & 0xff);
        out.close();
    } catch (IOException e) {
        // ScratchPad 入出力の例外処理
    }
}
```

例: **ScratchPad** 領域の読み取り

```
public void load() {
    try {
        InputStream in = Connector.openInputStream("scratchpad:///0")
        hi_score = in.read() << 24;
        hi_score |= (in.read() << 16);
        hi_score |= (in.read() << 8);
        hi_score |= in.read();
        in.close();
    } catch (IOException e) {
        // ScratchPad 入出力の例外処理
    }
}
```

例: **ScratchPad** 領域の読み取り

// **ScratchPad** の 4 番目のバイトから始まる UTF (つまり Unicode) 文字列を取り出す。
// 最初の 4 バイトは他の用途 (前の例では 4 バイトの数値) で使用されている。

```
public class UserNameScreen extends Panel implements TimerListener {
    private String name;
```

```

/** このクラスのインスタンスを作成する。 */
public UserNameScreen() {
    try {
        DataInputStream in = Connector.openDataInputStream(
            "scratchpad:///0;pos=4");
        name = in.readUTF();
        in.close();
    } catch (IOException e) {
        // ScratchPad 入出力の例外処理
    }
    ...
}
}

```

注意事項：

- ScratchPad への **StreamConnection** をオープンする場合、オープンモードとして **Connector.READ**、**Connector.WRITE**、**Connector.READ_WRITE** の各モードが指定できます。**Connector.READ_WRITE** モードでオープンした ScratchPad からは入力ストリーム、出力ストリームのいずれも取得できます。ただし、1つの **StreamConnection** について、入力ストリームと出力ストリームを同時に有効な状態にすることはできません。一方をオープンする際には、必ず他方がクローズされた状態となるようにしてください。
- Jar インフレーター機能 (**com.docomostar.util.JarInflater** クラス) に Jar ファイルを ScratchPad からの読み込みストリームとして与える場合、ScratchPad の URL には **pos** オプションおよび **length** オプションを使用して、厳密な Jar ファイル格納位置を指定するようにしてください。これを行わない場合、ファイルフォーマットエラーとして例外がスローされることがあります。

以下に、ScratchPad に格納された Jar ファイルの内容を Jar インフレーター機能を使用して読み込む例を示します。

例: ScratchPad に格納された Jar ファイルの利用

```

// ScratchPad 先頭から 1K バイトの位置に格納された、長さ 1867 バイトの
// Jar ファイルを使用する。
try {
    String url = "scratchpad:///0;pos=1024,length=1867";

    // ScratchPad からの読み込みストリームをオープンする。
    InputStream is = Connector.openInputStream(url);
    // Jar インフレーターオブジェクトを生成する。
    JarInflater ji = new JarInflater(is);
    // Jar インフレーターオブジェクトが生成された後は、コンストラクタに指定した
    // 読み込みストリームは不要となる。
    is.close();

    // Jar ファイルに含まれるファイルのエントリ名を指定して、目的のファイルに
    // アクセスするための読み込みストリームをオープンする。
    InputStream is2 = ji.getInputStream("datafile.bin");
    for (;;) {
        // 読み込んだデータを処理する。
        ...
    }
    // Jar エントリ読み込み用のストリームをクローズする。
    is2.close();
    // Jar インフレーターをクローズする。
    ji.close();
} catch (IOException ioe) {
    // ScratchPad アクセスでエラーが発生した場合はこの例外が発生する。
    ...
} catch (JarFormatException jfe) {
    // Jar インフレーターが Jar のファイルフォーマットエラーを検出した場合は
    // この例外が発生する。
    ...
}

```

なお、**JarInflater** クラスの詳細については API リファレンスを参照してください。

7.2 リソースの読み取り

JAR ファイルに同梱される画像ファイルやサウンドファイルなどのデータファイルをリソースと呼びます。リソースに対しても Generic Connection フレームワークの枠組みでアクセスすることができます。リソースは、読み取り専用でのみオープン可能です。リソースにアクセスするには、`Connector.open()` メソッドのパラメータで **READ** モードであることを明示的に指定するか、`Connection` オブジェクトを得ることなく入力ストリームを得るための簡易 API である `Connector.openInputStream()` を使用しないと例外がスローされます。static メソッド `Connector.open()` には、3 つのパラメータがあります。

- 1) 接続先の URL
- 2) オープンモード
- 3) タイムアウト例外要求フラグ (true または false)

リソースにアクセスする場合、最初のパラメータは `"resource:/// <jar-relative-path>"` という形式の URL 文字列です。<jar-relative-path>の部分には、JAR ファイルに格納されたリソースの相対パスを指定します。2 つめのパラメータには、`Connector.READ` を指定します (リソースは読み取り専用であるため)。3 つめのパラメータは、HTTP(S)接続および IrOBEX 接続をオープンする場合を除き、通常 `false` を指定します。短縮形式の `open()` メソッド (パラメータが URL だけのもの) はオープンモードが `Connector.READ_WRITE` になりますので、リソースアクセス時には使用しないでください。

リソースの URL は、メディアデータにアクセスする際の URL 文字列にも指定することができます。これにより、JAR ファイルに同梱された画像ファイルやサウンドファイルを、アプリケーションから容易に再生することができます。

次に、メディアデータとしてリソースを使用する例を示します。この例では、JAR ファイルに同梱された `bank.gif` というファイルを取り出して、`VisualPresenter` に設定する処理を行っています。

例:リソースの利用

```
/** welcome 画面を表示する。 */
class WelcomePanel extends Panel {

    /** このクラスの新しいインスタンスを作成する。 */
    public WelcomePanel() {
        final Panel instance = this;
        MediaImage bank;

        VisualPresenter vp1 = new VisualPresenter();
        try {
            bank = MediaManager.getImage(
                "resource:///BankingDemo/img/bank.gif");
            bank.use();
        } catch (Exception ce) {
            // リソース取得時に発生した例外の例外処理
        }
        vp1.setImage(bank);
        ...
    }
    ...
}
```

7.3 エラー処理

ScratchPad またはリソースの取得先を指定した URL を使用する i アプリでは、次の例外を処理する必要があります。

ConnectionNotFoundException

URL が無効であったときにスローされます。無効とは、指定された URL の形式を i アプリ実行環境が解釈できない、または指定された URL の種別とオープンモードが矛盾していることを指します。

ConnectionException

ステータス： SCRATCHPAD_OVERSIZE

ScratchPad のサイズを越えて書き込みを行おうとした場合にスローされます。

ステータス： ILLEGAL_STATE

接続オブジェクトの状態に反したメソッド呼び出しが行われた場合にスローされます。接続オブジェクトの状態に反したメソッド呼び出しとは、例えば READ モードで作成した接続オブジェクトに対して `openOutputStream()` メソッドを呼び出すようなことを指します。

ステータス： BUSY_RESOURCE

許された数以上のスレッドが同時に ScratchPad やリソースをオープンしようとしたときにスローされます。

その他の例外条件が発生すると、`IOException` または `IOException` のサブクラスがスローされます。

第 8 章

プラットフォームリソースのアクセス

i アプリから携帯電話のプラットフォームリソースを制御するために、`com.docomostar.system` パッケージに `PhoneSystem` クラスが定義されています。

`PhoneSystem` クラスで提供する機能は、大きく以下の 4 つに分かれます。

- プラットフォームリソースの属性値の設定、参照および制御可否の取得
- 携帯電話組み込みの効果音の鳴動
- 個体識別情報の参照
- i アプリ 自動起動可否の設定および参照

(a) プラットフォームリソースの属性値の設定、参照および制御可否の取得

`PhoneSystem` クラスには、プラットフォームリソースの属性値を設定する `setAttribute()` メソッド、プラットフォームリソースの属性値を参照する `getAttribute()` メソッド、および特定のプラットフォームリソースをその機種上で制御可能かテストする `isAvailable()` メソッドがあります。プラットフォームリソースを制御するには、プラットフォームリソースタイプとそれに対応する属性値を指定します。i アプリ実行環境で規定されている標準的なプラットフォームリソースタイプには以下のようなものがあります。

- ・ ディスプレイのバックライト（オンまたはオフに切り替え可能）
- ・ バイブレータ（オンまたはオフに切り替え可能）
- ・ 機体の折りたたみ状態（属性値の参照のみ 折りたたみ型、フリップ型の携帯電話のみ有効）
- ・ メール・メッセージ受信状態（属性値の参照のみ）
- ・ 電池パック残量（属性値の参照のみ。残量を何段階で取得できるかも調べることができます。）
- ・ 電源の種別（属性値の参照のみ）
- ・ 電波強度（属性値の参照のみ）
- ・ マナーモード設定状況（属性値の参照のみ）
- ・ エリア情報（属性値の参照のみ）
- ・ 画面可視・不可視状態（属性値の参照のみ）
- ・ 外部音声出力機器（有線イヤホン、Bluetooth イヤホンなど）の接続状態（属性値の参照のみ）

これらの属性は全て `PhoneSystem` クラスで定義されています。

【Star-1.0】 【Star-1.1】

Star-1.0 プロファイルでは、電池パック残量の参照は携帯電話のディスプレイに表示されるアイコンの表示内容に連動します。そのため、電池パック残量は充電中は参照できません。また、電源の種別の参照についてもアイコンの表示内容に連動します。

Star-1.1 プロファイルでは、電池パック残量の参照はアイコンの表示とは関係なく残容量を表します。したがって、充電中でも電池パック残量を参照することができます。また、電源の種別の参照においてもアイコンの表示とは関係なく現在の電源の種別を表します。

【Star-1.2】

Star-1.2 プロファイルでは、エリア情報リソースについて HSPA への対応のための拡張が行われています。Star-1.2 プロファイルで新設された属性値 `ATTR_AREAINFO_HSPA` は、HSDPA 圏内、かつ HSPA 圏内であることを表します（ただし実際に高速通信が可能かどうかは、エリアの基地局の利用状況に依存します）。なお、HSPA のエリアは HSDPA のエリアに包含される関係にあり、`ATTR_AREAINFO_HSDPA` は HSDPA 圏内、かつ HSPA 圏外であることを表します。

【Star-1.5】

外部音声出力機器の接続状態の参照は Star-1.5 プロファイルで追加されました。

注意事項：

- メール・メッセージ受信状態、電波強度、およびエリア情報、マナーモード設定状況は、携帯電話のディスプレイ上に表示される各種アイコンの表示内容に連動します。i アプリがこれらを参照するためには、ADF の `GetSysInfo` キーにてその i アプリが携帯電話のアイコン情報を参照することを宣言し、かつユーザーがその i アプリに対しアイコン情報の参照を許可する必要があります。ADF の記述方法については、16.5.1 項を参照してください。
- メール・メッセージ受信状態は携帯電話のメール・メッセージ受信アイコンに連動するため、電波状況により携帯電話がメール・メッセージ着信の通知を受けられないようなケースでは必ずしも正しい結果が得られない場合がありますのでご注意ください。
- 画面のバックライトやバイブレータなど、電力消費の大きいプラットフォームリソースを i アプリが長時間オンにし続けた場合、機種によっては省電力のためシステムにより自動的にオフの状態に戻されます。`PhoneSystem.getAttribute()` メソッドの返すプラットフォームリソース状態には、このようなシステムによる自動的な状態変更の結果も反映されます。
- 携帯電話の設定により携帯電話のキー操作が無効化されている状態では、i アプリはバイブレータの制御を行うことはできません。
- エリア情報リソースは、FOMA の HSDPA エリアに在圏しているかどうかといった、詳細な電波状況も調べることができます。ただし HSDPA エリアに在圏している場合でも、実際に HSDPA の高速通信が行えるかどうかは、その時点におけるエリアの基地局の利用状況に依存します。他のユーザーが多数 HSDPA を利用している場合など、HSDPA エリアに在圏していても HSDPA の高速通信は行われない場合がありますので注意してください。
- 無線 LAN 対応端末において、通信モードが無線 LAN シングルモード（強制的に無線 LAN を使用するモード）の設定となっている場合、エリア情報リソースの属性値は常に圏外を示す値となります。

(b) 携帯電話組み込みの効果音の鳴動

`PhoneSystem` クラスには、携帯電話にあらかじめ組み込まれた効果音を発音させるための `playSound()` メソッドが用意されています。効果音の種類は `PhoneSystem` クラスに定義されていますが、実際に発音される音はメーカーにより異なります。

(c) 個体識別情報の参照

`PhoneSystem` クラスには、i アプリから個体識別情報を参照するための `getProperty()` メソッドも用意されています。個体識別情報とは、デバイスの各個体に割り当てられるユニークな ID です。携帯電話の個体識別情報には以下の 2 種類があり、それぞれ同じ値が複数の個体に割り当てられることはありません。

- ・ 携帯電話本体の識別情報（製造番号）・・・ `PhoneSystem.TERMINAL_ID`
- ・ UIM カードを搭載する携帯電話での、UIM カードの識別情報（製造番号）・・・ `PhoneSystem.USER_ID`

これらの情報は、それ単独でユーザー個人を特定するなどの用途に使用することはできません（これらの情報を元にユーザー個人を特定する手段はありません）。しかし適切なユーザー管理システムとこれらの情報を組み合わせて利用することで、サーバーへのアクセス管理などのシステム管理を容易に実現することができます。

個体識別情報を取得するには、`getProperty()`メソッドの引数に上記のうちいずれの個体識別情報を取得するかを指定します。

注意事項：

- i アプリが個体識別情報を参照するには、ADF の `GetUtn` キーを使用して個体識別情報を参照することを宣言しなければなりません。`GetUtn` キーに値“terminalid”を指定すると携帯電話本体の識別情報を、値“userid”を指定すると UIM カードの識別番号を参照することができます。両方を参照する場合は、これら 2 つをカンマ区切りで併記します。

(d) i アプリ自動起動可否の設定および参照

携帯電話では、ADF の `LaunchAt` キーや i アプリ自動起動 API（i アプリ自動起動 API はトラステッド API のため、ここでは詳細は解説しません）によって i アプリを自動起動（タイマー起動）することができます。しかし、携帯電話システムには、その携帯電話上での i アプリの自動起動を許可するかどうかを表すユーザー設定項目があり、出荷時状態では自動起動不可の設定になっています。Star-2.0 プロファイルに対応する携帯電話では、i アプリ自動起動可否の設定状態を取得したり、i アプリ自動起動可否の設定を変更する機能を提供しています。

i アプリ自動起動可否の設定状態を取得するには、`PhoneSystem` クラスの `getAutoLaunch()` メソッドを使用します。i アプリ自動起動には ADF の `LaunchAt` キーによる起動、および i アプリ自動起動 API による起動の 2 種類があります。`getAutoLaunch()` メソッドの引数には、どちらによる i アプリ自動起動可否設定を取得するかを指定します。このメソッドを呼び出すと、引数で指定された方法による i アプリ自動起動可否を返します。

なお、携帯電話の中には、上記で解説した i アプリ自動起動可否設定（全体設定）に加えて、個々の i アプリごとに自動起動可否設定が可能な機種もあります。このような機種では、`getAutoLaunch()` メソッドの引数で指定された i アプリ個別の自動起動可否設定と全体設定としての i アプリ自動起動可否設定がともに起動可の場合のみ、`getAutoLaunch()` は `true` を返します。

i アプリ自動起動可否を設定するには、`PhoneSystem` クラスの `setAutoLaunch()` メソッドを使用します。このメソッドの引数には、i アプリ自動起動可否を `boolean` 値で設定します。i アプリ自動起動可否が変更されるような設定値を指定して `setAutoLaunch()` メソッドを呼び出すと i アプリはサスペンドし、ユーザーに i アプリ自動起動可否設定の変更を許可するかどうかを確認す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ると、引数に指定した値に応じて i アプリ自動起動可否設定の変更処理が行われます。変更処理が完了すると、i アプリはレジュームし、`setAutoLaunch()` メソッドは `true` を返します。ユーザーの許可が得られない場合、`setAutoLaunch()` メソッドは `false` を返します。

全体設定としての i アプリ自動起動可否設定に加えて、個々の i アプリごとの自動起動可否設定をサポートする機種では、`setAutoLaunch()` メソッドの引数で指定された自動起動可否設定が、すべての i アプリの自動起動可否設定に反映されます。

【Star-2.0】

i アプリ自動起動可否の設定および参照機能は Star-2.0 プロファイルで追加されました。

第 9 章 ミニアプリ

この章では、Star プロファイル上のアプリケーション形態の一つであるミニアプリ（ウィジェットアプリ）について説明します。

ミニアプリとは i アプリの一種であり、i ウィジェットサービスで用いられます。ミニアプリは **WidgetView** 上で同時に複数実行することができます。ミニアプリの開発も **Star API** を使用して行います。

ミニアプリはフルアプリと比べて以下の制約があります。

- ・ アプリケーションサイズ（JAR ファイル及び ScratchPad）が小さい
- ・ 利用できる機能や API が制限されている
- ・ 画面が小さい
- ・ **WidgetView** 上で同時に複数実行されるため、CPU リソース、メモリの割り当て等にも制限がある

（制限される機能や API については付録 A を参照してください）

これらのような制約はありますが、i アプリをミニアプリで開発することにより、ユーザーが好きなときに簡単に呼び出せるアプリケーションサービスを実現することができます。例えば PC の世界で容易にウィジェットの画面が表示されるように、待ち受け画面からワンボタンで、**WidgetView** を呼び出して、ミニアプリの情報をいつでも簡単に見ることができます。

また、フル+ミニアプリにおいてミニアプリからフルアプリを連携起動したり、**WidgetView** から i アプリ一覧を呼び出せるなど、フルアプリの起動経路をユーザーに分かりやすくする効果もあります。

9.1 ミニアプリの概要

9.1.1 ミニアプリの実行環境

ミニアプリは **WidgetView** と呼ばれるミニアプリ実行環境上で動作します。**WidgetView** では、複数のミニアプリを同時に実行することができ、各ミニアプリの画面をディスプレイ上にまとめて表示するようにする（貼り付け設定）ことができます。

フルアプリと同様に、ミニアプリもアプリケーションマネージャによってアプリケーションのダウンロードやダウンロードしたファイルの管理、ダウンロードしたファイルの起動・終了等が行われます。

9.1.2 WidgetView とミニアプリの表示状態

WidgetView は一覧表示状態、個別表示状態、ランチャー表示状態という 3 つの表示状態を持ちます。WidgetView の表示状態の定義を以下に示します。

表示状態	説明
一覧表示状態	WidgetView 上で、起動中の複数のミニアプリが画面に表示されている状態です。起動中のミニアプリのうちの 1 つのミニアプリにフォーカスが当たっていて、ユーザー操作により表示されている各ミニアプリにフォーカスを移動させることが可能です。ただし、この表示状態において、キー入力は起動中のどのミニアプリにもイベント通知されず、ユーザーはミニアプリを操作することができません。
個別表示状態	起動中の 1 つのミニアプリが選択されて画面に表示されている状態です。この表示状態において、キー入力は選択されているミニアプリにイベント通知され、ユーザーはミニアプリを操作することが可能です。個別表示状態で選択されているフル+ミニアプリは、同一 Jar ファイル内のフルアプリを連携起動することも可能です。
ランチャー表示状態	WidgetView がフロントアプリケーションの状態、起動中のミニアプリは画面に表示されておらず、携帯電話で保持しているミニアプリのアイコン一覧が画面に表示されている状態です。ユーザーはこの表示状態において起動させたいアプリを選択することが可能です。

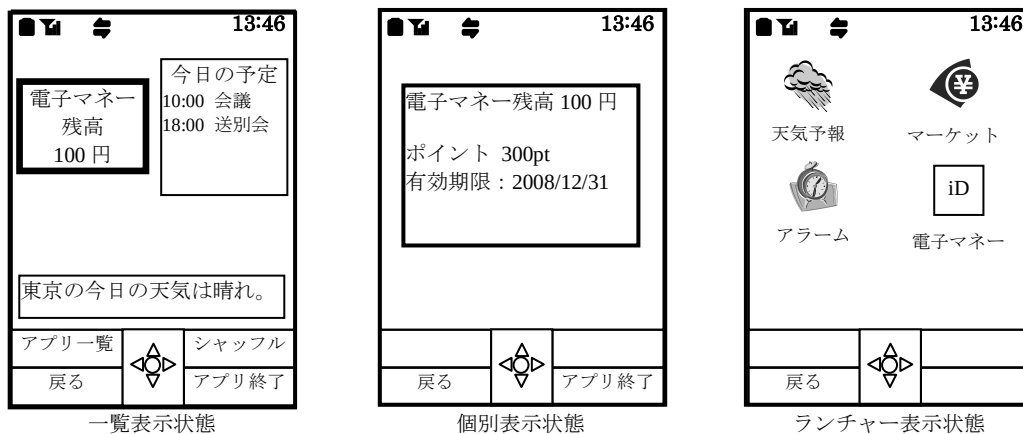


図 12 WidgetView の表示状態例

ユーザーの操作により各表示状態を遷移させることが可能で、各表示状態において、起動させるミニアプリを選択したり、起動中のミニアプリを操作したりと、実行できる機能が異なります。

携帯電話の電源を ON にした直後など、ミニアプリ実行環境やミニアプリが起動していない状態で WidgetView を起動すると、その携帯電話上でミニアプリ実行環境と貼り付け設定されたミニアプリが自動的に起動され、WidgetView は一覧表示状態で表示されます。ミニアプリの貼り付けや貼り付け解除は、ユーザーが自分の好みにあわせて設定することができます。一覧表示状態では、複数のミニアプリの情報を 1 つのディスプレイ上で提供することができます。また、一覧表示状態からユーザーが 1 つのミニアプリを選択して個別表示状態へ遷移すると、そのミニアプリに対する操作が行なえるようになります。一覧表示状態に表示されていないミニアプリを追加起動させたい場合には、ランチャー表示状態に遷移し、表示されているミニアプリのアイコン一覧から起動させたいミニアプリを選択します。また、WidgetView を閉じるとミニアプリ実行環境とミニアプリはサスペンドします。WidgetView を閉じた状態で他のネイティブアプリを起動した場合、ミニアプリ実行環境やミニアプリがすべて終了する場合があります。ミ

ミニアプリ実行環境やミニアプリがサスペンドしている状態で **WidgetView** を起動すると、ミニアプリ実行環境やミニアプリはレジュームします。

WidgetView 上で、ミニアプリは各種の表示状態に遷移します。ミニアプリの表示状態の定義を以下に示します。

ミニアプリの表示状態	説明	備考
Focused 状態	WidgetView の一覧表示状態において複数のミニアプリが画面に表示されている中で、フォーカスを取得しているミニアプリの表示状態です。一覧表示状態においてこの状態にあるミニアプリは必ず 1 つ存在します。	Selected 状態と比較して、ミニアプリが利用可能な機能が制限されます。
Unfocused 状態	WidgetView の一覧表示状態において複数のミニアプリが画面に表示されている中で、フォーカスを取得していないミニアプリの表示状態です。一覧表示状態において表示されているミニアプリの中で、Focused 状態のミニアプリ以外はすべてこの状態です。	
Selected 状態	WidgetView の個別表示状態において、画面に表示されているミニアプリの表示状態です。	ミニアプリは対応している全ての機能を利用可能です。
Unselected 状態	WidgetView の個別表示状態において、画面に表示されていないミニアプリの表示状態です。貼り付け設定されているミニアプリの中で、Selected 状態のミニアプリ以外はすべてこの状態です。	貼り付け設定されているミニアプリはサスペンドしています。

ミニアプリが利用可能な機能について、さらにその一部の機能は **Selected** 状態でしか使用することができないなど制限が設けられているものがあります。それらの機能については付録 A を参照してください。

ミニアプリの表示状態が **Focused** 状態から **Unfocused** 状態、**Unfocused** 状態から **Focused** 状態へ遷移した場合、ミニアプリが利用中の機能の動作に影響はありません。

Focused 状態と **Selected** 状態で動作が同じ機能においては、ミニアプリの表示状態が **Focused** 状態から **Selected** 状態へ遷移、**Selected** 状態から **Focused** 状態へ遷移した場合においてその機能の処理は継続されます。例えば、ミニアプリが実行している HTTP(S)通信、ScratchPad アクセス、携帯電話の個体識別情報取得といった処理はそのまま継続されます。

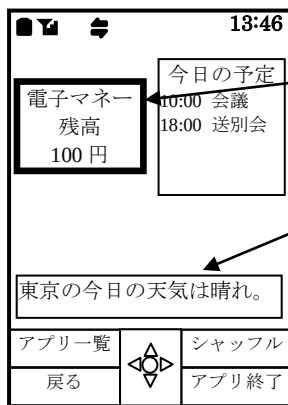
9.1.3 ミニアプリのライフサイクル

ミニアプリのアプリケーションの実行ライフサイクルには、フルアプリと同様の 3 つの実行状態（**Started** 状態、**Active** 状態、**Suspended** 状態）があります。これに加えて、**Active** 状態下において、ミニアプリでは **Full-Active** 状態と **Semi-Active** 状態があります。**Full-Active** 状態はミニアプリが利用可能な機能を何の制限もなく実行可能な状態です。**Semi-Active** 状態はミニアプリが利用可能な機能を実行可能ですが、携帯電話の CPU リソースやメモリ量などが一部制限される状態です。そのため **Semi-Active** 状態では **Full-Active** 状態と比べ実行速度が低下する場合があります。**Full-Active** 状態はフルアプリの **Active** 状態と同じですが、**Semi-Active** 状態は複数のミニアプリが同時実行している状態です。**WidgetView** の表示状態が一覧表示状態の場合は **Semi-Active** 状態で、個別表示状態の場合は **Full-Active** 状態となります。

ミニアプリが起動してから終了するまでの状態の遷移は、**WidgetView** の各表示状態とミニアプリの各表示状態により変化します。**WidgetView** の各表示状態とミニアプリの各表示状態における、ミニアプリの実行状態の定義を以下に示します。

状態	WidgetView の表示状態	ミニアプリの表示状態	ミニアプリの実行状態
①	一覧表示状態	Focused 状態	Started Semi-Active Suspended
②		Unfocused 状態	Started Semi-Active Suspended
③	個別表示状態	Selected 状態	Started Full-Active Suspended
④		Unselected 状態	Suspended
⑤	ランチャー表示状態	—	Suspended

携帯電話の画面上において、どのようなケースで上記のミニアプリの状態となるかのイメージを図に示します。(上記の表中の丸数字の状態が図中と対応します。)



一覧表示状態

貼り付け設定されており、かつフォーカスを取得している 1 つのミニアプリ (①)

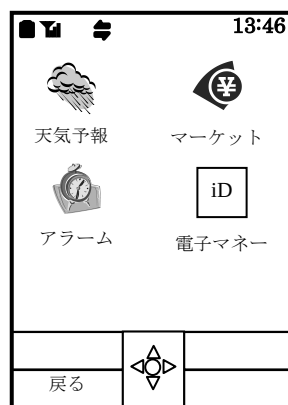
貼り付け設定されており、かつフォーカスを取得していないその他のミニアプリ (②)



個別表示状態

貼り付け設定されているミニアプリ (③)

貼り付け設定されているその他のミニアプリ (④)



ランチャー表示状態

貼り付け設定されているミニアプリ (⑤)

WidgetView が一覧表示状態から個別表示状態に遷移した場合、貼り付け設定されているミニアプリは Suspended 状態に遷移します (④)。しかし、ダウンロード直後にミニアプリが起動された場合などにおいては、そのミニアプリのみが単独で起動され、貼り付け設定されているミニアプリは起動していません。

ミニアプリの表示状態 (Focused / Unfocused / Selected) や実行状態 (Started / Full-Active / Semi-Active) はプログラムから取得することができます。各種状態の取得方法や取得可能な情報の詳細は後述します。

ミニアプリの起動契機と起動後の表示状態を示します。

起動契機	起動後の動作
メニュー (i アプリ一覧など) からの起動	該当のミニアプリのみが単独で起動されます。その他の貼り付け設定されているミニアプリは起動しません。 表示状態は Selected 状態。
ダウンロード直後の起動	
フルアプリからの連携起動	
ランチャー表示状態からの起動 (貼り付け設定されていないアプリ)	選択されたミニアプリが起動されます。 表示状態は Selected 状態。
WidgetView 表示キー押下による起動	貼り付け設定されている全てのミニアプリが起動されます。 表示状態は Focused 状態/Unfocused 状態。

ミニアプリの終了契機とその時の動作内容を示します。

終了契機	動作
電源 OFF	全てのミニアプリが終了します。 貼り付け設定は解除されません。
WidgetView とは同時に起動できないネイティブアプリケーションを起動した場合	全てのミニアプリが終了します。 貼り付け設定されている全てのミニアプリの貼り付け設定は解除されません。
ユーザー操作による強制終了 (「アプリ終了」ファンクションキー押下)	確認ダイアログでのユーザー選択の結果に従い、ミニアプリが終了します。 ミニアプリ終了の場合には、そのミニアプリの貼り付け設定が解除されます。
ミニアプリ自身による終了処理実行	ミニアプリは終了し、貼り付け設定が解除されます。
ミニアプリにてキャッチできない例外発生	ミニアプリは終了し、貼り付け設定が解除されます。

注意事項：

- 音声着信などのシステム割り込みやアプリ連携起動などにより WidgetView がサスペンドしレジュームする際、レジューム後のミニアプリの表示状態はサスペンド前の状態と異なっている場合があります。
- WidgetView の表示状態やミニアプリの表示状態の遷移は、原則としてユーザー操作によってのみ行なわれます。これらの状態遷移を i アプリからの API 呼び出しなどで制御することはできません (ただし個別表示状態のミニアプリが `StarApplication.terminate()` メソッドなどで終了した場合には、ユーザー操作がなくても個別表示状態から一覧表示状態への状態遷移が発生します)。
- `suspend()` メソッドによるサスペンドを解除するための起床イベントには、ミニアプリの状態遷移を指定することもできます。具体的には、`StarEventObject` クラスの `STAR_FACE_STATECHANGED` 定数のイベントを登録します。
- ユーザーのキー押下により WidgetView の表示状態が遷移する場合、その押下されたキーのイベントはミニアプリに通知されません。
- 一覧表示状態の WidgetView が操作されないまましばらく (数分) 経過すると、WidgetView は自動的に非表示となり待ち受け画面等に復帰します。その際、WidgetView 上で実行中のミニアプリは全てサスペンドします。

9.1.4 貼り付け設定

貼り付け設定とは、個々のミニアプリが次回以降の **WidgetView** 呼び出し時にも張り付いたままの状態とする設定のことです。ユーザーはミニアプリを **WidgetView** に貼り付け設定することで、ボタン操作で **WidgetView** を呼び出すと同時にアプリ画面を見ることができるようになります。貼り付け設定可能なミニアプリは最大 8 個です。

貼り付け設定はユーザーがミニアプリを起動した際に自動的に行われます。すでに 8 個のアプリを貼り付け設定済みの場合は、起動だけ行われ貼り付け設定は行われません。

貼り付け設定されたミニアプリは、**WidgetView** を一覧表示状態で表示した際、起動していない場合は自動的に起動され、サスペンドしている場合はレジュームします。

ミニアプリの貼り付け設定が解除される契機を以下に示します。

- ・「アプリ終了」ファンクションキー押下によりミニアプリが終了した場合。
- ・ミニアプリ自身により終了処理を実行した場合（**terminate()**メソッドを呼び出した場合）。
- ・i アプリ一覧から貼り付け設定されているミニアプリを削除し、そのアプリを **WidgetView** が起動しようとした場合。
- ・ミニアプリ実行中のセキュリティ違反によりミニアプリが異常終了した場合。
- ・ミニアプリにてキャッチできない例外が発生し、ミニアプリを終了した場合。
- ・ミニアプリを外部メモリへ移動後に **WidgetView** を表示した場合。

9.1.5 パレット

パレットとは、ミニアプリの描画内容と背景画像（「パレットのフェイス画像」）を合成したもので、**WidgetView** 上のミニアプリ表示の管理単位です。**WidgetView** の一覧表示状態では、貼り付け設定された複数のミニアプリのパレットが表示されます。個別表示状態では選択された一つのミニアプリのパレットのみが表示されます。

パレットのフェイス画像よりミニアプリ描画領域を小さく設定することで、ミニアプリ描画領域に装飾枠のような効果を与えることができます。さらにパレットのフェイス画像に透過 GIF を使用することで、ミニアプリの視覚的な形状(角を丸めるなど)にアクセントをつけることができます。

パレットは **WidgetView** の描画領域に配置され、アプリ名などと共に画面に表示されます。

9.2 ミニアプリの作成

ミニアプリはフルアプリ同様、**com.docomostar.StarApplication** クラスを継承して作成します。API の利用の仕方はフルアプリと同様ですが、利用が制限されたりフルアプリと挙動の異なる API も存在します。ミニアプリにおける制限事項については付録を参照して下さい。

アプリケーションをミニアプリとしてアプリケーションマネージャに通知するためには、ADF の **AppType** キーに “MiniApp” を指定します。配布用のアプリの JAR ファイルの作成方法はフルアプリと同様です。

API に定義されているクラスやメソッドで、ミニアプリのみで利用できる特別なものは存在しません。ただし、主にミニアプリ向けに定義されている参照可能なフィールド変数（定数）として以下があります。これらの変数は全て **StarApplication** クラスで定義されています。

- アプリの表示状態を表す定数

STAR_FACESTATE_MINI_* で定義されており、Focused、Unfocused、Selected を表す 3 種類があります。StarApplication.getAppFaceState() で取得できます。

- アプリの実行状態を表す定数

STAR_STATE_* で定義されており、Started、Full-Active、Semi-Active を表す 3 種類があります。StarApplication.getAppState() で取得できます。また、Active 状態の Full と Semi が変化した場合、StarApplication.stateChanged() のコールバックで引数に設定されます。

- アプリの実行タイプを表す定数

STAR_TYPE_MINIAPPLI で定義されています。StarApplication.getAppType() で取得でき、ミニアプリから実行した場合は必ずこの値が返ります。

- アプリの起動タイプを表す定数

LAUNCHED_FROM_* で定義されており、フルアプリから起動されたことを表す LAUNCHED_FROM_FULLAPPLI や、WidgetView の表示状態の遷移により起動されたことを表す LAUNCHED_FROM_WIDGETVIEW 等が定義されています。アプリ起動時の StarApplication.started() の引数として取得できます。

上記を利用することで、アプリケーション開発者はアプリの実行タイプやミニアプリの各種状態に応じた処理を記述することができます。特に、ミニアプリの表示状態によって利用可否が変わる機能を使用する i アプリでは、現在の表示状態を意識して機能を利用する必要があります。

(A) ミニアプリの状態遷移

ミニアプリの状態遷移については、イベントリスナーを利用することでイベントとして通知を受けることもできます。登録可能なリスナーと通知されるイベントは、用途に合わせて各種定義されています。以下にサンプルコードを示します。

例：ミニアプリでのイベントリスナーの利用

```
public class MiniSample extends StarApplication implements StarEventListener {
    public void started(int launchType) {
        // ミニアプリ状態遷移のイベントリスナーを登録
        addEventListener(StarEventObject.STAR_FACE_STATECHANGE_MINI_SELECTED, this);
        addEventListener(StarEventObject.STAR_FACE_STATECHANGE_MINI_FOCUSED, this);
        addEventListener(StarEventObject.STAR_FACE_STATECHANGE_MINI_UNFOCUSED, this);

        addWakeupEvent(StarEventObject.STAR_FACE_STATECHANGE_MINI_SELECTED);
    }

    // StarEventListener インタフェースの実装例
    public void updateStarApplication(StarEventObject event) {
        switch (event.getType()) {
            case StarEventObject.STAR_FACE_STATECHANGE_MINI_SELECTED: // Selected 状態の振る舞い
                break;
            case StarEventObject.STAR_FACE_STATECHANGE_MINI_FOCUSED: // Focused 状態の振る舞い
                break;
            case StarEventObject.STAR_FACE_STATECHANGE_MINI_UNFOCUSED: // Unfocused 状態の振る舞い
                break;
        }
    }
}
```

(B) パレット

パレット上の座標系は、パレットの左上を原点とし、左右方向に X 軸、上下方向に Y 軸を取ります。X 軸は右方向が正、Y 軸は下方向が正です。座標の単位はドットです。この座標系に対して、パレットのフェイス画像やミニアプリの描画領域を指定します。

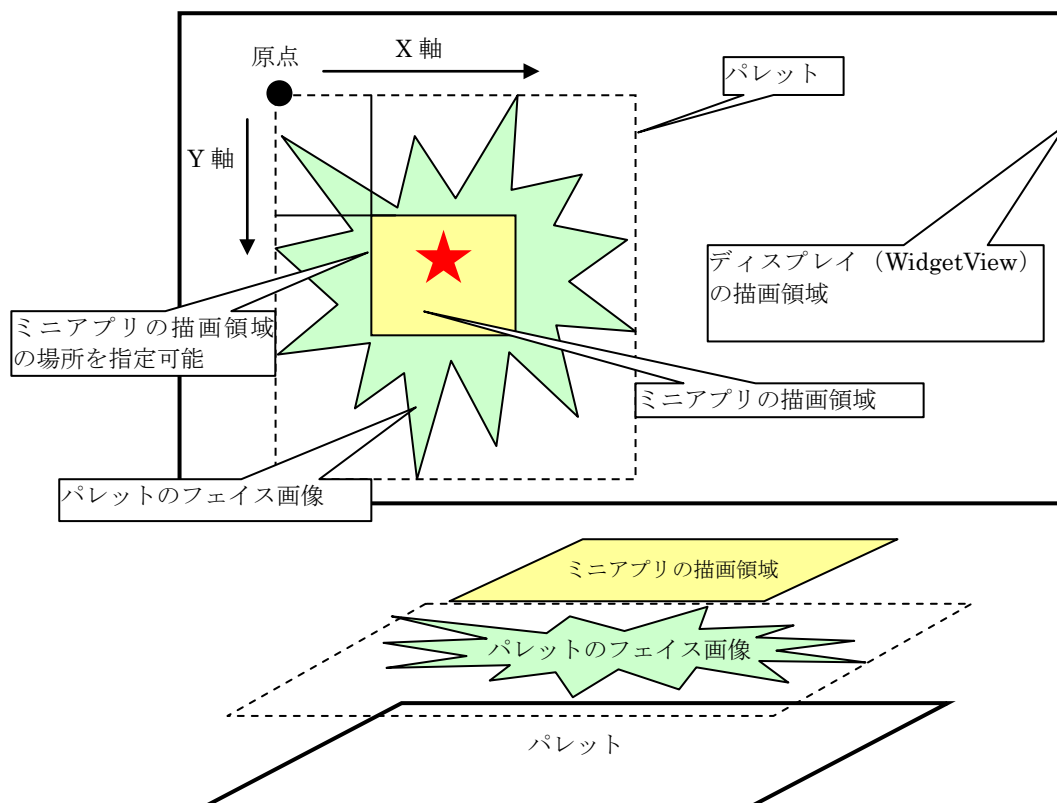


図 13 パレットの概要

パレットのサイズには4種類の固定サイズがあり、以下から選択します。

- ・ 横 440dots × 縦 80dots
- ・ 横 320dots × 縦 240dots
- ・ 横 240dots × 縦 320dots
- ・ 横 160dots × 縦 160dots

上記のサイズ制限により、ミニアプリの全画面表示は行えません。

パレットのフェイス画像よりミニアプリの描画領域を小さく設定することで、ミニアプリの描画領域に装飾枠のような効果を与えることができます。さらにフェイス画像に透過 GIF を使えることを利用して、ミニアプリの視覚的な形状にアクセントを付ける（角を丸めるなど）ことができます。

パレットのフェイス画像は、ミニアプリ用 JAR ファイル直下のディレクトリに格納する必要があります。パレットのフェイス画像の対応フォーマットは GIF（87a、89a の通常画像および透過 GIF。アニメ GIF、インターレース GIF は除く）、JPEG および PNG です。

パレットのフェイス画像は画像の左上をパレットの原点として、パレットに配置されます。パレットのフェイス画像がパレットサイズより大きい場合は、パレットの範囲に納まる分だけが表示されます（表示の自動サイズ調整などは行われません）。また、パレットのフェイス画像がパレットサイズより小さく、パレットにフェイス画像で描画されない領域がある場合は、その部分は透過表示となります。

パレットのサイズ、パレットのフェイス画像のファイル名、パレット内でのミニアプリの描画原点、描画サイズは ADF の `Pallet` キーで指定します。

パレットのフェイス画像の指定が無い場合は、ミニアプリの描画領域を除くパレットの領域が透過表示されます。

(C) フルアプリとミニアプリ間の連携起動

ミニアプリはフルアプリと合わせて1つの JAR ファイル（ダウンロード単位）として作成することもできます。この場合、ミニアプリからフルアプリを起動することや、フルアプリからミニアプリを起動することができます。ミニアプリとフルアプリ間の連携起動については 11.4.3 項を参照してください。

(D) ミニアプリ同士の競合

WidgetView 上では、複数のミニアプリを同時に実行することができます。ただし、起動中の複数のミニアプリから同一機能へのアクセス要求が発生した場合、後発の要求は処理がブロックされたり例外が発生することがあります。

ミニアプリの開発時に注意しておくべき制約を示します。

- JAR ファイル
ミニアプリにおいて、コンパイル後のクラスファイルやリソースファイル群を圧縮した JAR ファイルのサイズは 50K バイト以下とする必要があります。またフル+ミニアプリにおいては、フル+ミニアプリ全体の JAR ファイルに含まれるフルアプリの JAR ファイルのサイズは 10240K バイト以下とする必要があります（3.2 項に記載の制限により、フル+ミニアプリ全体の JAR ファイルと ScratchPad のサイズの合計も 10240K バイト以下である必要があります）。
- ScratchPad
1 つのミニアプリが利用可能な ScratchPad の領域は 200K バイトが上限となります。また、1 つ目のブロック（“`scratchpad:///0`”）のみしか利用できません。
- ヒープメモリ
個々のミニアプリに割り当てられるヒープメモリは、ヒープメモリ全体を複数のミニアプリで分割して使用するため、フルアプリで利用可能なヒープメモリの 1 割程度となります。機種に拠って異なりますが、ミニアプリで利用可能なヒープメモリは概ね 1 M バイト前後となります。
- クラスファイル
フル+ミニアプリにおいて、ミニアプリからはミニアプリの JAR ファイルに含まれるクラスファイルやリソースファイルのみ利用可能です。なお、フルアプリからはフルアプリの JAR ファイルに含まれるクラスファイルやリソースファイル、およびミニアプリの JAR ファイルに含まれるクラスファイルやリソースファイルの双方が利用可能です。

注意事項：

- ミニアプリの描画領域のサイズ（縦横サイズ）はディスプレイのサイズと比較して小さく、また一覧表示状態では、その描画領域がさらに 60%程度のサイズまで縮小して表示されます。従って、ミニアプリを使いやすいものにするためには、表示する情報の量を抑えても大きなフォントを使用し、視認性を上げるようにしてください。例えば、ミニアプリの画面では情報の要約のみを表示し、全情報の閲覧はミニアプリから連携起動されるフルアプリで行うようにすることなどを検討します。
- 一覧表示状態の WidgetView 上では複数のミニアプリが同時に実行されるため、ひとつひとつのミニアプリに割り当てられる CPU 能力は相対的に低下します。また、各ミニアプリへの CPU 能力の割り当ては携帯電話システムが制御し、ひとつのミニアプリが長時間 CPU 能力を占有することはできません。このような特性があるため、ミニアプリはできるだけイベントドリブン型のスタイルで作成するようにしてください。描画処理は、なるべく `Canvas.paint()` メソッドを用いて行うようにします。またミニアプリの動作速度を確認する場合は、他のミニアプリが WidgetView に貼り付けられた状態でテストすることを推奨します。
- ユーザーが WidgetView を呼び出すと、貼り付け設定されたミニアプリがサスペンド状態で待機していない場合にはそれらが全て起動されます。複数のミニアプリが貼り付け設定されていればそれらが順次起動されるため、起動完了までに時間を要する場合があります。WidgetView が呼び出されてからミニアプリが起動される（`StarApplication.started()` が呼び出される）までの間は、ミニアプリの画面（パレット）にはフェイス画像のみが表示されます。このため、ミニアプリでは適切なフェイス画像を設定

するようにしてください。また、ミニアプリは `StarApplication.started()` メソッドが呼び出されたら、なるべく早いタイミングで `Canvas` を作成して初回の描画を行うようにしてください。

- **WidgetView** 上では、複数のミニアプリが 1 つの i アプリ実行環境（仮想マシン）上で実行されます。このため、ミニアプリが仮想マシン上で唯一の資源に対して操作を行うと、他のミニアプリの実行に影響を与えることがあります。アプリケーション開発者は、以下のような点に注意してください。
 - i アプリ実行環境が提供するクラスのクラスオブジェクト（`java.lang.Class` クラスのオブジェクト）、および i アプリ実行環境が提供するクラスが持つ `public static` フィールドを `synchronized` 文の対象とすることは避けてください。これらは仮想マシン上で唯一の資源となるため、これらを `synchronized` 文の対象とすると、同時に実行されている他のミニアプリとの間で影響を及ぼしあう可能性があります。
 - `java.lang.System` クラスが持つ標準出力ストリーム（`out` フィールド）、標準エラー出力ストリーム（`err` フィールド）は仮想マシン上で唯一の資源となるため、これらの状態を変化させるようなメソッド（`close()` メソッドなど）は呼び出さないようにしてください。そのようなメソッドを呼び出すと他のミニアプリに状態変化が伝播し、相互に影響を及ぼしあう可能性があります。

第 10 章

赤外線通信による OBEX 外部接続

i アプリでは赤外線ポートを介して i アプリと外部機器を接続し通信を行うための OBEX 外部接続機能 (IrOBEX) がサポートされています。

i アプリと通信可能な外部機器には DoJa-2.0 プロファイル以降に対応した携帯電話自身も含まれ、また i アプリ仕様では OBEX 外部接続におけるクライアント API およびサーバー API の両方をサポートするため、携帯電話間で名刺やスケジュールなどの小さなデータを交換しあう i アプリを容易に開発することができます。

本章では、i アプリ実行環境における IrOBEX クライアント API、IrOBEX サーバー API、および携帯電話以外の外部機器をターゲットとしたシステムを開発する際のヒントについて解説します。

以降、本章では、IrOBEX を単に OBEX と記載します。

なお、OBEX 外部接続機能は IrDA 規格に基づいて実装されますが、本書では IrDA 規格そのものに関する解説は行いません。これらの規格に関する詳細は、Infrared Data Association の制定している各種規格書を参照してください。なお、OBEX 外部接続機能は、以下のバージョンの各規格に準拠しています。

- Infrared Data Association "Link Manager Protocol version 1.1"
- Infrared Data Association "Serial Infrared Physical Layer Specification version 1.4"
- Infrared Data Association "Tiny TP: A Flow-Control Mechanism for use with IrLMP version 1.1"
- Infrared Data Association "Object Exchange Protocol IrOBEX version 1.2"
- Infrared Data Association "Specification for Ir Mobile Communications(IrMC) version 1.1"
- Infrared Data Association "Serial Infrared Link Access Protocol version 1.1"

10.1 OBEX によるデータ送受信

OBEX (IrOBEX) は、赤外線ポートを介したデータ送受信の手順を「オブジェクトの交換」という形式で規格化したものです。ここでいうオブジェクトとは、一塊となることで意味を持つ、例えばファイルなどの一般的なデータ実体のことを指します。

OBEX 自体はデータ送受信の手順 (プロトコル) を規格化したものであり API を規格化したものではありません。i アプリ仕様では、Generic Connection フレームワーク上に、OBEX によるデータ送受信を取り扱うための API を規定しています。

OBEX の仕様は HTTP を参考として策定されており、OBEX クライアントが送信するリクエストに対して OBEX サーバーがレスポンスを返すクライアント/サーバーモデルに従っています。OBEX クライアントが OBEX サーバーにオペレーション (HTTP におけるメソッドに相当) の処理を要求し、OBEX サーバーはその結果としてレスポンスステータスや、処理結果としてのオブジェクトの内容を OBEX クライアントに返します。クライアントとサーバーの間でオブジ

エクトを送受するためのオペレーションとして、i アプリ実行環境は以下の2つを提供しています。

- GET オペレーション： OBEX クライアントが OBEX サーバーにオブジェクト送信を要求する
- PUT オペレーション： OBEX クライアントから OBEX サーバーにオブジェクトを送信する

プロトコル自体がバイナリ形式で表現されることを除き、リクエストおよびレスポンスがヘッダやオブジェクト本体（ボディ）から構成されること、および規定されているレスポンスステータスの内容などは HTTP と類似しています。

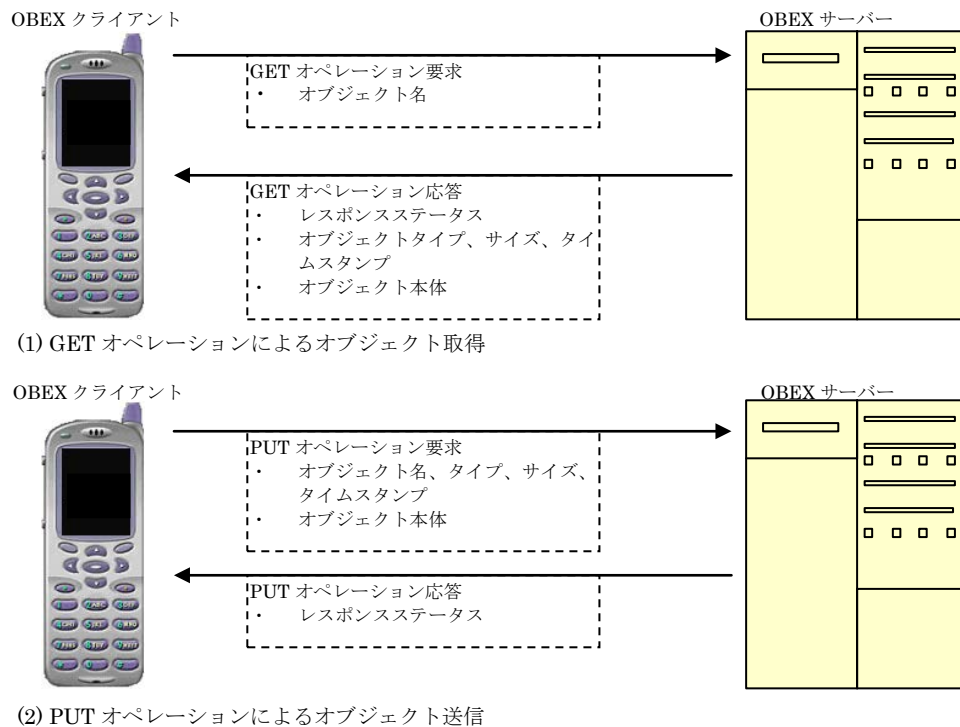


図 14 OBEX クライアントと OBEX サーバーの利用イメージ

OBEX 規格では、赤外線通信の packets 長の制限などに起因するオブジェクトの分割や再構成といった処理を行うための詳細な手順が規定されています。しかし i アプリ実行環境の提供する API がこれらの複雑な規定を隠蔽するため、開発者はより容易に赤外線通信を行う i アプリを開発することができます。

10.2 OBEX 外部接続 API

OBEX 外部接続 API は、Generic Connection フレームワーク上に構築された、i アプリ独自の API セットとして提供されます。OBEX 外部接続 API の構成を以下に図示します。

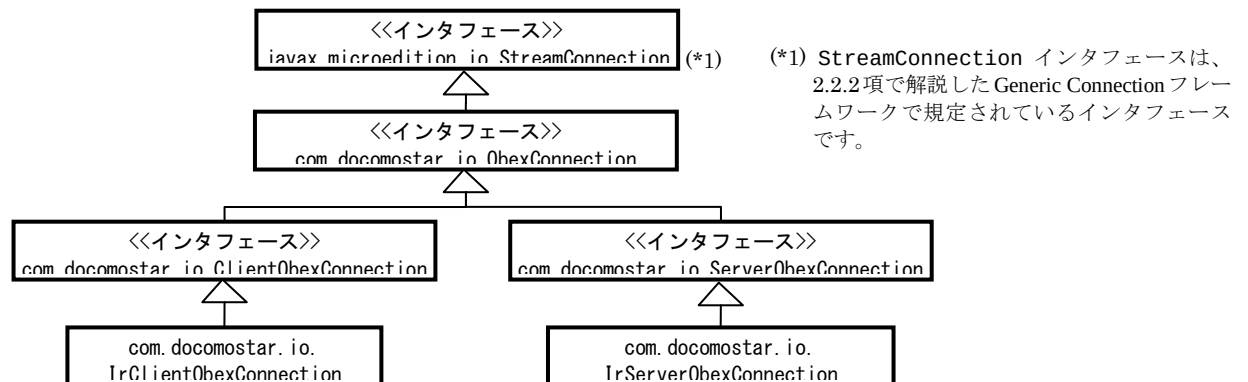


図 15 OBEX 外部接続 API の構成

ObexConnection インタフェースは OBEX クライアントと OBEX サーバーで共通に使用される定数やアクセスメソッドを定義するインタフェースであり、それを継承するサブインタフェースとして ClientObexConnection および ServerObexConnection が用意されています。これら 2 つのサブインタフェースでは、OBEX クライアントまたは OBEX サーバーに固有のメソッドが定義されています。

OBEX 外部接続で使用するコネクションオブジェクトは HTTP 通信や ScratchPad の場合と同様 Connector.open() メソッドを使用して取得しますが、このコネクションオブジェクトは ClientObexConnection または ServerObexConnection を実装しています。Connector.open() メソッドに指定する URL は、OBEX クライアントアプリケーションと OBEX サーバーアプリケーションでは異なります。

以降、OBEX クライアント API および OBEX サーバー API それぞれについて解説します。

10.2.1 OBEX クライアント API

OBEX クライアントが使用するコネクションオブジェクト (IrClientObexConnection クラスのオブジェクト) は ClientObexConnection インタフェースを実装しています。このコネクションオブジェクトを取得するには、アプリケーションプログラムは URL 文字列 "obex:/irclient" を指定して Connector.open() メソッドを呼び出します。

以下に、アプリケーションプログラムで IrClientObexConnection オブジェクトを取得、使用する例を示します。

例 : IrClientObexConnection の利用

```

IrClientObexConnection coc;
int response;
try {
    // IrClientObexConnection インスタンスを取得し、OBEX サーバーに接続する
    coc = (IrClientObexConnection)Connector.open("obex:/irclient",
                                                Connector.READ_WRITE, true);

    coc.connect();

    // GET オペレーションの実行
    coc.setOperation(ObexConnection.GET);           // オペレーションに GET を設定
}
  
```

```

coc.setName("userdata_1");           // GET 対象オブジェクトの名前を設定
coc.sendRequest();                   // オペレーションを送信

// OBEX サーバーから返却されたレスポンスコードをチェックし、正常であれば受信した
// オブジェクト本体を読み込む
response = coc.getResponseCode();
if (response==ObexConnection.SUCCESS) {
    InputStream in = coc.openInputStream();
    :                // GET オペレーションで要求したデータの読み込み処理
    in.close();
} else {
    throw new IOException();
}

// PUT オペレーションの実行
coc.setOperation(ObexConnection.PUT); // オペレーションに PUT を設定
coc.setName("userdata_2");           // PUT 対象オブジェクトの名前を設定
coc.setType("text/plain");           // PUT 対象オブジェクトのタイプを設定
coc.setTime(System.currentTimeMillis()); // PUT 対象オブジェクトのタイムスタンプを設定
OutputStream out = coc.openOutputStream();
:                // PUT オペレーションで送信するデータの書き込み処理
out.close();
coc.sendRequest();                   // オペレーションを送信
// レスポンスコードが正常であれば IrClientObexConnection を閉じて処理終了
response = coc.getResponseCode();
if (response!=ObexConnection.SUCCESS) {
    throw new IOException();
}
coc.close();

} catch (IOException ioe) {
    // 例外処理
    // API がスローする IOException、ConnectionException、
    // アプリケーションプログラムがスローする IOException の処理を行う
    :
}

```

アプリケーションプログラムは `IrClientObexConnection` を使用する際に、まず `connect()` メソッドを呼び出して OBEX サーバーとの間に赤外線通信のリンクを確立します。いったん確立された通信リンク上では、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。前出のサンプルコードでは、1 回確立した通信リンク上で GET オペレーションと PUT オペレーションを 1 回ずつ処理し、その後 `IrClientObexConnection.close()` を呼び出して通信リンクを切断しています。

OBEX クライアントでは、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 OBEX サーバーに送信します。このメソッドは OBEX サーバー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 OBEX サーバーから返却された 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。OBEX クライアント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。OBEX サーバーが OBEX クライアントの要求を正常に処理できた場合、一般的には OBEX クライアント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

(IrClientObexConnection.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。

OBEX クライアントの処理は、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 OBEX サーバーに送信していない状態で OBEX レスポンスヘッダを取得しようとすると例外が発生します。

次の図に、IrClientObexConnection の状態遷移を示します。

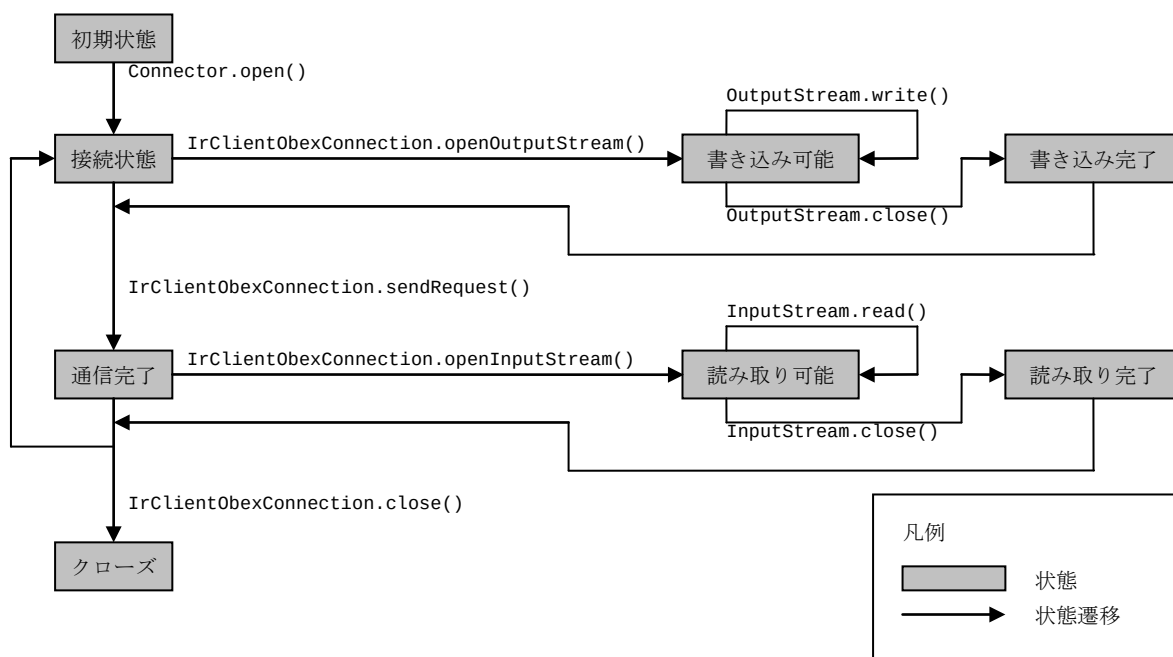


図 16 IrClientObexConnection の状態遷移

注意事項：

- OBEX 外部接続においては、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 IrClientObexConnection 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。
- 携帯電話の赤外線ポートへの通電は、アプリケーションプログラムから IrClientObexConnection.connect() を呼び出した際に、システムにより自動的に行われます。

- OBEX サーバーに対し 1 度のオペレーションで、オブジェクトを送信するとともに OBEX サーバーからオブジェクトを受信することはできません。OBEX クライアントは、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。
- OBEX クライアント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 アプリから同時に確立可能な通信リンクは 1 つだけです。また、i アプリのバックグラウンドで動作しているネイティブ機能を含め、パケット通信が行われている状況では、OBEX 外部接続の通信リンクを確立することはできません。
- あるスレッドが `connect()` メソッドや `sendRequest()` メソッドを呼び出して OBEX サーバーのアクションを待っている時、他のスレッドが `IrClientObexConnection` 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。
- GET、PUT の両オペレーションとも、1 回につき 100K バイトまでのオブジェクトを取り扱うことができます。この制限を越えたサイズのオブジェクトを送受信することはできません。
- アプリケーションプログラムが設定したオペレーション種別や OBEX ヘッダ、オブジェクトデータの内容は、1 回のオペレーションでのみ有効です。1 回の接続で連続してオペレーションを実行する場合、その都度これらを設定しなおすようにしてください。
- OBEX サーバー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。
- 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、i アプリの起動を行うことはできますが起動された i アプリが OBEX 外部接続を行うことはできません。
- i アプリが赤外線通信を行っている間は、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。

10.2.2 OBEX サーバーAPI

OBEX サーバーが使用するコネクションオブジェクト (`IrServerObexConnection` クラスのオブジェクト) は `ServerObexConnection` インタフェースを実装しています。このコネクションオブジェクトを取得するには、アプリケーションプログラムは URL 文字列 "obex:/irserver" を指定して `Connector.open()` メソッドを呼び出します。

以下に、アプリケーションプログラムで `IrServerObexConnection` オブジェクトを取得、使用する例を示します。

例 : `IrServerObexConnection` の利用

```
IrServerObexConnection soc;
try {
    // IrServerObexConnection インスタンスを取得する
    soc = (IrServerObexConnection)Connector.open("obex:/irserver",
                                                Connector.READ_WRITE, true);
    // OBEX クライアントからの通信リンク確立要求を待ち合わせる
    soc.accept();

    // OBEX クライアントからのオペレーション要求を処理する
    while (true) {
        soc.receiveRequest();                // オペレーション要求を待つ
        int operation = soc.getOperation(); // 到着したオペレーションの種別を取得する

        if (operation==ObexConnection.GET) {                // GET オペレーション要求
```

```

// 要求されたオブジェクトの名前を取得
String name = soc.getName();
// 指定された名前に対応するオブジェクトのヘッダ情報を設定
soc.setType("text/plain");
soc.setTime(System.currentTimeMillis());
// 指定された名前に対応するオブジェクトの書き込み処理
OutputStream out = soc.openOutputStream();
:
out.close();
// 正常終了を示すステータスコードを OBEX クライアントに返す
soc.sendResponse(ObexConnection.SUCCESS);
} else if (operation==ObexConnection.PUT) { // PUT オペレーション要求
// 送信されたオブジェクトの各種情報を取得
String name = soc.getName();
String type = soc.getType();
long time = soc.getTime();
// 送信されたオブジェクトの読み出し処理
InputStream in = soc.openInputStream();
:
in.close();
// 正常終了を示すステータスコードを OBEX クライアントに返す
soc.sendResponse(ObexConnection.SUCCESS);
} else if (operation==ObexConnection.DISCONNECT) { // 切断要求
break;
}
}

// 通信リンクを切断し、通信を終了する
soc.close();
} catch (IOException ioe) {
// 例外処理
// API がスローする IOException、ConnectionException の処理を行う
:
}

```

アプリケーションプログラムは `IrServerObexConnection` を使用する際に、まず `accept()` メソッドを呼び出して OBEX クライアントからの赤外線通信のリンク確立要求 (OBEX クライアントの `connect()` メソッド実行) を待ちます。いったん確立された通信リンク上では、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。前出のサンプルコードでは、1 回確立した通信リンク上で、OBEX クライアントからの切断要求を受け付けるまで繰り返しオペレーションの処理を行っています。

OBEX サーバーでは、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、OBEX クライアント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、OBEX クライアントが通信リンク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、OBEX サーバー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、OBEX クライアント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、OBEX クライアント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、OBEX クライアント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。OBEX サーバーが OBEX クライアントの要求を正常に処理できた場合、一般的には OBEX サーバー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 (`IrServerObexConnection.sendResponse()` メソッド) に `ObexConnection.SUCCESS` を返します。

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

次の図に、`IrServerObexConnection` の状態遷移を示します。

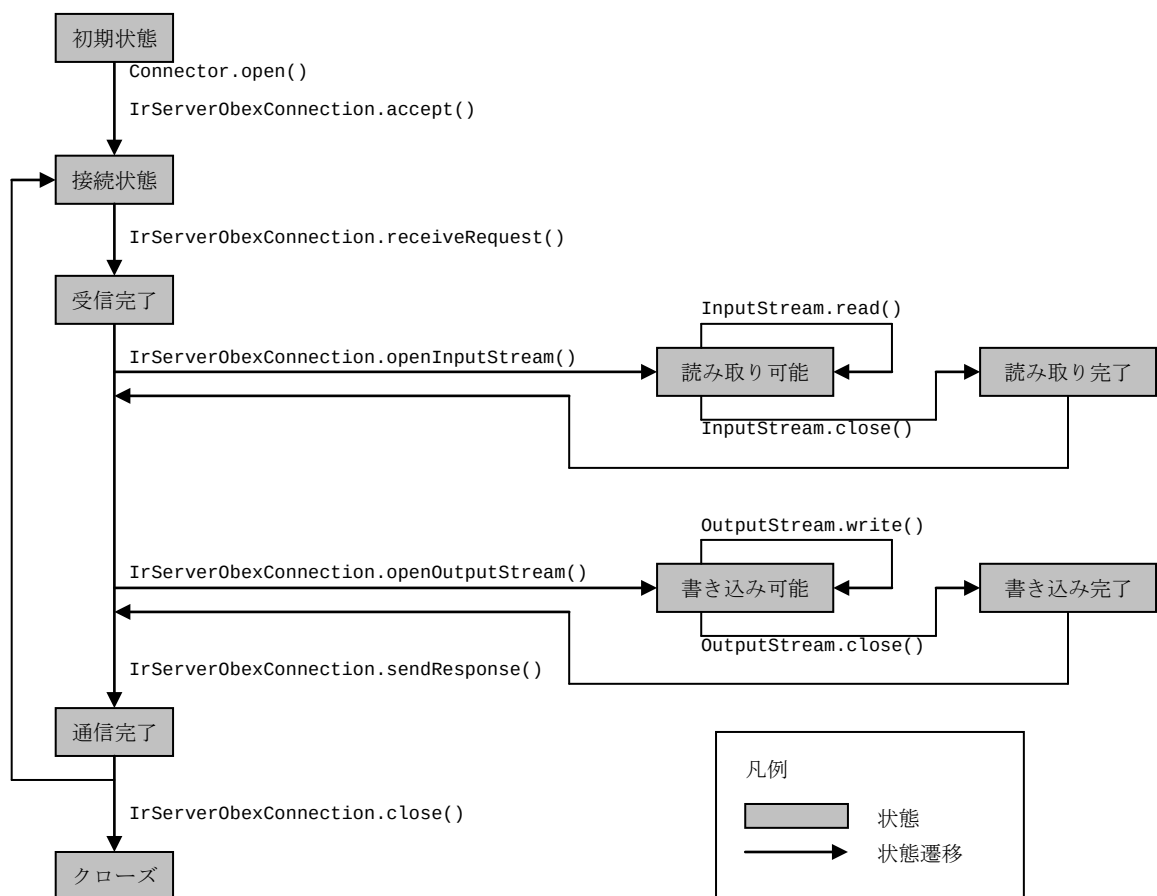


図 17 `IrServerObexConnection` の状態遷移

注意事項：

- OBEX 外部接続においては、`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 `IrServerObexConnection` 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。
- 携帯電話の赤外線ポートへの通電は、アプリケーションプログラムから `IrServerObexConnection.accept()` を呼び出した際に、システムにより自動的に行われます。

- OBEX クライアントからの PUT オペレーションに対して、OBEX サーバーがオブジェクトを送信することはできません。OBEX サーバーは 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。
- OBEX サーバー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。
- 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。また、i アプリのバックグラウンドで動作しているネイティブ機能を含め、パケット通信が行われている状況では、OBEX 外部接続の通信リンクを確立することはできません。
- あるスレッドが `accept()` メソッドや `receiveRequest()` メソッドを呼び出して OBEX クライアントのアクションを待っている時、他のスレッドが `IrServerObexConnection` 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。
- 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。
- 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、i アプリの起動を行うことはできませんが起動された i アプリが OBEX 外部接続を行うことはできません。
- i アプリが赤外線通信を行っている間は、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。

10.3 外部機器接続のヒント

以下に、i アプリ仕様における赤外線通信の下位レイヤ仕様、および携帯電話上の i アプリと携帯電話以外の外部機器との間で赤外線通信を行う上での注意点を示します。

携帯電話間で赤外線通信を行う i アプリを開発する場合はこれらを意識する必要はありません。携帯電話上の i アプリと他の外部機器との間で赤外線通信を行うシステムを構築するようなケースにおいて、外部機器側の設計を行う際の参考としてください。

なお、下記の記載内容に関する規定の詳細については IrDA 規格を参照してください。

<物理的条件>

通信速度： 共通的にサポートされる最大通信速度は 115.2kbps です。

通信距離： 携帯電話に搭載される赤外線デバイスは、IrDA 規格における IrMC Low Power Option に対応しています。このため、通信距離は屋内環境で 20cm 程度に制限されます。

<IrLAP 接続パラメータ>

Baud Rate： 9600bps、38400bps および 115200bps をサポートします。

Data Size： 64byte、128byte、256byte をサポートします。

Link Disconnect / Threshold Time： 3 秒 (Threshold=0 秒)、8 秒 (Threshold=3 秒) をサポートします。

<IrLMP デバイス名およびデバイスニックネーム>

デバイス名およびデバイスニックネームには、以下のような文字列が設定されます。

DOCOMO/<機種名>/j (例： DOCOMO/X01B/j)

<機種名>には、ユーザーエージェントに含まれている名前と同等のものが設定されます。外部機器からこの名前を参照することで、現在通信を行っている携帯電話の機種を判別することができます。

【Star-1.2】

Star-1.2 プロファイル以降、デバイス名およびデバイスニックネームに設定する文字列の一部が変更となりました。「DoCoMo」から「**DOCOMO**」への変更（小文字がすべて大文字に変更）となります。Star-1.1 プロファイル以前では、以下のような文字列が設定されます。

DoCoMo/<機種名>/j （例： DoCoMo/X01A/j）

<IrLMP サービスヒント>

サービスヒントのビットマップには、以下のフラグが ON に設定されます。

Byte1 : Bit1 PDA/Palmtop

Byte1 : Bit7 Extension

Byte2 : Bit8 Telephony

Byte2 : Bit13 IrOBEX

注意事項：

- OBEX で規定されているオペレーションのうち、i アプリ実行環境は OBEX クライアント、OBEX サーバーとも SetPath オペレーションをサポートしません。また、OBEX クライアントは ABORT オペレーションをサポートしません。
- i アプリ実行環境は、OBEX 認証をサポートしません。
- i アプリ実行環境において、OBEX クライアントおよび OBEX サーバーが受け付けることのできる OBEX ヘッダは Name、Type、Time、Body、End of Body の 5 つです。これら以外の OBEX ヘッダを使用することはできません。また、オブジェクトボディ（Body ヘッダおよび End of Body ヘッダ）は他のヘッダより後に送信してください。
- 携帯電話に搭載される赤外線通信機能は IrDA の相互接続認定を受けています。しかしこれは、IrDA 規格に準拠した全ての外部機器との接続性を保証するものではありません。使用する環境や相手側機器のハードウェアおよびソフトウェア特性などにより、接続することができない場合があります。

10.4 IrSimple

携帯電話の中には、オプション機能として i アプリからの IrSimple の利用をサポートするものがあります。IrSimple を利用すると高速、大容量の赤外線通信が行える一方、一部機能の利用に制限が生じるなどの注意事項もあります。

以下に、i アプリからの IrSimple の利用方法とその注意点について記載します。

IrSimple をサポートする機種のために、OBEX 外部接続 API に以下のメソッド、フィールドが追加されています。

● IrClientObexConnection.connect(int mode)

OBEX クライアント側から、通信モードを指定して赤外線通信のリンクを確立します（通信モードとは ObexConnection の COMM_MODE 定数のことです）。IrSimple を利用するための指定は、connect() メソッドに設定する通信モード引数で行います。

● IrClientObexConnection.getCommMode()

● IrServerObexConnection.getCommMode()

リンクが確立されたコネクションオブジェクトから、現在の通信モードを取得します。

● ObexConnection.COMM_MODE_IRDA

通信モード定数で、従来互換の IrDA 通信であることを表します。

● **ObexConnection.COMM_MODE_IRSIMPLE_INTERACTIVE**

通信モード定数で、IrSimple 双方向通信であることを表します。IrSimple 双方向通信では、従来の IrDA などと同様に、OBEX サーバーと OBEX クライアントの間でデータ転送の送達確認や再送制御などを行い、信頼性の高い通信を実現します。

● **ObexConnection.COMM_MODE_IRSIMPLE_UNILATERALLY**

通信モード定数で、IrSimple 片方向通信であることを表します。IrSimple 片方向通信では、OBEX クライアント側から一方通行的にデータを送信し、クライアント・サーバー相互の送達確認や再送制御は行いません。このため、双方向通信のような高い信頼性は得られませんが、より高速、高効率な通信を行うことができます。

IrSimple を使用する場合には、OBEX クライアント側 i アプリの通信リンク確立処理において、**connect(int)** メソッドの引数に **ObexConnection.COMM_MODE_IRSIMPLE_INTERACTIVE** もしくは **ObexConnection.COMM_MODE_IRSIMPLE_UNILATERALLY** を指定してください。

IrSimple を使用する場合、従来の IrDA と比較して以下の点が異なります。

- OBEX クライアントは、GET オペレーションを使用することはできません。IrSimple を使用する場合、OBEX クライアントは PUT オペレーションのみ使用することができます。さらに IrSimple 片方向通信では、1 回の通信リンク確立につき、実行可能なオペレーションの回数は 1 回に限られます。
- OBEX クライアントが一度の PUT オペレーションで送信可能なデータサイズの制限は、2M バイト以内までに緩和されます。

なお、OBEX サーバーが IrSimple に対応していない状況で、OBEX クライアントが IrSimple の利用を試みた場合は以下のような動作となることに注意してください。

1. OBEX クライアントが IrSimple 双方向通信で接続しようとした場合

IrSimple をサポートしていない OBEX サーバーに対して、OBEX クライアントが IrSimple 双方向通信で接続しようすると、IrSimple ではなく、従来互換の IrDA 通信の通信リンクが確立されます。

2. OBEX クライアントが IrSimple 片方向通信で接続しようとした場合

IrSimple 片方向通信では、OBEX クライアントは OBEX サーバーが IrSimple をサポートしているかどうか、さらには対向側に IrDA 機器が存在しているかどうかにも関係なく、一方的にデータを送信し、例外等も発生することなく正常に処理を完了させます。対向側の OBEX サーバーが IrSimple をサポートしていない場合、OBEX クライアントは正常にデータ転送を完了させますが、OBEX サーバー側は接続要求に応答せず、受信待ち状態で待ち続けます。

注意事項：

- IrSimple を使用する際も、**Connector.open()** メソッドに指定する URL 文字列は、通常の IrDA 通信を行う際と同じもの ("**obex:/irclient**"、"**obex:/irserver**") を指定します。
- IrSimple をサポートしていない携帯電話で **IrClientObexConnection.connect(int)** メソッドや **IrClientObexConnection.getCommMode()** メソッドを使用した場合、**UnsupportedOperationException** がスローされます。
- IrSimple 片方向通信では、OBEX クライアントから OBEX サーバーに向けて一方通行的に通信が行われるため、OBEX クライアントのリクエストに対して OBEX サーバーがレスポンスを返すことはありません。しかし、i アプリの OBEX サーバーに IrSimple 片方向通信を処理させる場合には、**ServerObexConnection.receiveRequest()** メソッドでリクエストを受け取ったら、それに対応して **ServerObexConnection.sendResponse()** メソッドを使用するようにしてください（これらのメソッドの利用方法については 10.2.2 項も参照してください）。

第 11 章 アプリケーション連携

i アプリでは、ブラウザやメールなど携帯電話上のネイティブアプリケーションや他の i アプリとの間での連携起動がサポートされています。また i アプリからブックマーク管理機能や画像データ管理機能などネイティブ機能の呼び出しもサポートしています。これらの機能を利用することで、i アプリの機能だけでなく携帯電話の機能をフル活用したサービスを i アプリで提供することができます。

なお、機能の性格上、アプリケーション連携機能の一部はトラステッド i アプリの範囲に含まれます。本書ではトラステッド i アプリに含まれる機能の解説は行いません。本書で解説される機能は、トラステッド i アプリ以外の一般の i アプリでも使用することができます。

現在サポートされているアプリケーション連携機能の内容は以下の通りです。

※ アプリケーション連携起動

- ブラウザからの i アプリ起動
- i アプリからのブラウザ起動
- メールからの i アプリ起動
- 外部機器（赤外線ポート）からの i アプリ起動
- i アプリからの i アプリ起動
- i アプリからの i アプリ更新機能（アプリケーションマネージャ）の起動

※ i アプリからのネイティブ機能呼び出し

- 通話機能の呼び出し（通話発信）
- 電話帳管理機能の呼び出し（電話帳エントリ、電話帳グループの新規登録）
- ブックマーク管理機能の呼び出し（ブックマークの新規登録）
- スケジュール管理機能の呼び出し（スケジュールの新規登録、i アプリからのスケジュールの起動）
- 画像データ管理機能の呼び出し（画像の新規登録・選択読み込み・ID 指定読み込み）
- カメラ機能の呼び出し（撮影および撮影画像の取得）
- 映像データ管理機能の呼び出し（映像データ（i モーション）の新規登録）
- アラーム機能の呼び出し（アラーム時刻の新規登録）
- データ BOX フォルダ管理機能の呼び出し（フォルダの新規登録、フォルダのエントリ ID の選択取得）
- コンテンツ取得保存機能の呼び出し（コンテンツのダウンロード、保存）

本章では、アプリケーション連携機能の利用方法、および各種制限事項などについて解説します。

11.1 ブラウザ連携起動

ブラウザ連携起動では、ブラウザからの i アプリ起動および i アプリからのブラウザ起動を行うことができます。なお、ブラウザからの i アプリ起動においては、起動対象の i アプリはあらかじめ携帯電話にインストールされていなければなりません。

11.1.1 ブラウザからの i アプリの起動

ブラウザから i アプリを起動させるためには、開発者は以下の 2 つを行います。

- (1) インターネット経由でアクセス可能な Web サイト上に i アプリ起動用の HTML ファイルを作成し、その中に i アプリ起動タグを記述します。i アプリ起動タグの記述内容により、そのタグを選択した際に携帯電話上で起動される i アプリが決定されます。
- (2) ADF の LaunchByBrowser キーにて、その i アプリが(1)で作成された HTML の URL からの起動を許可することを宣言します。LaunchByBrowser キーで許可を受けていない URL の HTML からは、その i アプリを起動することはできません。なお、LaunchByBrowser キーで指定された URL と HTML の URL は前方一致で比較され、LaunchByBrowser キーに指定された内容が HTML の URL に全て含まれていれば、その HTML に対し起動許可が与えられます。

ブラウザから i アプリを起動するには、まず i アプリ起動用 HTML をブラウザに表示させます。i アプリ起動タグはリンク形式で表示されます。ユーザーがこのリンクを選択することにより、携帯電話は対応する i アプリを起動します。

以下に、i アプリ起動用 HTML におけるタグの記述例を示します。

```
<OBJECT declare id="application.declaration"
data="http://www.nttdocomo.co.jp/java/abc.jam" type="application/x-jam">
<PARAM name="Param1" value="i-mode">
<PARAM name="Param2" value="i アプリ">
</OBJECT>
i アプリを起動するには
<A ista="#application.declaration" href="notapplicable.html">ここ</A>をクリック。
```

i アプリ起動のために使用されるタグは以下の通りです。

● A タグおよび ista 属性

A タグは、起動対象の i アプリに対応する OBJECT タグを参照するために使用されます。ユーザーがこの A タグを選択することで、携帯電話にインストールされている i アプリが起動されます。A タグの ista 属性には、OBJECT タグの id 属性に指定された名前を指定します。携帯電話上のどの i アプリが起動されるかは、後述の OBJECT タグに指定された ADF URL により決定されます。

なお、ista 属性を持つ A タグでは、必ずペアで href 属性を指定するようにしてください。また、アプリケーションのダウンロードで使用する A タグおよび属性 (ijam 属性) については、第 17 章を参照してください。

● OBJECT タグ

OBJECT タグは、i アプリに対応する ADF を特定するために使用されます。OBJECT タグの id 属性には、A タグの ista 属性で参照される名前 (HTML 内で一意) を指定します。また data 属性には ADF の位置を示す URL を指定します。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。ブラウザからの i アプリ起動では、data 属性に指定された URL と同じ ADF を参照している i アプリが起動対象となります。type 属性には data 属性で示されたデータ (この場合 ADF) のコンテンツタイプを指定します。ADF のコンテンツタイプは "application/x-jam" です。

OBJECT タグの内部には、下記に示す PARAM タグを記述することができます。

● PARAM タグ

PARAM タグは、ブラウザからの i アプリ起動時に、`StarApplicationManager.getParameter ()` メソッドにより取得できるパラメータを指定するために使用されます。

PARAM タグは、OBJECT タグ中に最大 16 個含めることができます。また、1 つの OBJECT タグ中に指定された全 PARAM タグの name 属性の値の長さと value 属性の値の長さの合計は最大 20480 バイトに制限されます。name 属性の値および value 属性の値には、それぞれ SJIS で日本語テキストを指定することができます。

以下に、携帯電話にダウンロードされている i アプリと、i アプリ起動用 HTML の内容の関係について図示します。

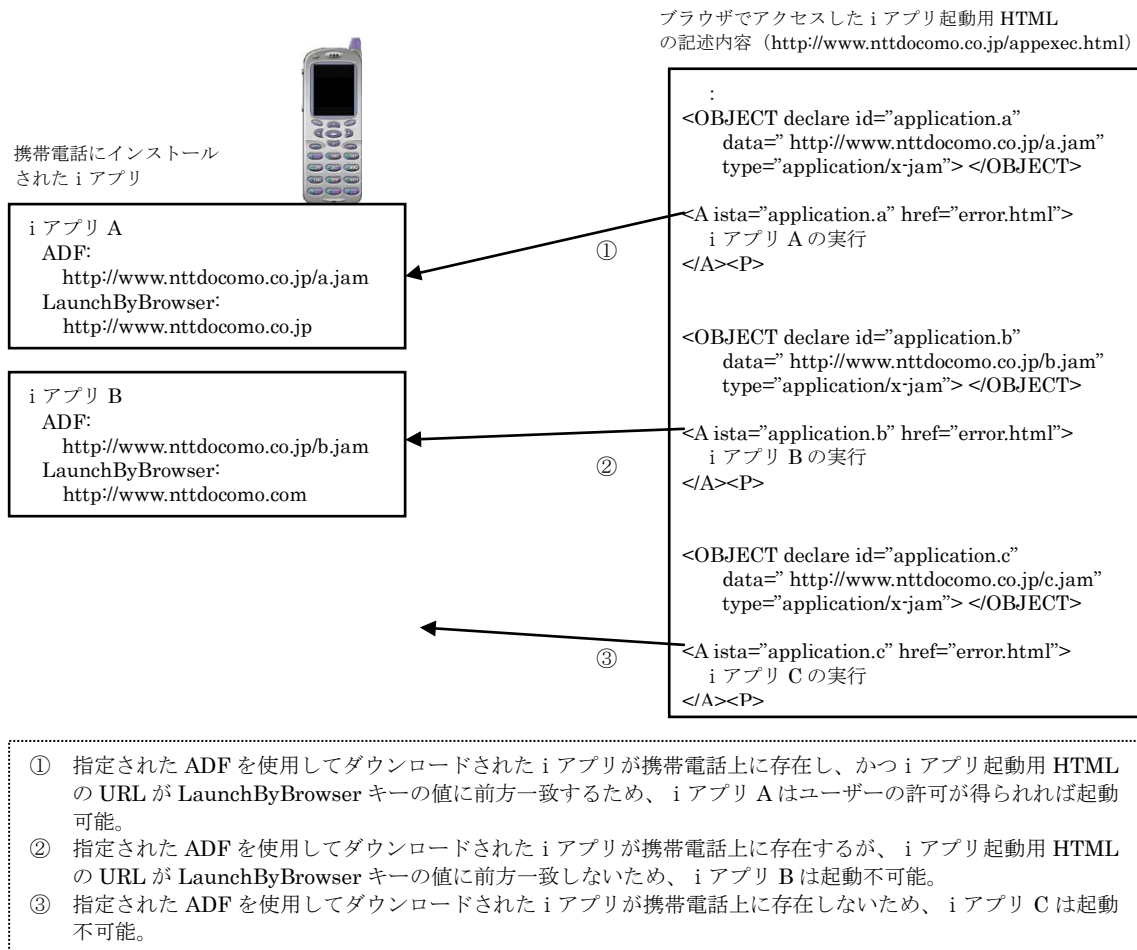


図 18 ダウンロード済み i アプリと i アプリ起動用 HTML の記述内容との関係

トルカ（詳細）の HTML コンテンツ部分に本項で解説したタグを記述することで、トルカビューアから i アプリを起動することもできます。この場合、トルカ（詳細）の HTML コンテンツ部分に記述する OBJECT タグ data 属性に記載する ADF URL には相対 URL は指定しないようにしてください。

トルカビューアからの i アプリ起動を許可するには、その i アプリの ADF に、LaunchByToruCa キーを指定します。各々のトルカデータにはトルカ（詳細）を取得するための URL が対応付けられていますが、トルカ（詳細）の取得先 URL に LaunchByToruCa キーの値が前方一致する場合は、そのトルカから該当の i アプリを起動することができます。

なお、ブラウザから i アプリを起動した際の i アプリの起動種別
(`StarApplication.started(int)`メソッドの引数) が

StarApplication.LAUNCHED_FROM_BROWSER となるのに対し、トルカビューアから i アプリを起動した際の i アプリの起動種別は StarApplication.LAUNCHED_FROM_TORUCA となります。

【Star-2.0】

Star-2.0 プロファイル以降、MyFACE に対応している携帯電話では、MyFACE の HTML コンテンツ部分に本項で解説したタグを記述することで、MyFACE から i アプリを起動することもできます。この場合、MyFACE のコンテンツ部分に記述する OBJECT タグ data 属性に記載する ADF URL には相対 URL は指定しないようにしてください。

MyFACE から i アプリ起動を許可するには、その i アプリの ADF に、LaunchByFaceApp キーを指定します。各々の MyFACE のコンテンツには自身の取得先 URL が対応付けられていますが、MyFACE のコンテンツの取得先 URL に LaunchByFaceApp キーの値が前方一致する場合は、その MyFACE から該当の i アプリを起動することができます。なお、MyFACE から i アプリを起動した際の i アプリの起動種別は StarApplication.LAUNCHED_FROM_FACE となります。

注意事項：

- 携帯電話ユーザーは、携帯電話の設定によりブラウザからの i アプリ起動を禁止することができます。またブラウザから i アプリを起動する前に、i アプリ実行環境はユーザーに i アプリを起動してよいか確認を行います。これらの機構によりユーザーの許可が得られなかった場合、その i アプリの起動は行われません。
 - ブラウザから起動された i アプリの動作が終了（StarApplication.terminate()により終了）した場合、およびユーザーが強制終了した場合は、i アプリを起動する前のブラウザの状態に復帰します。ただし、ブラウザから起動された i アプリがアプリケーション連携機能によって他のネイティブアプリケーションの機能の起動または呼び出しを行った場合は 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。
 - 起動先 i アプリは、フルのみアプリか、フル+ミニアプリでなければなりません。また、起動先 i アプリが、フル+ミニアプリである場合は、フルアプリが起動されます。
 - i アプリ連携起動用の A タグに、ista 属性に加え ijam 属性も指定した場合、i アプリを連携起動しようとした際に、その i アプリが更新されていることが検出されると i アプリの更新処理が行われます。そのような状況のとき、以下の点に注意する必要があります。
 - 更新処理に引き続いて起動しようとする i アプリがミニのみアプリ、あるいは更新処理後の ADF の AppType キーが “MiniApp,FullApp” の場合は、i アプリを起動することはできません。
 - 更新処理に引き続いて起動しようとする i アプリがフルのみアプリ、あるいは更新処理後の ADF の AppType キーが “FullApp,MiniApp” の場合、起動された i アプリの起動種別は StarApplication.LAUNCHED_FROM_BROWSER（ブラウザからの起動の場合）もしくは StarApplication.LAUNCHED_FROM_TORUCA（トルカビューアからの起動の場合）となります。
- また、ista 属性と ijam 属性が併記されていても、携帯電話上に対象の i アプリが存在しない場合には、i アプリのダウンロード処理に引き続いて起動された i アプリの起動種別は StarApplication.LAUNCHED_AFTER_DOWNLOAD となります。
- MyFACE から i アプリをダウンロードしたり、バージョンアップしたりすることはできません。そのため、MyFACE のコンテンツ部分の i アプリ用の A タグに、ijam 属性を指定しても無視されます。
 - MyFACE から起動された i アプリが強制終了された場合、i アプリを起動する前の MyFACE を表示した状態ではなく、待ち受け画面に遷移します。

11.1.2 i アプリからのブラウザ起動

i アプリからブラウザを起動するには、Launcher.launch()メソッドを使用します。ブラウザを起動するには、このメソッドの第 1 引数（起動対象となるアプリケーションの種別）に Launcher.LAUNCH_BROWSER を、また第 2 引数（起動引数を示す String 配列）の先頭要素

に接続先の URL (http スキームまたは https スキームをサポート) を設定します。URL は最長 255 バイトまで指定できます。

なおブラウザを起動する i アプリは、ADF の **UseBrowser** キーでブラウザ起動を行うことを宣言 (値 **"launch"** を設定) する必要があります。この宣言を行っていない i アプリはブラウザを起動することはできません。

Launcher.launch() メソッドを呼び出すと、i アプリはサスペンド状態に入ります。その後、以下の 2 点を満たすことでブラウザが起動されます。

- ・ パケット通信が可能であること (圏外など、パケット通信ができない状態でないこと)。
- ・ ブラウザ起動前、i アプリ実行環境がユーザーに求める起動確認に対しユーザーが同意すること。

いずれかを満たしていない場合、i アプリはレジュームし、**Launcher.launch()** メソッドは例外をスローすることなく呼び出し元に復帰します。また、両方の条件を満たしてブラウザが起動される際、i アプリは終了します。

なお、i アプリからブラウザを起動する際に i アプリを終了させず、サスペンド状態を継続することを指示するための API も用意されています。ただしこの機能はオプションであり、メーカーによってはサポートされない場合があります。

この機能を使用するには上記の手順において、**launch()** メソッドの第 1 引数に、**Launcher.LAUNCH_BROWSER** の代わりに **Launcher.LAUNCH_BROWSER_SUSPEND** を指定します。この方法でブラウザを起動した場合はブラウザ起動後も i アプリはサスペンド状態を保っており、ブラウザ終了時にレジューム復帰します。

注意事項：

- **HttpConnection** を使用した HTTP 通信時と異なり、i アプリからのブラウザ起動では、i アプリダウンロード元サイト以外のサイトにアクセスすることができます。ただしリクエストメソッドは GET に限定され、i アプリ実行環境によるユーザーへのブラウザ起動確認時に URL の内容がユーザーに提示されます。

11.2 メール連携起動

メール連携起動では、ユーザーに配信されたメールから i アプリの起動を行うことができます。なお、起動対象の i アプリはあらかじめ携帯電話にインストールされていなければなりません。

11.2.1 メールからの i アプリの起動

メールから i アプリを起動させるためには、開発者は以下の 2 つを行います。

- (1) ADF の **LaunchByMail** キーにて、対応する i アプリが、特定の送信元から受け取ったメールからの起動を許可することを宣言します。**LaunchByMail** キーで許可を受けていないメール送信元のメールからは、その i アプリを起動することはできません。なお、**LaunchByMail** キーで指定されたメールアドレスとメール送信元アドレスは後方一致で比較され、**LaunchByMail** キーに指定された内容がメール送信元アドレスに全て含まれていれば、そのメールに対し起動許可が与えられます。
- (2) (1)の指定が行われた i アプリをダウンロードしているユーザーの携帯電話に対し、起動許可を得られるメールアドレスからメールを送信します。このメールには、起動する i アプリを決定するための情報や i アプリの起動パラメータなど (i アプリ起動情報) を記述します。

i アプリ起動情報が記述されたメールをメールで閲覧すると、i アプリ起動用のリンクなどが表示されます。ユーザーがこのリンクを選択することにより、携帯電話は対応する i アプリを起動します。

以下に、i アプリ起動情報を含むメールの記述例を示します。

```
.....  
.....  
.....  
(ここまでメールで閲覧可能なメール本文)  
--B:A  
TEXT=" i アプリ A "  
ADF="http://www.nttdocomo.co.jp/java/abc.jam"  
"Param1"="i-mode"  
"Param2"=" i アプリ "
```

(注) 各行とも行末は<CR><LF> (0x0d0a) です。

i アプリ起動情報の記法は以下の通りです。

- --B:A

--B:A は、メール本文と i アプリ起動情報の境界を示すバウンダリ識別子です。この識別子（アルファベット部は大文字である必要があります）のみを含む行以降は i アプリ起動情報と見なされます。i アプリ起動情報は、ユーザーがメールを閲覧する際にその内容を直接見ることはできません。また、メールの返信や転送などにおいても、i アプリ起動情報は引用されません。

- TEXT キー

TEXT キーには、i アプリ起動用リンクで表示される文字列の設定を行います。先の記述例ではメール閲覧時に " i アプリ A " という文字列がリンクとして表示され、ユーザーがそのリンクを選択することで i アプリが起動されます。

- ADF キー

ADF キーには、起動したい i アプリに対応する ADF の URL を指定します。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。メール連携においては、メールに記述された ADF キーの URL と同じ ADF を参照している i アプリが起動対象となります。

- パラメータ指定キー

開発者は、キーの名前と値をそれぞれダブルクォートで囲むことにより、i アプリに引き渡すことのできるパラメータを定義することができます。このように指定されたパラメータは、`StarApplicationManager.getParameter()` メソッドで取得することができます（ブラウザからの i アプリ起動における PARAM タグと同等）。

パラメータ指定キーは、1 つのメール内に最大 16 個含めることができます。また、1 つのメール内で使用されている全パラメータ指定キーの名前の長さの合計は最大 20480 バイトに制限されます。ただし現在の i モードメールで送受信可能な文面の最大サイズはこれよりも小さいため、実際にはパラメータ指定キーで授受できるパラメータのサイズは i モードメールの容量制限の影響を受けます。キーの名前および値には、それぞれ日本語テキストを指定することができます。パラメータに日本語テキストを使用する場合、長さの制限は SJIS 表現に対し適用されます。

以下に、携帯電話にダウンロードされている i アプリと、メールに記述された i アプリ起動情報の内容の関係について図示します。

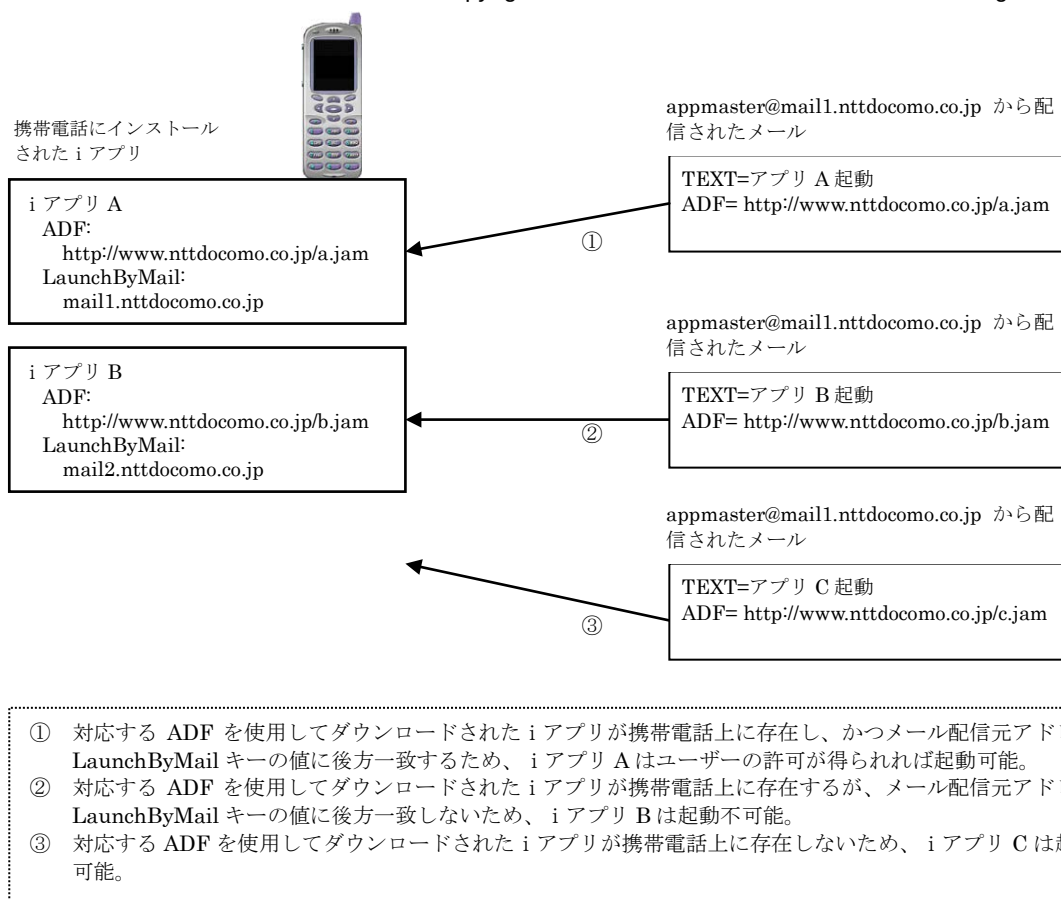


図 19ダウンロード済み i アプリと i アプリ起動情報の記述内容との関係

注意事項：

- 携帯電話ユーザーは、携帯電話の設定によりメールからの i アプリ起動を禁止することができます。またメールから i アプリを起動する前に、i アプリ実行環境はユーザーに i アプリを起動してよいか確認を行います。これらの機構によりユーザーの許可が得られなかった場合、その i アプリの起動は行われません。
- 起動先 i アプリは、フルのみアプリか、フル+ミニアプリでなければなりません。また、起動先 i アプリが、フル+ミニアプリである場合は、フルアプリが起動されます。
- メールから起動された i アプリの動作が終了(StarApplication.terminate()により終了)すると、i アプリを起動する前のメーラの状態（元のメールが表示されている状態）に復帰します。ただし、メールから起動された i アプリをユーザーが強制終了した場合、およびメールから起動された i アプリがアプリケーション連携機能によって他のネイティブアプリケーションの機能の起動または呼び出しを行った場合は 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。
- メールに i アプリ起動情報と他の添付情報（バウンダリ識別子でメール本文と区切られた添付情報 メロディ添付など）を同時に設定することはできません。これらを同時に設定した場合は、いずれもが無効なものとして無視されます。
- メールのサイズは、i アプリ起動情報を含めて i モードメールの制限の範囲内でなければなりません。本機能は i モードメール分割受信には対応しないため、メールの分割が発生しないサイズで送信する必要があります。

11.3 外部機器連携起動

外部機器連携起動では、外部機器から赤外線ポートを介して携帯電話に起動指示を行うことにより、携帯電話上の i アプリの起動を行うことができます。なお、起動対象の i アプリはあらかじめ携帯電話にインストールされていなければなりません。

11.3.1 外部機器からの i アプリの起動

外部機器から i アプリを起動させるためには、開発者は以下の 2 つを行います。

- (1) 外部機器上に、赤外線ポートを介して携帯電話へ i アプリの起動を指示するための OBEX アプリケーションを作成・配置します。外部機器側の赤外線通信に関する要件は 10.3 項を参照してください。外部機器からの i アプリ起動指示は、後述の vTrigger オブジェクトを携帯電話に送信することにより行います。
- (2) 外部機器から起動させたい i アプリに対応する ADF の AllowPushBy キーにて、i アプリが特定の起動コマンドを含んだ vTrigger オブジェクトを送信する外部機器からの起動を許可することを宣言します。AllowPushBy で宣言する起動コマンドと外部機器が送信する起動コマンドが完全に一致していない場合、その i アプリを起動することはできません。AllowPushBy キーは以下のように指定します。ADF の記述方法の詳細については 16.5.1 項を参照してください。

AllowPushBy = Irda:<起動コマンド>

携帯電話の赤外線ポートは、ユーザーによるメニューやボタンなどの操作により通電されます。この状態でユーザーが外部機器と携帯電話の赤外線ポートを互いに対向させ、vTrigger オブジェクトを送受することにより i アプリが起動されます。

vTrigger とは、i アプリ対応携帯電話において、外部機器連携起動をサポートするために規定されたオブジェクトフォーマットです。vTrigger は複数のプロパティ（属性名と値のペア）の集合として表現されるテキストデータです。赤外線ポートを介して vTrigger オブジェクトを携帯電話に送信することにより、携帯電話上の i アプリに起動指示を出すことができます。

以下に、vTrigger の書式について解説します。

```
BEGIN:VTRG
VERSION:<vTrigger フォーマットバージョン>
ADFURL:<起動対象 i アプリの ADF URL>
COMMAND:<起動コマンド>
<パラメータ>
END:VTRG
```

※行末は<CR><LF>（0x0d0a）としてください。

記法は以下の通りです。

- **BEGIN:VTRG**
vTrigger オブジェクトの開始を表します。
- **VERSION:<vTrigger フォーマットバージョン>**
使用している vTrigger オブジェクトのフォーマットバージョンを指定します。現在までのプロファイルにおいては、vTrigger フォーマットバージョンとして"1.0"を指定します。
- **ADFURL:<起動対象 i アプリの ADF URL>**
この vTrigger オブジェクトで起動させたい i アプリに対応する ADF の URL を ASCII 形式で指定します（最大 255 バイト）。
- **COMMAND:<起動コマンド>**
起動コマンドを ASCII 形式で指定します（最大 250 バイト）。ここで指定されている起動コマンドと、i アプリの ADF の AllowPushBy キーで起動許可が与えられている起動コマンド（AllowPushBy キー

の値として"irda:"以降に指定されるコマンド文字列) が完全に一致していなければ、i アプリを起動することはできません。

● <パラメータ>

起動された i アプリに引き渡すパラメータを指定します (オプション)。1 つのパラメータは以下のフォーマットを取ります。

PARAM [**;**<エンコード指定>][**;**<文字セット指定>]:<パラメータ名>;<パラメータ値>

※[] は省略可能であることを示します。なお、エンコード指定と文字セット指定の位置は入れ替わっていても構いません。

パラメータ名およびパラメータ値は、i アプリ起動時に

StarApplicationManager.getParameter() メソッドにより取得できるパラメータを指定するために使用されます。<パラメータ>は 1 つの vTrigger オブジェクトに最大 16 個まで指定することができます。ただし、全ての<パラメータ>の名前と値の合計長は 20480 バイトに制限されます。

エンコード指定および文字セット指定は以下のように指定します。

ー エンコード指定

ENCODE=<エンコード名> の形式で指定します。エンコード名には"QUOTED-PRINTABLE"または"BASE64"のいずれかが指定できます。指定がない場合はエンコード指定が行われていないものと見なされます。

ー 文字セット指定

CHARSET=<文字セット名> の形式で指定します。現在までのプロファイルにおいて、有効な文字セット名は"SHIFT_JIS"のみです。また、文字セット指定がない場合も"SHIFT_JIS"が指定されたものと見なされます。

なお vTrigger オブジェクトでは、ボディの内容は 7 ビットコードで記述されている必要があります。パラメータの名前や値に日本語 (SHIFT_JIS) を指定する場合は、必ず QUOTED-PRINTABLE 形式または BASE64 形式でエンコードするようにしてください。エンコードされた日本語パラメータは、指定されたエンコード種別に従って i アプリ実行環境で自動的にデコードされ、さらに String オブジェクトに変換された上で StarApplicationManager.getParameter() メソッドに渡されます。

● END:VTRG

vTrigger オブジェクトの終了を表します。

以下に、vTrigger オブジェクトの記述例を示します。

```
BEGIN:VTRG
VERSION:1.0
ADFURL:http://www.nttdocomo.co.jp/java/a.jam
COMMAND:startup_Application_A
PARAM:Param1;i-mode
PARAM;ENCODING=BASE64;CHARSET=SHIFT_JIS:Param2;gomDQYN2g4o=
END:VTRG
```

OBEX レイヤにおける vTrigger オブジェクトの送信とは、Body ヘッダに vTrigger オブジェクトの内容を設定し、PUT オペレーションにて送信することをいいます。携帯電話は外部機器からの i アプリ起動要求に対し、i アプリの起動可否を判断して以下の OBEX レスponsコードを返します。

レスポンスコード	OBEX における規定	外部機器連携における規定
0xA0	OK, Success	指定された i アプリの起動が可能である
0xC4	Not Found	指定した i アプリが存在しない
0xE1	Database Locked	指定した i アプリの外部機器からの起動が禁止されている、または AllowPushBy キーの起動コマンドと vTrigger オブジェクトで受信した起動コマンドが一致しない

レスポンスコード 0xA0 が外部機器に返された時点では、携帯電話上ではまだ i アプリは起動されていません。i アプリの起動は、携帯電話がレスポンスコードを返却し、赤外線通信が終了された後に行われます。

以下に、携帯電話にダウンロードされている i アプリと、vTrigger オブジェクトの記述内容の関係について図示します。

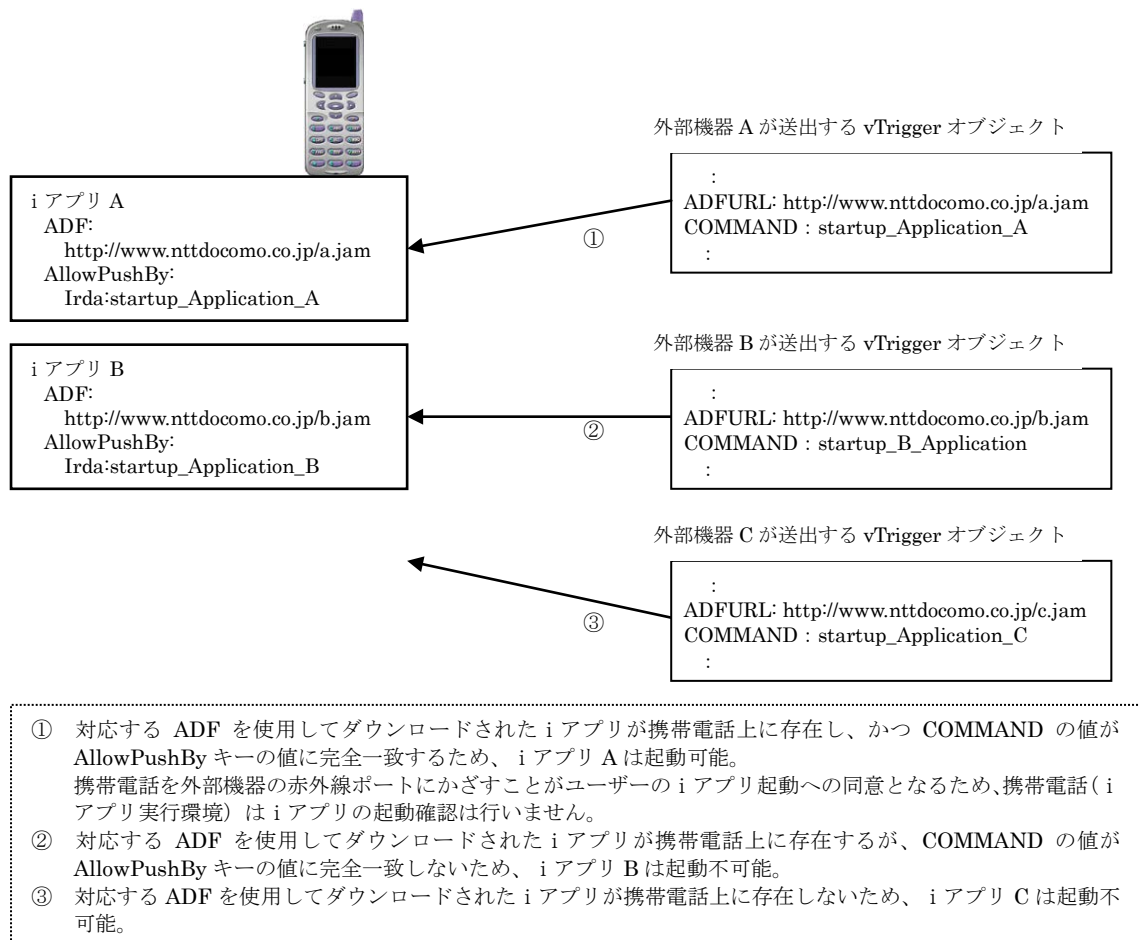


図 20 ダウンロード済み i アプリと vTrigger オブジェクトの記述内容との関係

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機器からの i アプリ起動を禁止することができます。外部機器からの起動が禁止されている場合、vTrigger オブジェクトを受信しても i アプリの起動は行われません。
- 外部機器から起動コマンド (vTrigger オブジェクト) を受信するためには、ユーザーはあらかじめ携帯電話の赤外線ポートへの通電操作を行っている必要があります。赤外線ポートへの通電操作の方法はメーカーにより異なります。
- 外部機器から起動された i アプリの動作が終了すると、待ち受け状態に復帰します。ただし、外部機器から起動された i アプリをユーザーが強制終了した場合、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。
- 外部機器から vTrigger オブジェクトを携帯電話に送信する際、その vTrigger オブジェクトの Name ヘッダの値は末尾 (拡張子) が ".vtg" となるようにしてください。携帯電話は、受信したオブジェクトの名前の拡張子が ".vtg" である場合に、そのオブジェクトを vTrigger オブジェクトと認識します。

- 起動先 i アプリは、フルのみアプリか、フル+ミニアプリでなければなりません。また、起動先 i アプリが、フル+ミニアプリである場合は、フルアプリが起動されます。

11.4 i アプリ連携起動

i アプリ連携起動では、アプリケーションプログラムからの API 呼び出しによって他の i アプリの起動を行うことができます。i アプリ連携起動には、さらに以下の 3 種類の形態があります。

- ・ 連携モード

連携モードでは i アプリが他の i アプリを起動する際に、起動先 i アプリを ADF の URL により直接指定することができます。また、起動元 i アプリは起動先 i アプリにパラメータを渡すことができます。ただし連携モードでは、起動元 i アプリと起動先 i アプリは同一ホストからダウンロードされたものでなければなりません。

- ・ ランチャーモード

ランチャーモードでは連携モードと異なり、起動元 i アプリとはダウンロード元ホストが異なる i アプリでも起動することができます。ただし i アプリを起動する際、起動先 i アプリは ADF の URL ではなく、携帯電話内にダウンロードされている個々の i アプリに対応付けられたエントリ ID を使用して指定します。このため、ランチャーモードで i アプリを起動するには、ターゲットとする起動先 i アプリのエントリ ID をあらかじめ取得する必要があります。i アプリのエントリ ID の取得は、API 呼び出しによりユーザーの確認の下で行われます。ランチャーモードにおいては、起動元 i アプリから起動先 i アプリにパラメータを渡すことはできません。

- ・ 同一パッケージ内での i アプリ間連携起動

同一パッケージ内での i アプリ間連携起動では、フル+ミニアプリにおいて一方の i アプリから他方の i アプリを連携起動することができます。また、連携モードと同様に起動元 i アプリから起動先 i アプリにパラメータを渡すことができます。

なお、起動対象の i アプリはあらかじめ携帯電話にインストールされていなければなりません。

11.4.1 連携モードによる i アプリ連携起動

i アプリを連携モードで起動するには、`Launcher.launch()` メソッドを使用します。i アプリを連携モードで起動する場合、このメソッドの第 1 引数（起動対象となるアプリケーションの種別）に `Launcher.LAUNCH_STARAPPLI` を、また第 2 引数（起動引数を示す `String` 配列）の先頭要素に起動したい i アプリの ADF の URL を指定します。

起動先 i アプリにパラメータを渡す場合には、第 2 引数の `String` 配列の 2 番目以降（配列添え字が 1 以上）の要素を使用します。起動元 i アプリから渡されたパラメータは、起動先 i アプリ側で `StarApplicationManager.getParameter()` メソッドを使用して取得することができます。1 つのパラメータは、変数名と値を示す 2 つの配列要素から構成されます。以下に、起動元 i アプリと起動先 i アプリでパラメータを授受する例を示します。

※起動元 i アプリでのパラメータ設定

※起動先 i アプリでのパラメータ取得

```
String[] args = new String[5];
// 起動先 i アプリの ADF URL 設定
args[0] = "http://www.nttdocomo.co.jp/app.jam"

// パラメータ設定
args[1] = "Param1"; // 1 つ目のパラメータの名前
args[2] = "i-mode"; // 1 つ目のパラメータの値
args[3] = "Param2"; // 2 つ目のパラメータの名前
args[4] = "i アプリ"; // 2 つ目のパラメータの値

// i アプリ連携起動 (連携モード)
Launcher.launch(
    Launcher.LAUNCH_STARAPPLI, args);
```



```
String param1, param2;

// param1 には "i-mode" が格納される
param1 =
StarApplicationManager.getParameter("
Param1");

// param2 には "i アプリ" が格納される
param2 =
StarApplicationManager.getParameter("
Param2");
```

図 21 連携モードにおける起動元 i アプリから起動先 i アプリへのパラメータ授受

パラメータは最大 16 個（パラメータ名と値の組み合わせで 16 セット）まで指定することができます。また、指定するパラメータ名と値の長さの合計は最大 20480 バイトに制限されます（バイト数はデフォルトエンコーディングで評価されます）。

なお i アプリ連携起動を行う i アプリは、ADF の **LaunchApp** キーで i アプリ連携起動を行うことを宣言（値 **"yes"** を設定）する必要があります。この宣言を行っていない i アプリは、i アプリ連携起動を行うことはできません。

注意事項：

- 起動元 i アプリの ADF URL と起動先 i アプリの ADF URL のホスト名は同一でなければなりません。また、起動元 i アプリと同一の ADF URL を持つ i アプリを起動することはできません。
- 起動先 i アプリが起動される際、起動元 i アプリは終了します。
- 起動先 i アプリの動作が終了すると、携帯電話は待ち受け状態に復帰します。ただし、起動先 i アプリをユーザーが強制終了した場合、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。
- 起動元 i アプリがトラステッド i アプリの場合、起動先 i アプリもトラステッド i アプリでなければなりません。
- 起動先 i アプリは、フルのみアプリか、フル+ミニアプリでなければなりません。また、起動先 i アプリがフル+ミニアプリである場合は、フルアプリが起動されます。
- 起動先 i アプリとして、DoJa バイナリ互換環境機能を利用して動作する DoJa アプリを指定することはできません。Star アプリからは Star アプリのみ連携起動することができます。

11.4.2 ランチャーモードによる i アプリ連携起動

ランチャーモードによる i アプリ連携起動を行う場合、開発者は以下の 2 つの処理を起動元 i アプリ（以下ランチャー i アプリとします）に実装します。

なお連携モードと同様、ランチャーモードでの i アプリ連携起動を行う場合も、ADF の **LaunchApp** キーで i アプリ連携起動を行うことを宣言（値 **"yes"** を設定）する必要があります。

(A) 起動先 i アプリのエントリ ID の取得

ランチャーモードで i アプリを起動するには、その携帯電話にダウンロードされている各 i アプリと 1 対 1 に対応付けられているエントリ ID を使用します。エントリ ID はランダムな整数値であり、有効なエントリ ID を得るためにはユーザーによる i アプリ選択プロセスを経る必要があります（アプリケーションプログラムが無効なエントリ ID を指定して i アプリの起動を試みた場合、i アプリ実行環境はユーザーに警告を表すダイ

アログを表示します)。i アプリ選択プロセスは、`com.docomostar.system.ApplicationStore` クラスを使用して行います。このクラスの static メソッド `selectEntry()` を呼び出すと、i アプリはサスペンドして画面上に i アプリ一覧が表示されます。その i アプリ一覧からユーザーがいずれかの i アプリを選択すると i アプリはレジュームし、`selectEntry()` メソッドは選択された i アプリに対応する `ApplicationStore` オブジェクトを返します。返却された `ApplicationStore` オブジェクトに対して `getId()` メソッドを呼び出すことで、選択された i アプリのエントリ ID を取得することができます。

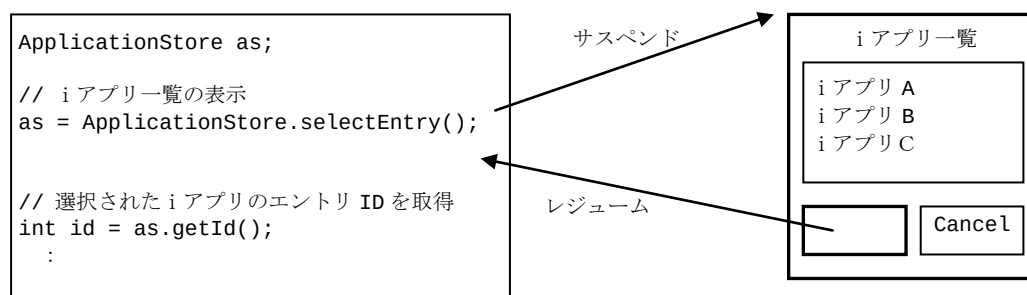


図 22 i アプリのエントリ ID の取得

i アプリに対応付けられたエントリ ID は、その i アプリが携帯電話上に存在する限り有効であり、i アプリが更新された場合でも更新後の i アプリに引き継がれます。

なお、ユーザーが選択した i アプリについて、エントリ ID 以外の情報（例えば i アプリ名など）を取得することはできません。

注意事項：

- Star アプリから `selectEntry()` メソッドを呼び出した場合、DoJa アプリがその携帯電話にインストールされていても DoJa アプリは選択操作の対象外となります。また、`selectEntry()` メソッドを呼び出した i アプリ自身を選択することもできません。

(B) エントリ ID を指定しての i アプリの起動

エントリ ID を指定してランチャーモードで i アプリを起動するには、`Launcher.launch()` メソッドを使用します。`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）には `Launcher.LAUNCH_AS_LAUNCHER` を、また第 2 引数（起動引数を示す `String` 配列）の先頭要素に文字列化したエントリ ID を指定します。

連携モードと異なり、ランチャーモードでは起動先 i アプリにパラメータを渡すことはできません。

注意事項：

- 起動先 i アプリが起動される際、起動元 i アプリは終了します。
- 起動先 i アプリの動作が終了すると、携帯電話は待ち受け状態に復帰します。ただし、起動先 i アプリをユーザーが強制終了した場合、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。
- 起動先 i アプリは、フルのみアプリか、フル+ミニアプリでなければなりません。また、起動先 i アプリがフル+ミニアプリである場合は、フルアプリが起動されます。
- i アプリの内容によってはランチャーモードで起動されることが適切でない場合があります。そのような i アプリは、ランチャーモードで連携起動されることを拒否することができます。その場合、ADF の `LaunchByApp` キーで連携起動の拒否を宣言（値 `"deny"` を設定）します。この設定が行われている i アプリは、i アプリ選択プロセスでの選択対象外となります。
- i アプリ選択プロセスにおいて、選択可能な i アプリが携帯電話上に存在しない場合やユーザーが選択操作をキャンセルした場合は、`ApplicationStore.selectEntry()` メソッドは `null` を返却します。

- 新規にダウンロードされた i アプリにはその時点でランダムなエントリ ID が割り当てられ、その i アプリを削除しない限り（i アプリの更新を行っても）有効なまま存続します。ダウンロードされた i アプリを削除して再度ダウンロードした場合は、新規ダウンロードと見なされ再度ランダムなエントリ ID が割り当てられます。

11.4.3 同一パッケージ内での i アプリ間連携起動

同一パッケージ内での i アプリ間連携起動は、フル+ミニアプリ（ADF の AppType キーに “FullApp,MiniApp” あるいは “MiniApp,FullApp” が指定され、フルアプリ用 JAR ファイルとミニアプリ用 JAR ファイルが 1 つの JAR ファイルにパッケージされている i アプリ）において、一方のアプリから他方のアプリを連携起動する機能です。

同一パッケージ内の i アプリ間で連携起動するには、`com.docomostar.StarApplication` クラスの `changeAppType()` メソッドを使用します。フルアプリからミニアプリを起動する場合は `StarApplication.STAR_TYPE_MINIAPPLI` を、ミニアプリからフルアプリを起動する場合は `StarApplication.STAR_TYPE_FULLAPPLI` を、このメソッドの第 1 引数（変更したいアプリケーションの実行タイプ）に指定します。起動先 i アプリにパラメータを渡す場合には、パラメータ名と値の組を格納した `java.util.Hashtable` オブジェクトを第 2 引数に指定します。パラメータは最大 16 個まで指定することができます。また、指定するパラメータ名と値の長さの合計は最大 20480 バイトに制限されます（バイト数はデフォルトエンコーディングで評価されます）。

アプリの実行タイプに合わせて、ミニアプリとフルアプリ間の連携起動を行うサンプルコードを以下に示します。

例：ミニアプリとフルアプリの連携起動

```
//アプリの実行タイプにより分岐
switch (getAppType()) {
    //ミニアプリからフルアプリを起動
    case StarApplication.STAR_TYPE_MINIAPPLI:
        changeAppType(StarApplication.STAR_TYPE_FULLAPPLI, null);
        break;
    //フルアプリからミニアプリを起動
    case StarApplication.STAR_TYPE_FULLAPPLI:
        changeAppType(StarApplication.STAR_TYPE_MINIAPPLI, null);
        break;
}
```

起動元 i アプリから渡されたパラメータは、連携モードによる i アプリ連携起動の起動先 i アプリ側での処理と同様、`com.docomostar.StarApplicationManager` クラスの `getParameter()` メソッドを使用して取得することができます。

本機能によって起動されたことを表す起動種別（`StarApplication.started()` メソッドの引数として取得する起動種別）として、`StarApplication.LAUNCHED_FROM_FULLAPPLI` と `StarApplication.LAUNCH_FROM_MINIAPPLI` が定義されています。

なお、本機能を使用する i アプリに、連携モードやランチャーモードによる i アプリ連携起動に必要な ADF の `LaunchApp` キーの宣言は不要です。

注意事項：

- 起動先 i アプリが起動される際、起動元 i アプリは終了します。

11.5 i アプリ更新機能連携起動

i アプリ更新機能連携起動では、i アプリからアプリケーションマネージャの持つ i アプリ更新機能を起動することができます。アプリケーションマネージャの i アプリ更新機能とは、携帯電話の i アプリ一覧のサブメニューなどで提供される「バージョンアップ」項目に相当する機能です。i アプリはこの機能を利用して、自身の更新をユーザーに促すことができます。

i アプリ更新機能を起動するには、`StarApplicationManager.upgrade()` メソッドを使用します。

i アプリ更新機能を起動すると、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 `StarApplicationManager.upgrade()` 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) i アプリ実行環境は、パケット通信が可能であること（携帯電話のディスプレイに、通信状態を示す "i" マークが点灯または点滅している状態）を確認し、パケット通信可能であれば、ユーザーに i アプリ更新の確認を求めます。パケット通信が不可能な場合やユーザーの同意が得られなかった場合、i アプリはレジュームし、`StarApplicationManager.upgrade()` メソッドは例外をスローすることなく呼び出し元に復帰します。
- (3) (2) でユーザーの同意が得られると、i アプリは終了して i アプリ更新機能が起動し、呼び出し元 i アプリに対応する ADF のダウンロードが行われます。ADF をダウンロードした結果、その i アプリを更新する必要があると判断された場合（ADF の `LastModified` キーが更新されていないなど）は i アプリの更新は行われません。
- (4) JAR ファイルをダウンロードし、携帯電話上の i アプリを更新します。その後、ユーザー確認が必要な ADF 項目（HTTP 通信の有無など）の内容に変更があれば、それらの個別設定画面に移ります。
- (5) JAR ファイルのダウンロードおよび個別設定が完了すると、更新後の i アプリが自動的に再起動されます。フル+ミニアプリがこの機能を使用して更新された場合は、フルアプリとして再起動します。

11.6 通話機能の呼び出し

通話機能の呼び出しでは、i アプリから携帯電話の通話発信機能（ダイヤラー）を呼び出すことができます。

i アプリから通話発信機能を呼び出すには、`com.docomostar.system.Phone` クラスの `call()` メソッドを使用します。このメソッドを、引数に電話番号を表現する文字列を指定して呼び出すことにより通話発信機能が呼び出されます。テレビ電話をサポートしている FOMA 携帯電話では、引数の指定によりテレビ電話の発信を行わせることもできます。また、プッシュトークをサポートしている携帯電話では、呼び出された通話発信機能を操作することによってプッシュトーク発信を行うこともできます。

電話番号文字列には、0～9 までの数字、'#'、'*'、','（ポーズ：1 秒間）、'/'（ポーズ：キー入力待ち）を指定することができます。ポーズ指定は電話番号文字列中に連続して指定することはできません。また、電話番号文字列中に現れる '-'、'('、')'、'.' は無視されます。

なお、通話発信機能を使用する i アプリは、ADF の `UseTelephone` キーで通話発信機能を使用することを宣言（値 `"call"` を設定）する必要があります。この宣言を行っていない i アプリは通話発信機能を使用することはできません。

注意事項：

- `Phone.call()` メソッドを呼び出すと、i アプリはサスペンド状態に入ります。その後、i アプリ実行環境がユーザーに求める発信確認に対しユーザーが同意することによりダイヤル操作が行われます。通話が終了すると i アプリはレジュームし、`call()` メソッドは呼び出し元に復帰します。

【Star-1.1】

通話機能の呼び出しでは、テレビ電話を行うことを示す指示子"**tel-av:**"および **Phone.TEL_AV** が規定されています。しかし、現在の携帯電話はこれらの指示子の有無に関係なく、**Phone.call()**メソッドを呼び出すとユーザーに発信種別（音声通話、テレビ電話、プッシュトーク等）を選択させるダイアログが表示されます。このように、上記指示子は現在には意味を持たないため、これら指示子は **Star-1.1** プロファイル以降では非推奨機能となっています。

11.7 電話帳管理機能の呼び出し

電話帳管理機能の呼び出しでは、i アプリから携帯電話のネイティブの電話帳管理機能呼び出し、ユーザーの同意の下で電話帳エントリ（電話番号、氏名、メールアドレスなどのセット）の新規登録および電話帳グループの新規登録を行なうことができます。

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。

11.7.1 電話帳グループの新規登録

電話帳グループを新規登録するには、**com.docomostar.system.PhoneBookGroup** 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、登録したい電話帳グループの名前または **null** を指定します。

電話帳グループ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 **PhoneBookGroup.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) 電話帳グループ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 アプリからの電話帳グループの新規登録を許可、キャンセルするだけでなく、登録するグループ名を変更することもできます。
- (3) ユーザーが電話帳グループの新規登録を許可するとそのときに設定されていた名前で電話帳グループ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、新規登録された電話帳グループ名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。エントリ ID は、後述する電話帳エントリの新規登録において、電話帳エントリのグループ指定を行うために利用することができます。
ユーザーが電話帳グループ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：

- メーカーによっては電話帳グループの新規登録は、グループの追加ではなく携帯電話にプリセットされたグループの名称変更により行われる場合があります。

11.7.2 電話帳エントリの新規登録

電話帳エントリを新規登録するには、**com.docomostar.system.PhoneBook** クラスの static メソッド **addEntry()** を使用します。このメソッドは、電話帳エントリに登録する情報として以下のものをサポートします。

- ・ 名前
- ・ 名前の読み仮名

- ・ 電話番号（複数指定可能）
- ・ メールアドレス（複数指定可能）
- ・ 登録先の電話帳グループ（グループ名またはグループのエントリ ID を指定）

これらの情報をパラメータとして **addEntry()** メソッドに与えるには、以下の 2 つの方法があります。

- (1) これらの項目を、**addEntry()** メソッドの引数として列挙します。
- (2) 電話帳エントリに指定する各項目をカプセル化する **com.docomostar.system.PhoneBookParam** オブジェクトを作成し、**addEntry()** メソッドの引数に指定します。

電話帳エントリ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 **PhoneBook.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) 電話帳エントリ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 アプリからの電話帳エントリの新規登録を許可、キャンセルするだけでなく、登録するエントリの各項目を変更することもできます。
- (3) ユーザーが電話帳エントリの新規登録を許可するとそのときに設定されていた名前で電話帳エントリ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。

ユーザーが電話帳エントリ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：

- 1 つの電話帳エントリに対して登録可能な電話番号やメールアドレスの数は機種により異なりますが、どの機種でも各々最低 3 つまでは登録可能です。
- 機種によっては、電話帳エントリの名前および読み仮名の項目を姓と名に分けて管理できるものがあります。**PhoneBookParam** オブジェクトを使用しない場合、そのような機種では名前、読み仮名とも姓の方にまとめて設定されます。
PhoneBookParam オブジェクトを使用する場合は、名前や読み仮名を姓と名に分けて設定することができます。
- 電話帳グループ名に **null** を設定すること、および電話帳グループのエントリ ID に -1 を設定することは、登録先の電話帳グループを指定しないことを意味します。この場合、その携帯電話の標準の場所に電話帳エントリが登録されます。
- 電話帳グループのエントリ ID を指定して電話帳エントリの登録を行う際、指定したエントリ ID に対応する電話帳グループが存在しない場合は例外が発生します。これに対して電話帳グループの名前を指定して電話帳エントリの登録を行う際、その名前の電話帳グループが存在しない場合は、i アプリ実行環境はユーザーにその電話帳グループを作成するかどうかを問い合わせます。

11.8 ブックマーク管理機能の呼び出し

ブックマーク管理機能の呼び出しでは、i アプリから携帯電話のネイティブのブックマーク管理機能呼び出し、ユーザーの同意の下でブックマーク（接続先 URL とタイトル）の新規登録を行なうことができます。

なお、ブックマーク管理機能を使用する 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。

ブックマークを新規登録するには、**com.docomostar.system.Bookmark** 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、登録したいブックマークのタイトルおよび接続先 URL を指定します。

ブックマーク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 **Bookmark.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) ブックマーク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 アプリからのブックマークの新規登録を許可、キャンセルするだけでなく、登録する URL やタイトルを変更することもできます。
- (3) ユーザーがブックマーク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ブックマーク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。

ユーザーがブックマーク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：

- ブックマークとして登録可能な URL のスキームは HTTP または HTTPS のみです。

11.9 スケジュール管理機能の呼び出し

スケジュール管理機能の呼び出しでは、i アプリから携帯電話のネイティブのスケジュール管理機能（メーカーによってはカレンダーなど別の名称で呼んでいる場合があります）を呼び出し、ユーザーの同意の下でスケジュール（説明文や日時）の新規登録を行なうことができます。また、i アプリからスケジューラを連携起動することもできます。

なお、スケジュール管理機能を使用する 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。

11.9.1 スケジュールの新規登録

スケジュールを新規登録するには、**com.docomostar.system.Schedule** 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、スケジュールデータの新規登録用パラメータである **ScheduleParam** オブジェクトを指定します。**ScheduleParam** オブジェクトにはスケジュール件名、スケジュール詳細、スケジュールの日時、アラーム ON/OFF を指定することができます。

また、携帯電話のスケジューラが位置情報をサポートしている場合、登録するスケジュールに位置情報を設定することもできます。

日時指定タイプは、このスケジュールが 1 回限りのものか、それとも周期的に繰り返し設定するものであるかを指定するものです。また開始日時と終了日時はそれぞれ、スケジュールに登録する予定時間帯の開始と終了を表します。開始日時と終了日時は **java.util.Calendar** クラスで表現されます。

繰り返し回数と期間の集合は、ともに日時指定タイプに対する補助情報です。繰り返し回数は日時指定タイプが「1 回限り」以外の場合に有効で、その予定時間帯について、日時指定タイプに対応した期間（「毎週」なら 1 週間）を何回繰り返しかを指定します。繰り返し回数が無限であることを表す **ScheduleDate.COUNT_INFINITE** を指定することもできます。

期間の集合は、繰り返し期間の間に複数回のイベントを登録する場合に指定します。例えば、「毎週火曜と木曜」「毎年 3 月と 9 月」などといったスケジュールを表現する場合に使用しま

す。期間の集合は `int` 型の配列で指定します。期間の集合に設定可能な値は、日時指定タイプによって以下のように異なります。

- 日時指定タイプが「1 回限り」および「毎日」
期間の集合は意味を持ちません。設定しても無視されます。
- 日時指定タイプが「毎週」
期間の集合を設定する場合、配列要素には `java.util.Calendar` クラスで定義されている、曜日を表す定数を指定します。
- 日時指定タイプが「毎月」
期間の集合を設定する場合、配列要素には日付を表す整数値（1～31）を指定します。
- 日時指定タイプが「毎年」
期間の集合を設定する場合、配列要素には `java.util.Calendar` クラスで定義されている、月を表す定数を指定します。

以下に、2008 年 10 月 1 日(水)から、毎週水曜日 22:00 から翌日 3:00 までを 5 回繰り返すスケジュールを新規登録する例を示します。

例：スケジュールの新規登録

```
// 開始日時を 2008 年 10 月 1 日 22:00 に設定する。
Calendar start = Calendar.getInstance();
start.set(Calendar.YEAR, 2008);
start.set(Calendar.MONTH, Calendar.OCTOBER);
start.set(Calendar.DATE, 1);
start.set(Calendar.HOUR, 22);
start.set(Calendar.MINUTE, 0);

// 終了日時を翌日（2008 年 10 月 2 日）3:00 に設定する。
Calendar end = Calendar.getInstance();
end.set(Calendar.YEAR, 2008);
end.set(Calendar.MONTH, Calendar.OCTOBER);
end.set(Calendar.DATE, 2);
end.set(Calendar.HOUR, 3);
end.set(Calendar.MINUTE, 0);

// スケジュールの日時を生成する。
ScheduleDate sd = new ScheduleDate(
    ScheduleDate.WEEKLY,
    5, // 5 回繰り返す
    null, // 期間の集合は指定しない
    start,
    end);

// スケジュールデータを生成する。
ScheduleParam sp = new ScheduleParam();
sp.setSummary("予定あり");
sp.setDescription("アルバイト");
sp.setDate(sd);
sp.setAlarm(true);

try {
    // スケジュールを新規登録する。
    Schedule.addEntry(sp);
} catch (InterruptedException e) {
    // 登録操作が異常終了した場合にスローされる
    ...
}
```

上記を実行した場合、携帯電話には以下の日時のスケジュールが登録されます。

- 2008 年 10 月 1 日（水）22:00 ～ 10 月 2 日（木）3:00

- 2008 年 10 月 8 日（水） 22:00 ～ 10 月 9 日（木） 3:00
- 2008 年 10 月 15 日（水） 22:00 ～ 10 月 16 日（木） 3:00
- 2008 年 10 月 22 日（水） 22:00 ～ 10 月 23 日（木） 3:00
- 2008 年 10 月 29 日（水） 22:00 ～ 10 月 30 日（木） 3:00

なお、上記サンプルにおいて、期間の集合に水曜日と土曜日を指定してスケジュールを登録する場合は、**ScheduleDate** オブジェクトを以下のように生成します。

例：期間の集合に水曜日と土曜日を指定する場合

```
ScheduleDate sd = new ScheduleDate(  
    ScheduleDate.WEEKLY,  
    5, // 5 回繰り返す  
    new int[] {Calendar.WEDNESDAY, Calendar.SATURDAY}, // 水曜日と土曜日を指定する  
    start,  
    end);
```

この場合、上記 5 件のスケジュールに加えて、土曜日 22:00 から翌日 3:00 を表す以下のスケジュールが登録されます。

- 2008 年 10 月 4 日（土） 22:00 ～ 10 月 5 日（日） 3:00
- 2008 年 10 月 11 日（土） 22:00 ～ 10 月 12 日（日） 3:00
- 2008 年 10 月 18 日（土） 22:00 ～ 10 月 19 日（日） 3:00
- 2008 年 10 月 25 日（土） 22:00 ～ 10 月 26 日（日） 3:00
- 2008 年 11 月 1 日（土） 22:00 ～ 11 月 2 日（日） 3:00

スケジュール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 **Schedule.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) スケジュール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 アプリからのスケジュールの新規登録を許可、キャンセルするだけでなく、登録する説明文などを変更することもできます。
- (3) ユーザーがスケジュール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スケジュール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。

ユーザーがスケジュール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：

- スケジュールの登録において、すべての日時指定タイプがサポートされています。
- 期間の集合において、どのような日時指定タイプで期間の集合が使用可能かはメーカーにより異なります。どの機種でも共通的に期間の集合が使用可能な日時指定タイプは「毎週」のみです。また、特定の機種でどのような日時指定タイプで期間の集合が使用できるかは **Schedule.getMultiRepeatableTypes()** メソッドを使用して調べることができます。

- 日時指定タイプに「毎週」を指定し期間の集合に含まれない曜日を開始日時に指定した場合、または「毎月」あるいは「毎年」を指定し期間の集合に含まれない開始日時を指定した場合は、スケジュールを登録することはできません。

11.9.2 スケジュールの連携起動

i アプリからスケジュールを起動するには、`Launcher.launch()`メソッドを使用します。スケジュールを起動するには、このメソッドの第1引数（起動対象となるアプリケーションの種別）に `Launcher.LAUNCH_SCHEDULER` を指定します。スケジュール連携起動においては `launch()`メソッドの第2引数（起動引数を示す `String` 配列）は使用しないため、第2引数には `null` を指定してください。

注意事項：

- `Launcher.launch()`メソッドを呼び出すと、i アプリはサスペンド状態に入ります。その後、i アプリ実行環境がユーザーに求めるスケジュール起動確認に対しユーザーが同意することによりスケジュール起動が行われます。スケジュール連携起動では、スケジュールの起動が行われてもi アプリは終了しません（サスペンド状態を保ちます）。ユーザー操作によりスケジュールが終了すると、i アプリはレジュームし `launch()`メソッドは制御をアプリケーションプログラムに戻します。

11.10 画像データ管理機能の呼び出し

携帯電話は、マイピクチャと呼ばれるネイティブの画像管理領域を持っています。画像データ管理機能の呼び出しでは、i アプリから携帯電話のネイティブの画像データ管理機能の呼び出し、マイピクチャ領域からの画像の入出力を行うことができます。画像データ管理機能の呼び出しには、`com.docomostar.system.ImageStore` クラスを使用します。

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 アプリからアクセス可能な画像は以下に限定されます。

- ・ その携帯電話のカメラ機能により撮影・保存された画像
- ・ i アプリにより保存された画像（他の i アプリから保存された画像を含みます。ただしその i アプリが画像を保存する際に、他の i アプリからその画像にアクセスすることを禁じている(*1)場合を除きます。）
- ・ ネイティブの赤外線通信ツールにより、その携帯電話の外部から取り込まれた画像
- ・ 携帯電話出荷時にプレインストールされている画像
- ・ メモリカードなどの外部メモリを搭載している携帯電話では、外部メモリからマイピクチャ領域に移された画像
- ・ ブラウザやメールなどネットワーク経由でダウンロード・保存された画像のうちコンテンツプロバイダにより再配布可能と設定されているもの

(*1) i アプリからネイティブの画像管理領域に画像を保存する際に、他の i アプリからその画像にアクセスすることを禁止することができる API が用意されています。詳細は 11.10.1 項を参照してください。

注意事項：

- 携帯電話ではネイティブの画像データ管理機能で映像データ（i モーション形式データ）を保存、管理できるものがありますが、i アプリから本機能を使用して i モーション形式データを取り扱うことはできません。本機能では、静止画像のみが取り扱い対象となります。

11.10.1 画像データの新規登録

画像データを新規登録するには、**ImageStore** 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、登録したい画像を表す **MediaImage** オブジェクトを指定します。

画像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 **ImageStore.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) 画像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 アプリからの画像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画像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) ユーザーが画像データの新規登録を許可すると画像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、新規登録された画像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。エントリ ID は、後述する画像データの ID 指定読み込みにおいて、ユーザー操作なしに画像データを読み込むために使用することができます。

ユーザーが画像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。

addEntry() メソッドのバリエーションである **addEntry(MediaImage, boolean)** メソッドを使用して画像を保存することで、他の i アプリからその画像にアクセスしたり、携帯電話ネイティブの機能を使用してその画像を携帯電話外部に持ち出すことを禁止することもできます。

注意事項：

- **addEntry()** メソッドに指定する **MediaImage** オブジェクトは、使用可能な状態 (**use()** メソッドが呼び出された状態) でなければなりません。
- 複数の画像を一度の操作 (ユーザー確認等) で保存するための **ImageStore.addEntry(MediaImage[])** は i アプリオプション API に属するメソッドであり、メーカーによってはサポートされない場合があります。
- メーカーによっては、i アプリから以下の両方の条件を満たす画像を新規登録しようとした場合、その画像はデコメ絵文字用の画像フォルダに格納されることがあります。
 - 20x20 ドットの GIF 画像 (アニメーション GIF を含む) もしくは JPEG 画像である。
 - 再配布可能である (再配布不可の設定が行われていない)。
- **ImageStore.addEntry(MediaImage)** メソッドを使用して登録した画像データは、携帯電話ネイティブの機能 (赤外線通信など) により携帯電話の外部に保存することができます。携帯電話の外部に保存されることが好ましくない画像データは、以下のいずれかの方法で保存することを検討してください。
 - マイピクチャ領域ではなく ScratchPad に保存する。
 - **ImageStore.addEntry(MediaImage, boolean)** メソッドを使用してマイピクチャ領域に保存する。
 - **MediaResource.setRedistributable()** メソッドにより明示的に再配布不可と設定した上で、**ImageStore.addEntry(MediaImage)** メソッドを使用してマイピクチャ領域に保存する。
- メーカーによっては Flash コンテンツをネイティブの画像管理領域に保存することをサポートする場合があります。このような機種では、**MediaManager.getImage()** メソッドの引数に Flash コンテンツのデータを指定して呼び出すことで **MediaImage** オブジェクトを取得し、その **MediaImage** オブジェクトを **ImageStore.addEntry()** メソッドの引数に指定することで Flash コンテンツを保存します。なお、Flash コンテンツから生成した **MediaImage** オブジェクトを上記以外の目的 (例えば **VisualPresenter** で再生する) で使用することはできません。メーカーによっては、ネイティブ画像管理領域に保存されている Flash コンテンツを画像データの選択読み込みや画像データの ID 指定読み込

みで取り出すことができますが、取り出した Flash コンテンツの **MediaImage** オブジェクトを表示したり再生することはできません。i アプリから Flash コンテンツを再生するには、i アプリオプション API である i アプリ-Flash 連携機能を使用します。

11.10.2 画像データの選択読み込み

画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画像データ要求（メソッド呼び出し）に対し、その都度画面に表示される画像一覧などからユーザーが選択操作した画像データを与える機能です。画像データの選択読み込みを行うには、**ImageStore** クラスの static メソッド **selectEntry()** を使用します。

画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 **ImageStore.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) 画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画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画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、選択された画像データに対応する **ImageStore** オブジェクトを返します。

ユーザーが画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 **ImageStore** オブジェクトからは、選択された画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 **ImageStore** オブジェクトからは画像データのエントリ ID を取得することもでき、後述する画像データの ID 指定読み込みにおいて、ユーザー操作なしに再度その画像データを読み込むために使用することができます。

11.10.3 画像データの ID 指定読み込み

画像データのエントリ ID は、画像がマイピクチャ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、その画像データが削除または別画像により上書きされるまで変わることはありません。i アプリは画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画像データにアクセスすることができます。画像データの ID 指定読み込みを行うには、**ImageStore** クラスの static メソッド **getEntry()** を使用します。

getEntry() メソッドの引数に画像データのエントリ ID を指定して呼び出すと、その画像データに対応する **ImageStore** オブジェクトが返されます。この **ImageStore** オブジェクトからは、画像データの選択読み込み時と同様に画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。

11.11 カメラ機能の呼び出し

カメラ機能の呼び出しでは、携帯電話に搭載されているカメラデバイスに i アプリからアクセスする手段を提供します。動画撮影に対応している機種では、静止画像だけでなく動画の取り扱いをサポートする場合があります。

カメラ機能の呼び出しには、**com.docomostar.device.Camera** クラスを使用します。

11.11.1 カメラの制御

カメラ制御機能は `com.docomostar.device.Camera` クラスで提供されます。携帯電話には複数のカメラデバイスが搭載されていることがありますが、個々のデバイスはカメラ ID（個々のデバイスに固定的に割り当てられた、0 から始まる整数）により識別されます。アプリケーションプログラムは、`Camera.getCamera()` メソッドにカメラ ID を指定することにより、カメラオブジェクトを取得することができます。

この機能を使用することで、i アプリはカメラオブジェクトに対し以下のような制御を行うことができます。

- ・ カメラデバイスの設定実行
- ・ ネイティブのカメラ機能の起動
- ・ 撮影画像の取得

(A) カメラデバイスの設定実行

カメラデバイスの設定内容には、デバイス属性と撮影画像サイズ、フレーム画像の 3 つがあります。

デバイス属性の設定は、`Camera.setAttribute()` メソッドにより行います。規定されている属性には以下のようなものがあります。

- ・ 連写モード（静止画撮影時のみ有効）
- ・ 画質
- ・ マイク音量（動画撮影時のみ有効）

実際のカメラデバイスがどの属性に対応しているかは機種により異なりますが、アプリケーションプログラムは `Camera.isAvailable()` メソッドにより、その携帯電話で対応している属性を判定することができます。また、`Camera.getAttribute()` により各属性の現在の設定値を知ることができます。

撮影画像サイズの設定には `Camera.setImageSize()` メソッドを使用します。撮影画像サイズは縦横のドット数で表します。指定された撮影画像サイズがその携帯電話でサポートされていない場合、システムは指定されたサイズをサポートしているサイズに補正します。また、フレーム画像の設定には `Camera.setFrameImage()` メソッドを使用します。

カメラデバイスは複数の撮影画像サイズをサポートすることがありますが、その全てのサイズでフレーム付き撮影を行えるとは限りません。カメラデバイスの撮影画像サイズに関する情報は、以下のメソッドを使用することで取得することができます。

- ・ `Camera.getAvailablePictureSizes()` : 撮影可能な静止画像のサイズの取得
- ・ `Camera.getAvailableMovieSizes()` : 撮影可能な動画のサイズの取得
- ・ `Camera.getAvailableFrameSizes()` : フレーム付きで撮影可能な静止画像のサイズの取得

上記に加え、接写モードと通常モードをシステム的に切り替え可能な機種向けに、オプションとして i アプリからフォーカスモードを設定・取得する機能があります。

i アプリからフォーカスモードを設定・取得する機能に対応している機種では、`Camera` クラスの `setFocusMode()` メソッドを使用して、カメラ起動時点で適用されるフォーカスモードを設定することができます。ただしフォーカスモードは、カメラ起動後にユーザー操作で変更することもできます。その場合、ユーザーが設定したフォーカスモードの内容は `Camera` オブジェクトにはフィードバックされません。

また、`getFocusMode()` メソッドで現在設定されているフォーカスモードを確認したり、`getAvailableFocusModes()` メソッドでその機種でサポートされているフォーカスモードのリストを取得することができます。

現在のプロファイルでは、以下のフォーカスモードが規定されています。

- `FOCUS_NORMAL_MODE`（通常モード）
- `FOCUS_MACRO_MODE`（接写モード）
- `FOCUS_HARDWARE_MODE`（ハードウェアのユーザー操作によってのみモード変更可能）

- FOCUS_AUTO_MODE (オートフォーカスモード)
- FOCUS_CONTINUOUS_AUTO_MODE (コンティニユアスオートフォーカスモード)

【Star-1.5】

FOCUS_AUTO_MODE (オートフォーカスモード)、FOCUS_CONTINUOUS_AUTO_MODE (コンティニユアスオートフォーカスモード) は Star-1.5 プロファイルで追加されました。

(B) ネイティブのカメラ機能の起動

アプリケーションプログラムは、`Camera.takePicture()` メソッドを呼び出すことで、静止画撮影モードでカメラ機能を起動することができます。また i アプリからの動画撮影に対応している機種では、`Camera.takeMovie()` メソッドを呼び出すことで動画撮影モードでカメラ機能を起動することもできます。

カメラ機能を起動すると i アプリの実行は中断し、カメラ操作画面がディスプレイに表示されます。実際の撮影はユーザーの操作に基づいてのみ行われ、アプリケーションプログラムがシャッターの操作などを行うことはできません。

ユーザーがカメラ操作画面を終了させると、i アプリの実行が再開されます。カメラ機能終了時にユーザーが撮影画像の保存を行っていた場合、その画像はカメラオブジェクト内に保持されています。

(C) 撮影画像の取得

カメラ機能で撮影された画像は、次にカメラ機能が起動されるか、i アプリが終了するか、または `Camera.disposeImages()` メソッドにより明示的に破棄されるまでカメラオブジェクト内に保持されており、アプリケーションプログラムからメディアイメージまたはバイトストリームの形で参照することができます。

カメラオブジェクトに保持されている画像をメディアリソース (`MediaResource`) の形式で取得するには、`Camera.getImage()` メソッドを使用します。`Camera.getImage()` メソッドにより取得したメディアリソース オブジェクトは `Camera.takePicture()` で撮影した場合はメディアイメージ (`MediaImage`) に、`Camera.takeMovie()` で撮影した場合はメディアムービー (`MediaMovie`) にキャストする必要があります。また、画像をバイトストリーム (静止画像の場合は JPEG フォーマット、動画の場合は i モーションフォーマット) の形式で取得するには、`Camera.getInputStream()` メソッドを使用します。これらのメソッドの引数には、カメラオブジェクト内に保存されている画像のインデックスを指定します。画像のインデックスには通常は 0 を指定しますが、静止画の連写撮影を行っている場合には、インデックスで目的の画像を指定することができます。

11.11.2 撮影画像の携帯電話間での交換

携帯電話の中には、携帯電話ネイティブの機能として、携帯電話間で赤外線ポートを介して撮影画像を交換できる機能を持った機種があります。この機能は OBEX 外部接続機能に基づいて実現されているため、OBEX 外部接続機能を使用する i アプリとこのネイティブアプリケーションの間でも画像の交換を行うことができます。

ネイティブアプリケーションが画像を交換する際には、IrMC vNote バージョン 1.1 に準拠したオブジェクトを使用します。ただし、本機能では画像交換に必要な追加情報を送受信するために vNote のオブジェクトプロパティを拡張しています。

以下に、本機能で使用するオブジェクトプロパティの内容を示します。

項番	プロパティ	値	備考
1	VERSION	1.1	v Note のバージョン
2	X-DOCOMO-TYPE	JPEG または GIF	画像データのフォーマット種別
3	X-DOCOMO-SIZE	<width>X<height>	画像のサイズ (横 X 縦) X はアルファベット半角大文字 (ASCII コード 0x58)

4	X-DOCOMO-FILESIZE	<filesize>byte	画像のファイルサイズ (バイト単位)
5	X-DOCOMO-FILENAME	備考欄参照	ファイル名 (拡張子を含め半角英数で 24 バイト以内の文字列)
6	SUMMARY	備考欄参照	画像タイトル (QUOTED-PRINTABLE エンコーディングを適用した、SHIFT-JIS を含む 18 バイト以内の文字列)
7	DATE	備考欄参照	撮影または編集日時 (UTC 表記)
8	X-DOCOMO-BODY	備考欄参照	Base64 形式でエンコードされた JPEG または GIF データ 本プロパティ値の終端には<CR><LF>を付加してください

以下に、画像データを交換するための vNote オブジェクトの記述例を示します。

```
BEGIN:VNOTE
VERSION:1.1
X-DOCOMO-TYPE:JPEG
X-DOCOMO-SIZE:120X120
X-DOCOMO-FILESIZE:3000byte
X-DOCOMO-FILENAME:picture_001.jpg
SUMMARY;CHARSET=SHIFT_JIS;ENCODING=QUOTED-PRINTABLE:=89=E6=91=9C
DATE:20020715T153530Z
X-DOCOMO-BODY;ENCODING=BASE64:<Base64 形式でエンコーディングされた画像データ本体+<CR><LF>>
END:VNOTE
```

※各行の行末は<CR><LF>(0x0d0a)としてください。X-DOCOMO-BODYについてはデータ末尾の<CR><LF>と合わせて2つの<CR><LF>が連続することになります。

i アプリ側からネイティブアプリケーション側に画像データを送信する場合、ネイティブアプリケーション側は OBEX サーバーとして動作します。i アプリ側は OBEX クライアントとして実装し、画像データ (vNote オブジェクト) を PUT オペレーションで送信するようにします。

またネイティブアプリケーション側から i アプリ側に画像データを送信する場合、ネイティブアプリケーション側は OBEX クライアントとして動作します。i アプリ側は OBEX サーバーとして実装し、ネイティブアプリケーションから PUT オペレーションで送信される画像データ (vNote オブジェクト) を受信するようにします。

注意事項：

- 画像の交換においては、vNote オブジェクトの複数送受信を行うことはできません。画像データは1回のオペレーションで1つだけ送信することができます。
- 現在、i アプリが OBEX 外部接続機能を使用して送受信できるデータのサイズは 100K バイトまでに制限されています。画像の交換においては、上記の vNote オブジェクト全体のサイズが制限以下である必要があります。これは、i アプリ側が画像を送信する場合も受信する場合も同様です。
- X-DOCOMO-FILESIZE プロパティには、Base64 形式にエンコードする前のファイルサイズを指定します。
- i アプリからネイティブアプリケーション側に画像データを送信する際、PUT オペレーションで送信するオブジェクトの Name ヘッダは"mypic.vnt"としてください。
- i アプリからネイティブアプリケーション側に画像データを送信する際、送信可能な画像サイズ (X-DOCOMO-SIZE で示される画像の縦横サイズ) は受信側携帯電話のカメラがサポートする撮影画像サイズに制限されます。

11.11.3 コード認識

コード認識機能では、携帯電話に搭載されたカメラデバイスを使用して画像で表現されたデータ (典型的にはバーコードなど) を取り込み、その内容を認識するための機能を提供します。コード認識機能は、com.docomostar.device.CodeReader クラスで提供されます。

現在のプロファイルでは、識別可能なコードのコード種別として以下を規定しています。

- `CodeReader.CODE_JAN13` (JAN コード標準タイプ)
- `CodeReader.CODE_JAN8` (JAN コード短縮タイプ)
- `CodeReader.CODE_OCR` (テキスト認識)
- `CodeReader.CODE_QR` (QR コード)
- `CodeReader.CODE_39` (CODE-39 コード)
- `CodeReader.CODE_NW7` (NW-7 コード)
- `CodeReader.CODE_128`(Code 128 (JIS X 0504・ISO/IEC 15417))
- `CodeReader.CODE_128_AIM`(Code 128 コード種別の一つで、 AIM Inc. 用に割りあてられたコード)
- `CodeReader.CODE_128_GENERIC`(Code 128 コード種別の一つで、 その他の Code 128 コード)
- `CodeReader.CODE_128_GS1`(Code 128 コード種別の一つで、 GS1-128 (旧称 EAN/UCC-128))

ただし、どの機種でも認識可能なコードは `CODE_JAN13`、`CODE_JAN8`、`CODE_QR`、`CODE_128` の4つです。

コード認識は以下の手順で行います。

- 1) `CodeReader` オブジェクトの取得と設定
- 2) カメラデバイスによるコードの取り込みと認識
- 3) コード認識の結果取得

<CodeReader オブジェクトの取得と設定>

`CodeReader` オブジェクトを取得するには、`CodeReader` クラスの static メソッド `getCodeReader()` を呼び出します。このメソッドの引数には、カメラ制御機能でも使用するカメラ ID を指定します（複数のカメラデバイスを搭載している機種では、コード認識に使用可能な解像度を持つデバイスがどれか1つに制限される場合があります）。

コードの取り込みを行う前には、あらかじめ `CodeReader` オブジェクトに、これから取り込ませようとしているコードのコード種別を設定する必要があります。コード種別の設定は、`CodeReader.setCode()` の引数に、上述のコード種別を指定することで行います。携帯電話各機種で取り扱うことのできるコード種別は、`CodeReader.getAvailableCodes()` メソッドで取得することができます。またメーカーによっては、コード種別に `CodeReader.CODE_AUTO` を指定することにより、コード種別の自動認識を行わせることができるものがあります。

<カメラデバイスによるコードの読み取りと認識>

カメラデバイスを起動してコードの読み取りと認識を行うには、`CodeReader.read()` 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、カメラデバイスの起動が行われます。

ユーザーがカメラデバイスを操作し、撮影およびコード認識処理を行わせることで、撮影されたコードがデータに変換されて `CodeReader` オブジェクトに保持されます。ユーザーが操作をキャンセルしたり、コード認識に失敗した場合は、コードのデータは保持されません。いずれの場合でも、ユーザーがカメラデバイスの操作を終了させると i アプリはレジュームし、アプリケーションプログラムに復帰します。

<コード認識の結果取得>

カメラデバイスによるコードの読み取り・認識に成功すると、アプリケーションプログラムは `CodeReader` オブジェクトからその結果を取得することができます。アプリケーションプログラムは、コード認識の結果として以下の情報を参照することができます。

- 認識したコードのコード種別 (`getResultCode()` メソッド)
- コード認識の結果得られたデータ (`getString()` メソッドまたは `getBytes()` メソッド)
- コード認識の結果得られたデータのデータ型 (`getResultType()` メソッド)

データ型には以下の 4 種類があります。

- ・ 数字文字列 (`CodeReader.TYPE_NUMBER`)
- ・ ASCII 文字列 (`CodeReader.TYPE_ASCII`)
- ・ 非 ASCII 文字列 (`CodeReader.TYPE_STRING`)
- ・ バイナリ (`CodeReader.TYPE_BINARY`)

JAN コードでは規格上の制限により数字文字列のみ取り扱うことができますが、QR コードや Code128 では数字文字列以外のデータも取り扱うことができます。

注意事項：

- コード認識機能はカメラ制御機能と同じカメラデバイスを使用するため、アプリケーションプログラムが同一のカメラ ID でこれらの機能を同時に使用しようとすると、開発者の意図しない結果を引き起こす場合があります。例えば、**Camera** オブジェクトが画像を保持している状態で `CodeReader.read()` メソッドを呼び出すと、**Camera** オブジェクトが保持していた画像は破棄されることがあります。
- JAN コードを読み取る場合、読み取った結果はチェックディジットを含めた全ての文字がアプリケーションプログラムに返されます。つまり、JAN コード標準タイプでは 13 桁、JAN コード短縮タイプでは 8 桁の数字文字列がアプリケーションプログラムに返されます。
- QR コードについては、以下の仕様に従ったものを取り扱うことができます。
 - － バージョン : 1～10 (バージョン 11 以上の使用可否はメーカーにより異なります)
 - － モデル : モデル 2 (モデル 1、および MicroQR はサポートしません)
 - － 誤り訂正レベル : L、M、Q、H
 また、漢字データを扱う際には、i アプリ実行環境のデフォルトエンコーディングである Shift-JIS コードを使用することができます。
- `CodeReader.getResultType()` メソッドにより、読み取ったデータのデータ型がどれだけ詳細に得られるかは、メーカーにより異なります。例えば数字文字列を読み込ませた場合、あるメーカーの携帯電話ではその型を詳細に `CodeReader.TYPE_NUMBER` と判断するかもしれませんが、別のメーカーの携帯電話では文字列の内容判定までは行わず `CodeReader.TYPE_ASCII` や `CodeReader.TYPE_STRING` と判定するかもしれません。このメソッドの返す値は、コード認識の結果得られたデータをアプリケーションプログラムが解析する上でのヒントとして使用するようになっています。

なお、携帯電話ネイティブのコード読み取りアプリケーションには特定の情報を含む 2 次元バーコードを読み取らせることによりその携帯電話内にダウンロードされている特定の i アプリを連携起動する機能が搭載されています。

i アプリを連携起動するための 2 次元バーコードに含まれる情報 (i アプリ起動情報) は、以下の例に示すようなテキストデータです。2 次元バーコード化された i アプリ起動情報をネイティブのコード認識アプリケーションで読み取ると、対応する i アプリを起動する旨の確認画面が表示されます。

< i アプリ起動情報のテキスト例 >

```
LAPL:
ADFURL:http¥://www.nttdocomo.co.jp/java/a.jam;
CMD:startup_Application_A;
PARAM:param1,i-mode;
PARAM:param2,iappli;
;
```

※便宜上折り返し表示を行なっていますが、テキスト内に改行コードは含みません。

以下に、i アプリ起動情報の書式および記法を解説します。

```
LAPL:
ADFURL:<起動対象 i アプリの ADF URL(*1)>;
CMD:<起動コマンド(*1)>;
```

PARAM:<パラメータ名(*2)>, <パラメータ値(*2)>;
: (必要に応じて PARAM を繰り返し)
;

- ※記述例同様、便宜上折り返し表示を行っていますがテキスト内に改行コードは含みません。
- (*1) これらに現れる特殊文字 '¥' ':' ';' には、直前にエスケープ文字 '¥' を付加します。
- (*2) これらに現れる特殊文字 '¥' ':' ';' ', ' には、直前にエスケープ文字 '¥' を付加します。

● LAPL:

i アプリ起動情報の開始を表す識別子です。

● ADFURL:<起動対象 i アプリの ADF URL>;

この i アプリ起動情報で起動させたい i アプリに対応する ADF の URL を ASCII 形式で指定します (最大 255 バイト)。末尾には必ずセミコロンを指定します。起動対象 i アプリは、あらかじめその携帯電話内にダウンロードされている必要があります。

● CMD:<起動コマンド>;

起動コマンドを ASCII 形式で指定します (最大 250 バイト)。末尾には必ずセミコロンを指定します。ここで指定されている起動コマンド (末尾のセミコロンは除きます) と、i アプリの ADF の AllowPushBy キーで起動許可が与えられている起動コマンド (AllowPushBy キーの値として "Code:" 以降に指定されるコマンド文字列) が完全に一致していなければ、i アプリを起動することはできません。ADF の AllowPushBy キーについては、16.5.1 項を参照してください。

● PARAM:<パラメータ名>, <パラメータ値>;

起動された i アプリに引き渡すパラメータを指定します (オプション)。1 つのパラメータは以下のフォーマットを取ります。

PARAM:<パラメータ名>, <パラメータ値>;

※末尾には必ずセミコロンを指定します。

パラメータ名およびパラメータ値は、i アプリ起動時に

StarApplicationManager.getParameter() メソッドにより取得できるパラメータを指定するために使用されます。パラメータ指定は 1 つの i アプリ起動情報に最大 16 個まで指定することができます。ただし、全てのパラメータの名前と値の合計長は 20480 バイトに制限されます。またパラメータの名前および値には ASCII 文字のみ使用可能です。

● ;

i アプリ起動情報の終了を表す識別子です。開始識別子 (LAPL:) と対となります。

これらの規定に従って作成した i アプリ起動情報のテキストを 2 次元バーコードに変換する方法については、2 次元バーコード編集のための各ツールのマニュアル等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 2 次元バーコードからの i アプリ起動を禁止することができます。2 次元バーコードからの起動が禁止されている場合、2 次元バーコードを読み取らせても i アプリの起動は行われません。
- 2 次元バーコードから起動された i アプリの動作が終了すると、i アプリ起動前の状態 (コード認識結果表示) に復帰します。ただし、2 次元バーコードから起動された i アプリをユーザーが強制終了した場合、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。
- 2 次元バーコードは人間が視覚的に直接解読することは困難ですが、対応機器 (読み取り専用機器だけでなく 2 次元バーコード対応の一般的な携帯電話なども含みます) を介することで容易に読み取ることができます。機密の保持が求められるようなデータは、i アプリ起動情報の起動コマンドやパラメータには設定しないなどの配慮が必要です。
- 起動先 i アプリは、フルのみアプリか、フル+ミニアプリでなければなりません。また、起動先 i アプリが、フル+ミニアプリである場合は、フルアプリが起動されます。

11.12 映像データ管理機能の呼び出し

映像データ管理機能の呼び出しでは、i アプリから携帯電話ネイティブの映像データ管理機能呼び出し、映像データ（i モーション）をネイティブ保存領域に新規登録することができます。

なお、本機能を使用する 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。

映像データをネイティブ保存領域に新規登録するには、**com.docomostar.system.MovieStore** 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、登録したい映像データ（i モーション形式データ）を表す **MediaMovie** オブジェクトを指定します。

映像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 **MovieStore.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) 映像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 アプリからの映像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映像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) ユーザーが映像データの新規登録を許可すると映像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、新規登録された映像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。ネイティブ保存領域からの映像データ取得をサポートする機種では、エントリ ID は映像データ取得のキーとして使用することができます。

ユーザーが映像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。

注意事項：

- **addEntry()** メソッドに指定する **MediaMovie** オブジェクトは、使用可能な状態（**use()** メソッドが呼び出された状態）でなければなりません。またこの **MediaMovie** オブジェクトは、i モーション形式の映像データから生成されたものでなければなりません。
- 映像データから生成した **MediaMovie** オブジェクトを **MovieStore** クラスで保存する際には、元の映像データに設定されていた再配布可否情報もそのままの形で保存されます。また、**MediaResource.setRedistributable()** メソッドを使用して、i アプリから再配布可否情報の設定を行うこともできます。
- **MovieStore** クラスには **addEntry()** メソッドの他に、**selectEntry()** メソッドや **getEntry()** メソッドなど、ネイティブ保存領域の映像データを取得する API が定義されています。しかしこれらは i アプリオプション API に属する機能であり、これらをサポートするかどうかはメーカーにより異なります。

11.13 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.docomostar.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、11.9 項で解説したスケジュール管理機能の呼び出しにおける **ScheduleDate** クラスの使用法と同様です。

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：

- 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。
- 期間の集合の利用可否、およびどのような日時指定タイプで期間の集合が使用可能かはメーカーにより異なります。特定の機種でどのような日時指定タイプで期間の集合が使用できるかは **Alarm.getMultiRepeatableTypes()** メソッドを使用して調べることができます。

11.14 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。本章で解説してい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。

データ BOX フォルダ管理機能では、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 が定義されています。（現在のプロファイルでは、データ BOX フォルダ管理機能がサポートするデータ保存領域種別はマイピクチャフォルダのみです。）

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。

11.14.1 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.docomostar.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はマイピクチャフォルダ (`DataBoxFolder.FOLDER_MY_PICTURE`) のみです。

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：

- 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、`ImageStore.getEntryIds()` メソッド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画像データのエントリ ID を一括して取得することができます。

なお、画像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。

11.14.2 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：

- 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、`ImageStore.getEntryIds()`メソッド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画像データのエントリ ID を一括して取得することができます。

なお、画像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。

11.15 コンテンツ取得保存機能の呼び出し

Star-1.1 プロファイル以降の携帯電話では、コンテンツ取得保存機能の呼び出しが行えます。

コンテンツ取得保存機能の呼び出しでは、i アプリから携帯電話ネイティブのコンテンツ取得保存機能呼び出し、従来の i アプリではヒープメモリの制限や通信機能の制限などによりダウンロードさせることが困難であったフル楽曲などの大容量コンテンツをダウンロードして、携帯電話ネイティブの保存領域に直接保存することができます。

本機能を使用する 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。

コンテンツ取得保存機能で、どの機種でも共通的に取り扱うことのできるコンテンツはフル楽曲コンテンツです。

メーカーによっては、コンテンツ取得保存機能における以下のようなオプション機能をサポートする場合があります

- i モーションコンテンツの取得保存
- ダウンロードしたコンテンツの、コンテンツ取得保存機能上での再生（試聴）
- ダウンロードしたコンテンツの外部メモリ(SD カードなど)への保存

コンテンツ取得保存機能の呼び出しは、`com.docomostar.system.ContentsDownloader` クラスのコンストラクタにダウンロードするコンテンツの URL を指定してオブジェクトを生成し、`download()`メソッドを呼び出すことで行います。なお、コンストラクタに指定するダウンロード URL にクエリーを付加することはできません。ダウンロード URL にクエリーを指定する場合は、後述の `setQuery()`メソッドを使用します。

`download()`メソッドを呼び出すまでの間に、以下のメソッドを呼び出すことでコンテンツ取得保存機能の動作に関する設定を行うことができます。

- `setQuery()`
コンテンツダウンロード時に送出するクエリーを指定します。`application/x-www-form-urlencoded` 形式でエンコードされた文字列を指定してください。
ここで指定された文字列は、リクエストメソッドが GET である場合はリクエストラインの一部として、POST である場合にはボディとして、ダウンロード元サーバーへ送信されます。
本メソッドが呼び出されない場合、クエリーなしとしてリクエストが送出されます。
- `setExpectedContentType ()`
ダウンロードしたいコンテンツのコンテンツタイプを指定します。サーバーは、コンテンツ取得のためのリクエストに対し、ここで指定されたコンテンツタイプと同じ値を `Content-Type` レスポンスヘッダで返す必要があります。現在のプロファイルで指定可能な値は、

- application/x-dcm-fullmusic (フル楽曲)
- video/3gpp (i モーション)
- audio/3gpp (i モーション)

となります。上記以外の値が指定された場合は、**UnsupportedOperationException** の例外が発生します。また、i モーションの取得保存をサポートしていない携帯電話では video/3gpp および audio/3gpp が指定された場合も **UnsupportedOperationException** の例外が発生します。

本メソッドが呼び出されない場合、"application/x-dcm-fullmusic"が指定されたものとして扱われます。

- **setRequestMethod()**

ダウンロード時に用いるリクエストメソッドを指定します。**HttpConnection.GET** あるいは **HttpConnection.POST** を指定します。これら以外が指定された場合、**IllegalArgumentException** の例外が発生します。

本メソッドが呼び出されない場合、**HttpConnection.GET** が指定されたものとして扱われます。

- **setPlayOptions()**

コンテンツ再生時の振る舞いを決定するオプションパラメータを設定します。本メソッドを使用する際は、設定したいオプションパラメータの名前と値を文字列として **Hashtable** オブジェクトに登録し、その **Hashtable** オブジェクトをメソッドの引数に指定します。

オプションパラメータは、ダウンロードしたコンテンツのコンテンツ取得保存機能上での再生（試聴）をサポートしていない機種でも有効です。ここで設定した内容は、後刻このコンテンツをネイティブのプレーヤーで再生する場合にも利用されます。

本メソッドで設定可能なパラメータは以下の通りです。

番号	パラメータ名	パラメータ概要
①	count	コンテンツの再生回数制限の再生回数を、0～15 までの数字文字列として設定します。本パラメータは、他のパラメータと組み合わせて使用することはできません。
②	period	コンテンツの再生期限制限の再生期限を、yyyy/mm/dd/hh/mm の形式の日時文字列として設定します。ここで指定する日時は、GMT として解釈されます。本パラメータは、他のパラメータと組み合わせて使用することはできません。
③	span	コンテンツの再生期間制限の再生期間（日数）を、1～999 までの数字文字列として設定します。本パラメータは、他のパラメータと組み合わせずに使用するか、もしくは start パラメータとのみ組み合わせて使用することができます。
④	start	コンテンツの再生期間制限の再生可能開始日を、yyyy/mm/dd/hh/mm の形式の日時文字列として設定します。ここで指定する日時は、GMT として解釈されます。このパラメータは、 span パラメータと組み合わせてのみ使用することができます。

コンテンツ取得保存機能の呼び出しを行うと、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 **ContentsDownloader.download()** 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) コンテンツ取得保存を行うためにネイティブのコンテンツ取得保存機能呼び出すことをユーザーに確認する画面が表示されます。
- (3) ユーザーがネイティブ機能呼び出すことを許可するとコンテンツのダウンロードが開始されます。
- (4) コンテンツのダウンロードが完了すると、ダウンロードしたコンテンツの種別に応じてコンテンツメニューが表示されます。コンテンツメニューでは、コンテンツの保存や情報表示が行えるほか、コンテンツの再生（試聴）を行える機種もあります。

- (5) コンテンツメニューで「保存」を選択すると i アプリはレジュームして `download()` メソッドはアプリケーションプログラムに復帰します。

注意事項：

- コンテンツ取得保存機能を利用する場合、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。また、`UseNetwork` キーを `"yes"` に設定しておく必要があります。
- コンテンツをダウンロードするサーバーは、i アプリから HTTP 通信を行うことができるサーバーでなければなりません。つまり、i アプリダウンロード元サイトと、スキーム、ホスト名、ポート番号が同じである必要があります。
- `setQuery()` メソッドに指定できる文字列は、リクエストメソッドが GET、POST のいずれかにかかわらず、どの機種でも少なくとも長さ 512 バイトまでは指定することができます。512 バイトを超える文字列を指定できるかどうかはメーカーにより異なります。
- フル楽曲サブスクリプション機能はサポートしません。
- `ContentsDownloader.download()` メソッドでなんらかの例外が発生した場合、取得したコンテンツは破棄されます。
- レンジリクエストによる大容量コンテンツダウンロードでは、ダウンロード中に通信に失敗して一部しかダウンロードできなかった場合でもそれを保存することができます。そのようなコンテンツは、別途ユーザーがネイティブのコンテンツ管理機能を操作して、残りの部分のダウンロードを行なうことができます。
- 本機能は、HTTP リダイレクトには対応していません。`ContentsDownloader` クラスのコンストラクタに指定する URL には、HTTP リダイレクトを行わない URL を指定して下さい。
- オプションパラメータで再生期限や再生期間を設定する場合、本機能使用時点でそれら期限や期間が過ぎ去ってはいけません。

【Star-1.1】

本機能は、Star-1.1 プロファイルで新設されました。

第 12 章

赤外線リモコン

携帯電話の赤外線ポートから一般的な赤外線リモコン対応機器に対して制御信号を送出するための API を利用して、i アプリから適切な制御信号を組み立てて送出的ることにより、それらの赤外線リモコン対応機器を外部から操作することができます。

注意事項：

- i アプリの赤外線リモコン機能では、赤外線信号の物理的な波形を組み立てて送出的ためのプリミティブな API が規定されています。一方、一般的な赤外線リモコン対応機器では、具体的にどのような波形を制御信号として受け付けるかは対応機器のメーカーが独自に設計しています。i アプリ仕様の範囲には、赤外線リモコンの制御信号の仮想化・標準化などは含みません。開発者は、ターゲットとする赤外線リモコン対応機器の仕様を確認の上、それに適した制御信号を送出するよう i アプリを設計する必要があります。
- i アプリの赤外線リモコン機能では、携帯電話からおおむね 2～3 メートル程度離れた場所にある機器を制御可能とすることを想定しています。ただし、実際にどの程度離れた機器を制御できるかは、携帯電話からの信号の出力や指向性だけでなく、周囲の環境や赤外線リモコン対応機器側での受光能力などにより影響を受けます。

12.1 制御信号の構成

i アプリの赤外線リモコン機能では、以下に示すような赤外線リモコンの制御信号を取り扱うことができます。日本国内で販売されている一般的な家電製品の多くは、制御信号を下記の枠組みで表現することができます。ただし、前述の通り制御信号の設計はその機器のメーカーが独自に行っているため、下記内容が全ての赤外線リモコン対応機器に適用されることを保証するものではありません。

キャリア周波数と変調方式：

赤外線リモコンの制御信号はキャリア周波数 25kHz～50kHz の搬送波（赤外線発光の ON と OFF で表現される矩形波）で搬送されるパルス位相変調信号（PPM 信号 以下、搬送波で搬送される論理的な波形を制御信号とします）として取り扱われます。制御信号の High 区間と Low 区間の長さの組み合わせによってビット値（論理 0 および論理 1）を表現します。

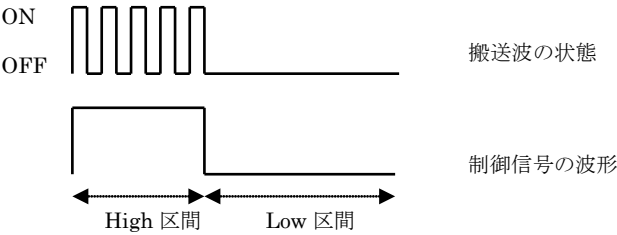


図 23 搬送波による制御信号の構成

ビット値の制御信号波形上での表現方法には以下の 2 つのパターンがあり、どちらの方法を取るかは赤外線リモコン対応機器側の設計により異なります。i アプリの赤外線リモコン機能では、下記のいずれの方法を使用するかを指定することができます(波形図の二重線部の長さで論理 0 と論理 1 が識別されます)。

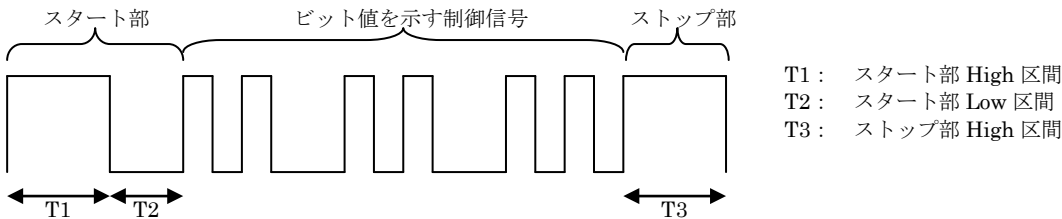
	論理 0 の波形の例	論理 1 の波形の例
High 先行パターン Low 区間の長さにより論理 0、論理 1 を表現		
Low 先行パターン High 区間の長さにより論理 0、論理 1 を表現		

上記において、以下の項目は赤外線リモコン対応機器の設計に依存しており、アプリケーションプログラムはこれらを機器に適した値に設定することができます。

- 搬送波 1 波形あたりの High 区間および Low 区間の長さ (これにより搬送波のキャリア周波数が決定付けられます)
- ビット値の表現方法 (High 先行パターンまたは Low 先行パターン)
- 制御信号上の、論理 0 の High 区間の長さおよび Low 区間の長さ
- 制御信号上の、論理 1 の High 区間の長さおよび Low 区間の長さ

データフレームの構成 :

赤外線リモコンの送出データは、上述の方法で変調されたビットデータの制御信号にスタート部 (一定の長さを持つ High 区間と Low 区間各々 1 つずつの制御信号の組み合わせ) およびストップ部 (一定の長さを持つ 1 つの High 区間から構成される制御信号) を付加したものとなります。



図中のビットデータは High 先行パターンの例 :

┐ を論理 0、┐ を論理 1 とすると、上記のデータフレームには 0 1 0 1 0 0 という 6 ビットのビット列が含まれることになります。

図 24 データフレームの構成

スタート部の High 区間と Low 区間、およびストップ部の High 区間の長さは赤外線リモコン対応機器の設計に依存しており、アプリケーションプログラムはこれらを機器に適した値に設定することができます。

また、1つのデータフレームに含めることのできるビットデータのビット数には、プロファイルバージョンごとに制限があります。データフレームは、12.2.2 項で述べる `com.docomostar.device.IrRemoteControlFrame` クラスで表現されます。データフレーム中のビット数の制限についても 12.2.2 項を参照してください。

赤外線リモコン送出データの全体構成：

赤外線リモコンの送出データは、データフレームの集合として表現されます。全く別のデータフレームが連続して送られることも、あるデータフレームが繰り返し送られることもあります。

また一般に赤外線リモコンは、規定されたデータフレームを一通り送信し終わると、再度先頭のデータフレームから送信を繰り返します。これは、例えばテレビなどのリモコンの音量ボタンを押し続けると、その間制御信号が送出され続けることと対応しています。

データフレームの繰り返し送信は、データ送出時に指定した時間（タイムアウト時間）または繰り返し回数が経過するか、アプリケーションプログラムが明示的にデータ送出をストップするまで行われます。

赤外線リモコンの送出データとして以下を指定した場合の創出例：

- ・ データフレーム A × 1
- ・ データフレーム B × 2
- ・ データフレーム C × 1

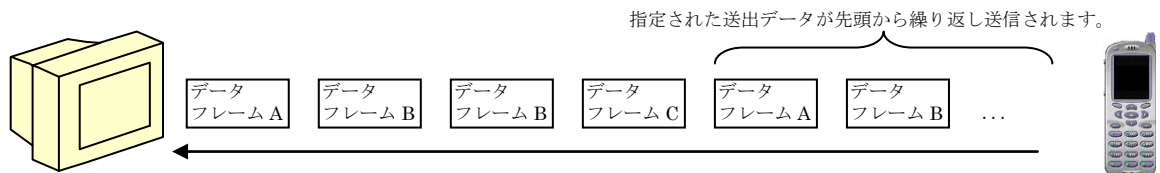


図 25 携帯電話からのデータフレームの送出例

アプリケーションプログラムは、作成したデータフレーム情報の送出順序、送出間隔、データ送出を終えるまでのタイムアウト時間または繰り返し回数を、ターゲットとする機器に適した内容に設定することができます。赤外線リモコン送出データは、12.2.1 項で述べる `com.docomostar.device.IrRemoteControl` クラスを使用して送出します。

12.2 赤外線リモコン API

前項に記載した赤外線リモコンの機能を i アプリから利用するためのクラスとして、以下のものがあります。

- ・ `com.docomostar.device.IrRemoteControl`
- ・ `com.docomostar.device.IrRemoteControlFrame`

`IrRemoteControl` クラスは赤外線ポートから制御信号を送信するための送出器としての役割を持ちます。また `IrRemoteControlFrame` クラスは、送出データを構成するデータフレームを表現します。アプリケーションプログラムは、`IrRemoteControlFrame` オブジェクトの形でデータフレームのセットを作成し、そのセットを `IrRemoteControl` オブジェクトにより赤外線ポートから送出します。

12.2.1 IrRemoteControl

`IrRemoteControl` クラスは、`IrRemoteControlFrame` の形で指定されたデータフレームを送出する機能を持ちます。また、リモコン機能全体に渡るコンフィグレーションを行うためのメソッドを備えています。

アプリケーションプログラムは、以下の手順で `IrRemoteControl` クラスを使用します。

- (1) `IrRemoteControl.getIrRemoteControl()` メソッドを使用して、`IrRemoteControl` オブジェクトを取得します。`IrRemoteControl` オブジェクトは携帯電話の赤外線リモコンポートと 1 対 1 に対応付けられており、アプリケーションプログラム中で複数のオブジェクトを持つことはできません。
- (2) 赤外線リモコンの物理的なコンフィグレーションを行います。赤外線リモコンの設定項目には以下のものがあります。
- ・ 搬送波の High 区間長、Low 区間長（キャリア周波数）の設定（`setCarrier()` メソッド）
 - ・ 制御信号上の論理 0 波形の表現パターンと High 区間長、Low 区間長（`setCode0()` メソッド）
 - ・ 制御信号上の論理 1 波形の表現パターンと High 区間長、Low 区間長（`setCode1()` メソッド）
- 論理 0 波形と論理 1 波形の表現パターンには、`IrRemoteControl` クラスの定数 `PATTERN_HL`（High 先行）または `PATTERN_LH`（Low 先行）を指定します。
- (3) 送信したいデータフレームを表す `IrRemoteControlFrame` オブジェクトのセットを生成し、`send()` メソッドを使用して送信します。`send()` メソッドでは、データフレームのセットを繰り返し自動送信する時間（タイムアウト時間）、もしくはデータフレームのセットを送出する際の繰り返し回数を指定することができます。デフォルトのタイムアウト時間は 10 秒です。
- `send()` メソッドは、データフレームの送信要求を受け付けた後すぐにアプリケーションプログラムに復帰し、その後アプリケーションプログラムとは非同期にデータフレームの送信が行われます。データフレームの送信は、以下のいずれかの契機まで繰り返し続けられます。
- ・ タイムアウト時間の経過、もしくは指定された繰り返し回数分のデータフレームセットの送信完了
 - ・ アプリケーションプログラムからの明示的な送信停止指示（`stop()` メソッド）
 - ・ 通話着信の発生などによる中断
 - ・ アプリケーションプログラムからの HTTP 通信の実行
 - ・ アプリケーションプログラムの終了（異常終了も含みます）

注意事項：

- 以下の場合には、アプリケーションプログラムは `send()` メソッドを呼び出すことはできません。
 - ・ 携帯電話がセルフモードに設定されている場合
 - ・ i アプリが赤外線ポートを使用している状態（赤外線リモコン、OBEX 外部接続）
 - ・ パケット通信中の場合
- 赤外線リモコンの設定において、設定値の単位や設定可能な値の範囲はメソッドにより異なっています。メソッド毎の設定値の単位などについては API リファレンスを参照してください。
- メーカーの採用する赤外線デバイスの特性によっては、API で指定した波形パラメータ（High 区間長、Low 区間長）の値がそのデバイスで取り扱うことのできる値（単位）に補正されることがあります。

12.2.2 IrRemoteControlFrame

`IrRemoteControlFrame` クラスは、個々のデータフレームを表現します。アプリケーションプログラムはこのクラスのコンストラクタを使用してオブジェクトを生成し、スタート部、ビットデータ部、ストップ部を設定することでデータフレームを構成します。

データフレームの設定項目には以下のものがあります。

- ・ データフレームの送信間隔（`setFrameDuration()` メソッド）
そのデータフレームの送信開始時点から、次のデータフレームの送信を開始するまでの間隔を指定します。
- ・ データフレームの繰り返し送信回数（`setRepeatCount()` メソッド）
1 つのデータフレームセットの中で、同じデータフレームを連続して送信する場合は、その回数を指定します。同じデータフレームを連続して送信しない場合は 1 を指定します。
- ・ スタート部の形式（`setStartHighDurationin()` および `setStartLowDurationin()` メソッド）
そのデータフレームにおけるスタート部の High 区間および Low 区間の長さを指定します。

- ・ ストップ部の形式 (`setStopHighDuratooin()` メソッド)
そのデータフレームにおけるストップ部の High 区間の長さを指定します。
- ・ ビットデータの値 (`setFrameData()` メソッド)
`byte` 配列または `long` 値の形で、ビットデータとして送出する値を指定します。

なお、どの機種でも送信することのできるビットデータ長は、1 フレームあたり 1024 ビットです。

注意事項：

- データフレームの設定において、設定値の単位や設定可能な値の範囲はメソッドにより異なります。メソッド毎の設定値の単位などについては API リファレンスを参照してください。
- メーカーの採用する赤外線デバイスの特性によっては、API で指定したスタート部、ストップ部およびデータフレーム送信間隔に関するパラメータの値がそのデバイスで取り扱うことのできる値（単位）に補正されることがあります。

第 13 章

3D グラフィックス・3D サウンド

本章では、3D グラフィックス描画機能、3D サウンド制御機能の概要について解説します。なお、本プロファイルに対応した携帯電話では、標準の 3D グラフィックスエンジンとして株式会社エイチアイの Mascot Capsule Engine Micro3D Edition のバージョン 4 が搭載されています。このエンジンを活用した i アプリプログラミングの詳細については、株式会社エイチアイより提供されるドキュメント「Micro3D Programming for i アプリ」を参照してください。

注意事項：

- DoJa プロファイルでは、3D グラフィックス・3D サウンド機能は LE 仕様におけるオプション機能となっています。Star プロファイルでは、LE 仕様は規定されていませんが、一部の機種で本機能が搭載されていない場合があります。

13.1 3D グラフィックス描画機能

13.1.1 3D グラフィックス描画機能

3D グラフィックス描画機能は、比較的簡単な API 呼び出しで 3D モデルの描画を行うことができます。

3D グラフィックス描画機能は、`com.docomostar.ui.graphics3d` パッケージで提供されます。また、3D に関する数値演算のために `com.docomostar.ui.util3d` パッケージで提供されるユーティリティ機能を使用します。

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。

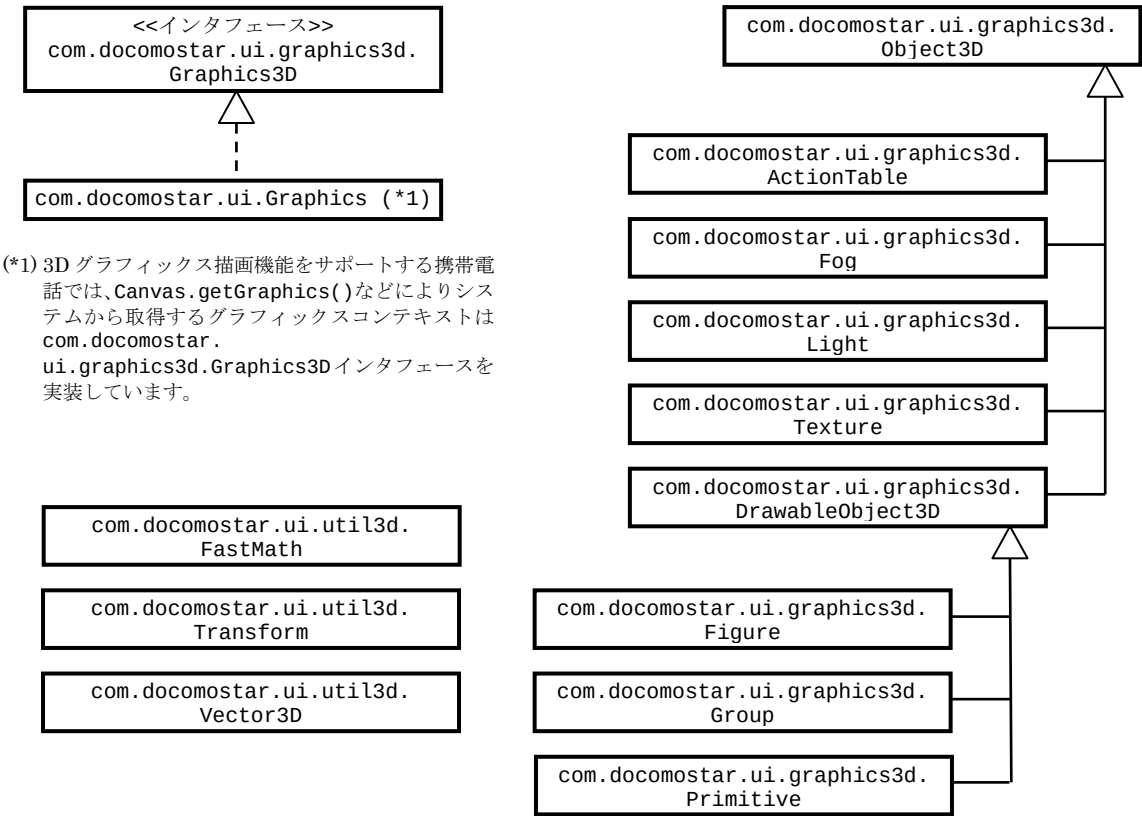


図 26 3D グラフィックス描画機能のクラス構成

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	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。Image に対応するグラフィックスコンテキストがこのインタフェースをサポートするかどうかはメーカーにより異なります。
Object3D	全ての 3D オブジェクトの基底となるクラスです。3D オブジェクトとは、3D グラフィックスのレンダリング結果に影響を与えるオブジェクト全般を指します。3D オブジェクトには、フィギュアやプリミティブのように直接レンダリングの対象物とすることのできるオブジェクトと、ライトやフォグのようにレンダリング結果に効果を与える性質を持つオブジェクトに大別されます。このクラスでは 3D オブジェクトの生成方法として createInstance() を規定していますが、3D オブジェクトの種類によってはコンストラクタを使用してオブジェクトを生成するものもあります。
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。フレームは全ての 3D オブジェクトが内部に保持している「現在時刻」と対応しており、後述の Figure オブジェクトに現在時刻を設定することで、3D モデルにポーズを取らせることができます。
Fog	フォグ効果を与えるためのデータを保持するクラスです。線形フォグモードと指数フォグモードをサポートします。
Light	光源データを保持するクラスです。いくつまでの光源をサポートするかはメーカーにより異なります (getMaxLights()) メソッドで確認することができます)。環境光源、平行光源、点光源、スポット光源をサポートします。

Texture	テクスチャのデータを保持するクラスです。テクスチャオブジェクトを使用することで、モデルマッピングに使用するテクスチャと環境マッピングに使用するテクスチャの両方を取り扱うことができます。なお、本機能における環境マッピング用テクスチャの使用方法は i アプリ拡張 API の高レベル 3D グラフィックス描画機能における使用方法から変更されています。本機能では、環境マッピング用のテクスチャは 3D グラフィックスコンテキスト (Graphics3D オブジェクト) ではなく各モデルマッピング用テクスチャオブジェクトに設定して使用します。
DrawableObject3D	レンダリング可能な 3D オブジェクトの基底となるクラスです。このクラスを継承する Figure クラス、Group クラス、Primitive クラスの各オブジェクトは、Graphics3D.renderObject3D() メソッドによるレンダリング対象とすることができます。また、3D 空間内で 2 つのオブジェクト同士が衝突しているかどうかを簡易に判定するためのメソッドや、ブレンド機能、テクスチャ歪み補正機能などのメソッドを備えています。
Figure	3D モデルのデータを保持するクラスです。このオブジェクトはレンダリング可能であり、Graphics3D.renderObject3D() メソッドにレンダリング対象として指定することができます。
Group	3D オブジェクトの集合体であるグループを表現するクラスです。このオブジェクトはレンダリング可能であり、Graphics3D.renderObject3D() メソッドにレンダリング対象として指定することができます。複数の Figure オブジェクトや Primitive オブジェクトを属させて一度にレンダリングを行わせたり、Fog オブジェクトや Light オブジェクトを属させてグループ内のレンダリングに効果を与えたりすることができます。
Primitive	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などを格納するための、プリミティブ配列を保持するクラスです。このオブジェクトはレンダリング可能であり、Graphics3D.renderObject3D() メソッドにレンダリング対象として指定することができます。
FastMath	3D 関連機能での使用を前提とした高速数値演算ユーティリティです。本プロファイルに対応する携帯電話の i アプリ実行環境では浮動小数点数演算が使用できますが、このクラスでは浮動小数点数演算を内部的に整数演算に置き換えます。このため、通常の浮動小数点数演算より大きな誤差を含む可能性はありますが高速に演算を行うことができます。
Transform	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。
Vector3D	3 次元ベクトルを表すクラスです。内積・外積計算、正規化といったユーティリティメソッドを備えています。

注意事項：

- 本プロファイルに対応した携帯電話では、標準の 3D グラフィックスエンジンとして株式会社エイチアイの Mascot Capsule Engine Micro3D Edition のバージョン 4 が搭載されています。このエンジンを活用した i アプリプログラミングの詳細については、株式会社エイチアイより提供されるドキュメント「Micro3D Programming for i アプリ」を参照してください。
- i アプリ拡張 API の高レベル 3D グラフィックス描画機能 (com.docomostar.opt.ui.j3d パッケージ) と本機能を 1 つの i アプリの中で同時に使用することはできません。一方を使用中に他方を使用しようとすると例外がスローされます。
- 本機能では、Java 言語の浮動小数点型を使用したインタフェースを使用しています。ただしシステム内部の演算では、性能を確保するために整数型を使用した演算が行われます。
- 携帯電話の中には、3D グラフィックス関連の処理を高速化するため専用のハードウェアを搭載しているものがあります。そのような機種では、ハードウェア実装の制限から他の機種やエミュレータと比較して演算精度が落ちる場合があります。特に、Z 方向の位置関係の演算 (デプス・バッファ処理) にこの問題が作用した場合、ごく似通った前後関係のオブジェクト同士が意図通りの前後関係で描画されないことがありますので注意してください。

13.2 3D サウンド制御機能

本プロファイルでは、携帯電話に搭載されたステレオスピーカーを応用して、左右だけでなく上方や後方などさまざまな方向から立体的にサウンドが聴こえるような効果を与えることのできる 3D サウンド制御機能を定義しています。この機能を使用することにより、聴者と仮想音源の位置関係 (定位と呼びます) を指定してユーザーに特定の方向から音が聴こえるように感

じさせたり、時間の経過とともに定位を変更して仮想音源が音を出しながら移動しているように感じさせたりすることができます。

本項では、3D サウンド制御 API の使用方法について解説します。

【Star-2.0】

Star-2.0 プロファイル以降、3D サウンド制御機能はオプションになりました。3D サウンド制御機能をサポートしない機種で 3D サウンド制御 API を利用した際の振る舞いは 13.2.4 項を参照してください。

13.2.1 3D サウンド制御機能のクラス構成

3D サウンド制御機能は、`com.docomostar.ui` パッケージおよび `com.docomostar.ui.sound3d` パッケージで提供されます。また 3D グラフィックス描画機能と同様、3D に関する数値演算のために `com.docomostar.ui.util3d` パッケージで提供されるユーティリティ機能を使用します。

3D サウンド制御機能を構成する、または関連するクラス群を以下に示します。

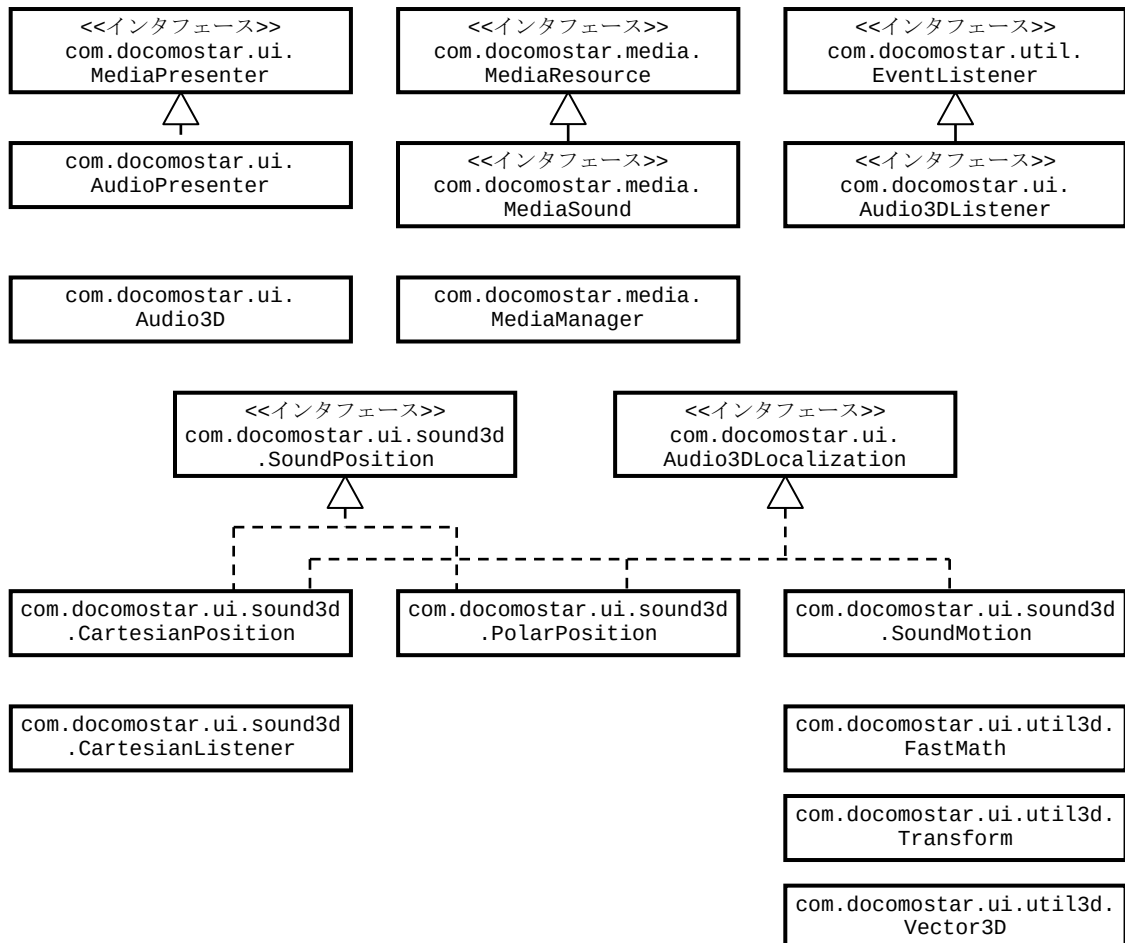


図 27 3D サウンド制御機能のクラス構成

以下に 3D サウンド制御機能を構成する各クラス、インタフェースの機能概要を示します。

クラス・ インタフェース	機能概要
AudioPresenter	オーディオプレゼンタです。3D サウンド制御ではオーディオプレゼンタが仮想音源に相当し、各オーディオプレゼンタに対応付けられたコントローラ (Audio3D オブジェクト) によって定位 (聴者と仮想音源の位置関係) を制御します。本プロファイルでは、AudioPresenter クラスにはそのプレゼンタに対応する Audio3D オブジェクトを取り出すための getAudio3D() メソッドが用意されています。
Audio3D	各オーディオプレゼンタ毎に 3D サウンド制御を行うためのコントローラです。Audio3D オブジェクトを使用して 3D サウンド制御の有効/無効を切り替えたり、定位の制御を行います。i アプリの中でいくつの仮想音源を使用できるか (3D サウンド制御リソース数) はメーカーにより異なりますが、どの機種でも利用可能な 3D サウンド制御リソース数は 4 つです。i アプリはこの制限の範囲内で Audio3D オブジェクトを使用することができます。
MediaSound	サウンドデータを表現するメディアデータです。3D サウンド制御では i アプリから定位を指示することで 3D 制御を行う方法の他に、あらかじめサウンドデータに定位情報を埋め込んでおき、サウンド再生時にそれを使用して自動的に 3D 制御を行う方法を取ることができます。サウンドデータに定位情報を埋め込む場合は 1 つのサウンドに複数の 3D サウンド制御リソースを割り当てることができます。本プロファイルの MediaSound クラスでは、対応するサウンドデータに埋め込まれた定位情報にいくつの 3D サウンド制御リソースが割り当てられているかを取得する機能が用意されています。
Audio3DLocalization	定位を表すインタフェースです。3D サウンド制御機能では定位を表現する方法 (クラス) が複数ありますが、どのクラスもこのインタフェースを実装しています。 なお、このインタフェースではメソッドやフィールドは定義されていません。
SoundPosition	聴者の位置と仮想音源の位置によって表される、固定された定位を表すインタフェースです。このインタフェースを実装したクラスでは、聴者の位置と仮想音源の位置を指定することによって両者の位置関係を表します。 なお、このインタフェースではメソッドやフィールドは定義されていません。
PolarPosition	SoundPosition インタフェースを実装するクラスで、極座標系を使用して聴者と仮想音源の位置関係を表します。極座標系ではその原点に一定の方向を向いた聴者がいるものとし、そこからの方位角、仰角、距離を指定して仮想音源の位置を表現します。
CartesianPosition	SoundPosition インタフェースを実装するクラスで、x 軸、y 軸、z 軸の 3 軸から構成されるデカルト座標系を使用して聴者と仮想音源の位置関係を表します。PolarPosition が常に聴者が原点に存在するローカル座標系と捉えることができるのに対し、CartesianPosition は任意の位置に存在する聴者と仮想音源を包含するいわゆるワールド座標系と捉えることができます。ただし CartesianPosition でも最終的には聴者からみた仮想音源の位置関係が計算され、使用されます。
CartesianListener	CartesianPosition クラスで定位を表現する際に、デカルト座標系上に置く聴者を表現するクラスです。デカルト座標系で定位を表すために必要となる聴者の向きの設定は、このクラスを使用して行います。
SoundMotion	SoundMotion クラスは「移動する定位」を表現するためのクラスです。移動開始からの経過時間とその時点での定位を経路情報として設定することで、その経路に沿って仮想音源が移動していくようにユーザーに感じさせることができます。
Audio3DListener	3D サウンド制御に関する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イベントリスナを定義するインタフェースです。本プロファイルでは、SoundMotion (移動する定位) における移動完了イベントが定義されています。
FastMath	3D 関連機能での使用を前提とした高速数値演算ユーティリティです。本プロファイルに対応する携帯電話の i アプリ実行環境では浮動小数点数演算が使用できますが、このクラスでは浮動小数点数演算を内部的に整数演算に置き換えます。このため、通常の浮動小数点数演算より大きな誤差を含む可能性はありますが高速に演算を行うことができます。
Transform	3 次元アフィン変換用の行列を扱うクラスです。CartesianListener に聴者の向きを与える際に、視点座標への変換行列を設定して使用することができます。

Vector3D	3次元ベクトルを表すクラスです。内積・外積計算、正規化といったユーティリティメソッドを備えています。座標系上の位置や向きなどを表現するために使用されます。
----------	---

13.2.2 Audio3D

Audio3D クラスは、3D サウンド制御を行うためのコントローラの役割を持ちます。Audio3D オブジェクトは **AudioPresenter** オブジェクト毎に割り当てられており、**AudioPresenter.getAudio3D()** メソッドを使用して取得することができます。

3D サウンド制御のオン、オフを切り替えるには、**Audio3D.enable()** メソッドおよび **Audio3D.disable()** メソッドを使用します。3D サウンド制御では i アプリからの定位の指示により 3D 制御を行う方法の他に、あらかじめサウンドデータに定位情報を埋め込んでおき、サウンド再生時にその情報を使用して自動的に 3D 制御を行う方法を取ることもできます。3D 制御をサウンドデータに埋め込まれた定位情報を使用して行うか、あるいはサウンドデータに埋め込まれた定位情報を無視して i アプリからの定位の指示を有効とするかは、**enable()** メソッドの引数で指定します。

Audio3D.enable() を使用して 3D サウンド制御を有効にすると、システムはサウンドを 3D 制御するためのリソース（3D サウンド制御リソースと呼びます）を消費します。このリソースの総数はメーカーにより異なりますが、どの機種でも最低限 4 つは備えています。3D サウンド制御リソースとは仮想音源、言い換えると仮想的に移動するスピーカーを表現するためのリソースであり、i アプリが定位を指示して制御を行うケースでは 1 つの **AudioPresenter** につき 1 つのリソースが消費されます。しかしサウンドデータに埋め込まれた定位情報を使用して制御を行うケースでは、サウンドデータの内容によっては 1 つの **AudioPresenter** が複数のリソースを使用することがあります。あるサウンドデータがいくつの 3D サウンド制御リソースとを使用するかは、**MediaSound.getProperty()** メソッドに引数 **MediaSound.AUDIO_3D_RESOURCES** を指定することで知ることができます。

3D 制御を有効にした後、定位を指定して具体的に聴者と仮想音源の位置関係を設定するには、**Audio3D.setLocalization()** メソッドを使用します。移動する定位（**SoundMotion**）で定位を制御する場合には、**Audio3D.setListener()** を呼び出すことで定位の移動完了をイベントとして通知を受けるためのイベントリスナーを使用することができます。

定位の指定方法は次項にて解説します。

注意事項：

- 移動する定位（**SoundMotion**）を使用しなくても、i アプリが一定の時間間隔ごとに定位の設定変更を繰り返すことで定位が移動しているように感じさせることができます。この場合、定位を変更する間隔は目安として **Audio3D.setTimeResolution()** が返す時間（単位 msec）以上を空けるようにしてください。定位の変更には一定の負荷が伴うため、このメソッドが返す間隔より小さな間隔で定位の変更を行っているシステムに過度の負荷がかかり、意図通りの動作とならない場合があります。

13.2.3 定位の指定

聴者と仮想音源の位置関係を表す定位を表現するには、以下の 3 種類の方法があります。

CartesianPosition / CartesianListener を使用する

x 軸、y 軸、z 軸の 3 軸から構成されるデカルト座標系を用いて聴者と仮想音源の位置を指定します。聴者、仮想音源とも座標系内の任意の位置に存在することができますが、最終的には聴者からみた音源の位置（方向と距離）が計算されて実際の効果に反映されます。**CartesianListener** は、座標上に配置する聴者を表すクラスです。**CartesianListener** オブジェクトに聴者の位置と向きを設定し、そのオブジェクト、および仮想音源の位置を **CartesianPosition** オブジェクトに設定することで、両者の位置関係を表現します。

PolarPosition を使用する

Z 軸の正方向を向いた聴者を球状の座標の中心に置き、その聴者からの方位角、仰角および距離を指定することで仮想音源の位置を指定します。常に聴者が座標原点に存在しているため、指定した仮想音源の位置がそのまま聴者との位置関係を表します。

SoundMotion を使用する

CartesianPosition や **PolarPosition** が、アプリケーションプログラムからパラメータを設定変更しない限り自動的に移動しない固定された定位を表すのに対し、**SoundMotion** は指定された経路をたどって自動的に移動する定位を表します。経路情報は、移動開始からの経過時間とその時点での定位

(**CartesianPosition** または **PolarPosition**) をセットとして与えることで指定します。経路情報を複数与えることで、複雑な経路をたどらせることもできます。**SoundMotion** を使用する場合、**Audio3D.setListener()** メソッドを使用して 3D サウンド制御リスナーを設定することで、定位の移動完了の通知をイベントとして受け取ることができます。

これらのクラスは、定位の表現のためのインタフェース **Audio3DLocalization** を実装しており、**Audio3D.setLocalization()** の引数に設定することができます。

SoundMotion は、定位の移動を行う前にその経路があらかじめ決まっている場合に使用します。これに対し、移動中の定位の経路がその場その場でのユーザー操作によって動的に決定付けられるような場合には **SoundMotion** は使用せず、**CartesianPosition** や **PolarPosition** を使用してアプリケーションプログラムが動的に位置の設定変更を行うようにします。

13.2.4 3D サウンド制御未サポート機種における振る舞い

本項では、3D サウンド制御機能をサポートしていない Star-2.0 プロファイル対応機種における、前項までの 3D サウンド制御 API を使用した場合の振る舞いについて解説します。

3D サウンド制御機能をサポートしない Star-2.0 プロファイル対応機種においても、**Audio3D** オブジェクトを取得することができ、**Audio3D** クラスの各メソッドを呼び出した場合も例外がスローされることなくリターンします。ただし、一部のメソッドでは呼び出し時の振る舞いが異なります。呼び出し時に振る舞いが異なるメソッドを以下に示します。

- 3D サウンド制御の有効・無効を切り替えることはできません。
3D サウンド制御の状態は常に無効状態であり、有効・無効を切り替えることはできません。
enable() メソッドあるいは **disable()** メソッドを呼び出しても、何もせずリターンします。
- 3D サウンド制御リソースは常に 2 です。
getResources() メソッドは常に 2 を返します。
- 定位を指定することはできません。
setLocalization() メソッドを呼び出しても、何もせずリターンします。
- 3D オーディオのイベントリスナーを登録することはできません。
setListener() メソッドを呼び出しても、何もせずリターンします。

なお、3D サウンド制御機能をサポートしない機種では、3D サウンド制御を無効状態とみなしてサウンドデータを再生します。

注意事項：

- 上記で解説した 3D サウンド制御未サポート機種における振る舞いは、Star-2.0 プロファイルに対応した機種にのみ適用されるもので、過去のプロファイルの機種には適用されません。

第 14 章

内部ストレージ

Star-2.0 プロファイルに対応した携帯電話では、i アプリが利用するメディアリソースなどのデータを、携帯電話ネイティブのマルチメディア系ファイル保存用領域（データ BOX）にある内部ストレージに保存することができます。

内部ストレージは、ScratchPad に比べて保存領域が大きいいため、内部ストレージ機能は大容量データの記憶領域としての利用に適しています。

注意事項：

- i アプリが内部ストレージに保存したデータを携帯電話のデータ BOX メニューから参照することはできません。また、i アプリ一覧画面のサブメニューなどからも参照することはできません。
- 内部ストレージに保存したデータは、データを保存した i アプリからのみ参照・削除することができます。
- ユーザー操作により i アプリが削除される際、該当 i アプリが内部ストレージに保存したデータも削除されます。
- 携帯電話の中には、携帯電話本体と外部メモリの間で i アプリの移動をサポートする機種があります（1.4.5 項参照）。このような機種において、内部ストレージを使用する i アプリを携帯電話本体から外部メモリに移動する際、内部ストレージに保存したデータは削除されます。

14.1 内部ストレージ機能 API の API 構成

この項では内部ストレージ機能 API のクラス、インタフェースの構成について解説します。

内部ストレージ機能 API は、以下のパッケージに属するクラス、インタフェースから構成されます。

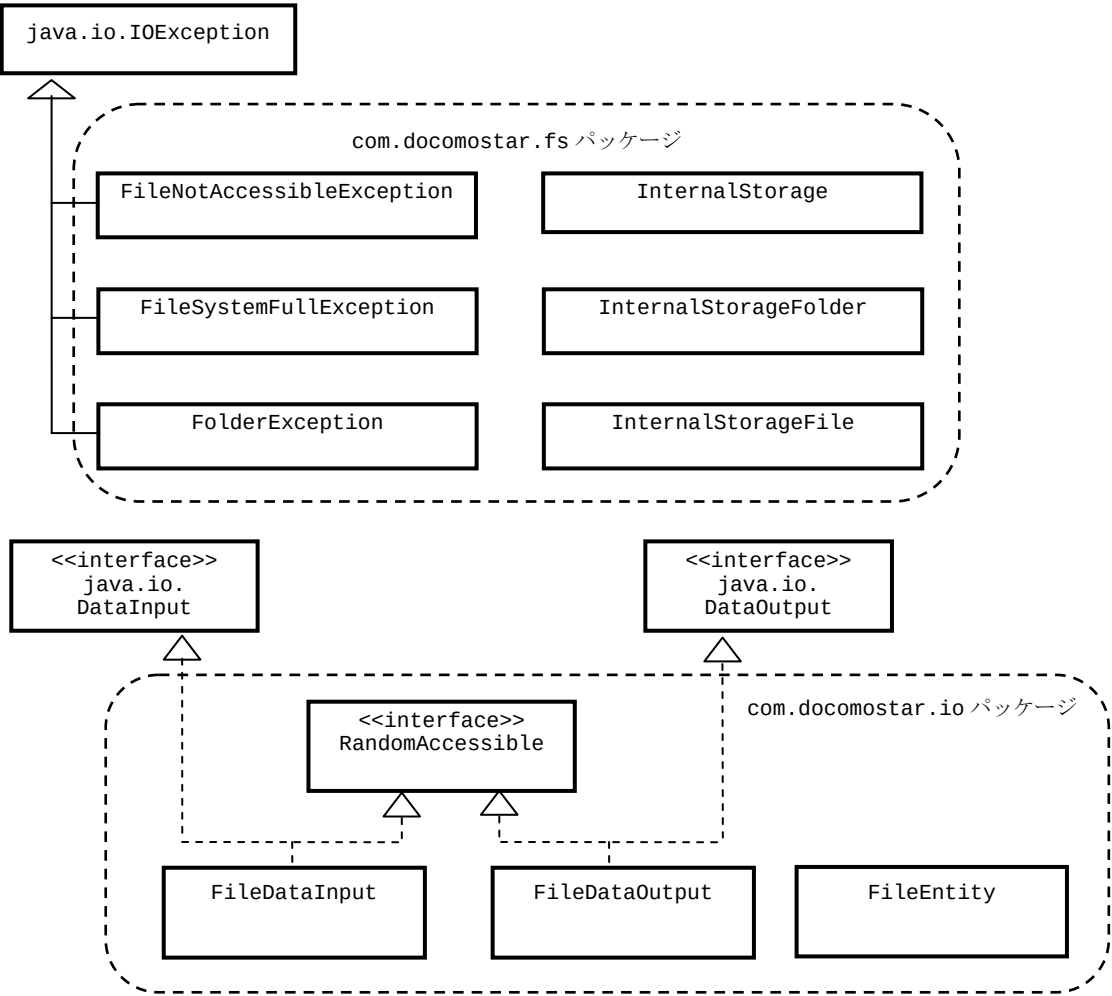
com.docomostar.fs パッケージ

com.docomostar.fs パッケージでは、ファイルシステムに関する機能を扱うためのクラスを規定しています。

com.docomostar.io パッケージ

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

以下に、内部ストレージ機能 API のクラス・インタフェース構成を図示します。



各クラス、インタフェースの機能概要は以下の通りです。

com.docomostar.fs パッケージ

クラス	機能概要
InternalStorage	内部ストレージを表現するクラスです。内部ストレージ内のフォルダやファイルを取得したり、内部ストレージで利用できる空き容量を取得する機能を提供します。
InternalStorageFolder	内部ストレージに作成されたフォルダを表すクラスです。i アプリ固有フォルダもこのクラスで表現されます。初めてi アプリから内部ストレージにアクセスした際、i アプリ実行環境によってそのi アプリに固有のフォルダが作成されます。i アプリではそのi アプリ固有フォルダ配下にフォルダやファイルを生成することができます。
InternalStorageFile	内部ストレージに作成されたファイルを表現するクラスです。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、 <code>com.docomostar.io.FileEntity</code> クラスのオブジェクトを取り出して使用します。
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既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	内部ストレージに空き容量がないために書き込みに失敗したことを示す例外です。

FolderException	存在しないフォルダの取得や既に存在している名前のフォルダの作成を指示した場合や、i アプリ固有フォルダ以外のフォルダに対して操作を指示したなどの理由で、フォルダの操作に失敗したことを示す例外です。
-----------------	--

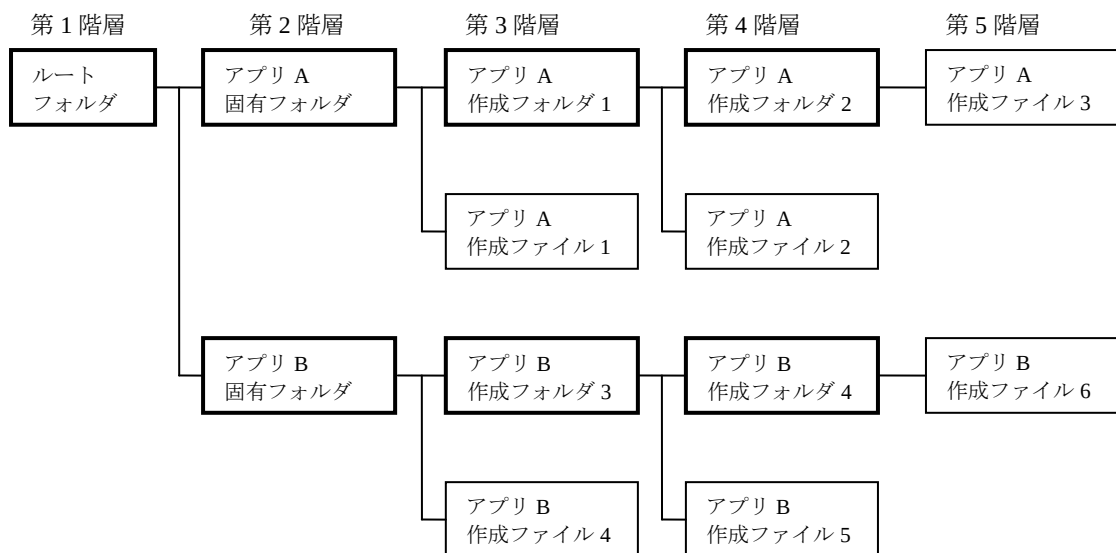
com.docomostar.io パッケージ

クラス、インタフェース	機能概要
FileEntity	ファイルの実体を表現するクラスです。ファイル内のデータにアクセスするには、まず InternalStorageFile 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。

14.2 内部ストレージの構造

内部ストレージ機能におけるフォルダやファイルは、最上位層をルートフォルダとし、ルートフォルダの直下に i アプリ固有フォルダ、その配下に i アプリが作成したフォルダやファイルが位置するという階層構造によって管理されています。

内部ストレージ機能における階層構造を以下に示します。



各階層におけるフォルダについて説明します。

- ルートフォルダ
内部ストレージの最上層のフォルダです。最上層はルートフォルダのみが存在し、それ以外のフォルダやファイルは存在しません。
内部ストレージ機能を利用する全ての i アプリで共通のフォルダです。

- i アプリ固有フォルダ
ルートフォルダ直下に存在する、個々の i アプリと一対一に対応するフォルダです。
i アプリ固有フォルダは、それぞれの i アプリが初めて `InternalStorage.getMyFolder()` メソッドを呼び出した際に、i アプリ実行環境によって作成されます。
他の i アプリと対応付けられている i アプリ固有フォルダに対してフォルダやファイルの生成/取得などの操作を行うことはできません。
- i アプリ固有フォルダ配下
i アプリは、自身と対となる i アプリ固有フォルダ(自アプリ固有フォルダ)配下に対してのみ、フォルダやファイルを生成したり、取得したりすることができます。

i アプリから作成可能なフォルダおよびファイルの名前には以下の付与規則があります。

- フォルダ名：“xxxxxxxxxxx”
“x～x”の部分には 1 文字以上 12 文字以下の ASCII 文字が指定可能です。
- ファイル名：“xxxxxxxx.yyy”
“x～x”の部分には 1 文字以上 8 文字以下の ASCII 文字が指定可能です。
“y～y”の部分には 0 文字以上 3 文字以下の ASCII 文字が指定可能です。なお、“y～y”の部分が 0 文字の場合は‘.’を付加することはできません。

“x～x”、“y～y”の部分に指定可能な ASCII 文字は以下の通りです。

- ‘A’～‘Z’（大文字のみ。小文字を指定した場合は全て大文字として扱われます。）
- ‘0’～‘9’
- ‘_’（アンダースコア）

なお、ルートフォルダのフォルダ名は“/”です。また i アプリ固有フォルダのフォルダ名は、i アプリ実行環境によって i アプリ固有フォルダが生成される際に自動的に付与されます。

注意事項：

- 現在のプロファイルでは、内部ストレージ機能でサポートされる階層の数は前述の図で第 5 階層までです。第 4 階層に位置する `InternalStorageFolder` オブジェクトに対してフォルダを生成することはできません。

14.3 内部ストレージ機能 API の利用

上述したクラス、インタフェースを使用して i アプリが実際に内部ストレージとの間でデータの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自アプリ固有フォルダを表す `InternalStorageFolder` オブジェクトを取得する。
- (2) `InternalStorageFolder` オブジェクトのメソッドを使用して、そのフォルダにサブフォルダやファイルを作成したり、フォルダ内に存在するフォルダやファイルを取得する。
いずれの場合も、フォルダは `InternalStorageFolder` オブジェクト、ファイルは `InternalStorageFile` オブジェクトとして i アプリに返される。
- (3) `InternalStorageFile` 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。

14.3.1 自アプリ固有フォルダの取得

自アプリ固有フォルダを取得するには、**InternalStorage** クラスの **static** メソッド **getMyFolder()** メソッドを使用します。このメソッドは、既に自アプリ固有フォルダが存在する場合にはそのフォルダを表す **InternalStorageFolder** オブジェクトを返します。自アプリ固有フォルダが存在しない場合、i アプリ実行環境は、このメソッドを呼び出した i アプリと対となる i アプリ固有フォルダを生成し、そのフォルダを表す **InternalStorageFolder** オブジェクトを返します。

14.3.2 フォルダの操作

InternalStorage.getMyFolder() メソッドで自アプリ固有フォルダを取得すると、そのフォルダに新規にフォルダやファイルを生成したり、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、フォルダは **InternalStorageFolder** オブジェクト、ファイルは **InternalStorageFile** オブジェクトとして表現されます。

フォルダの新規作成：

フォルダを新規に作成するには、**InternalStorageFolder.createSubFolder()** メソッドを使用します。このメソッドの引数にフォルダ名（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ォルダが作成され、そのフォルダに対応する **InternalStorageFolder** オブジェクトが返ります。

ファイルの新規作成：

ファイルを新規に作成するには、**InternalStorageFolder.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 **InternalStorageFile** オブジェクトが返ります。

既存フォルダへのアクセス：

既存フォルダにアクセスする主な方法には、以下の 2 つがあります。

- フォルダの名前を **InternalStorageFolder.getFolder()** メソッドの引数に指定し、そのフォルダに対応する **InternalStorageFolder** オブジェクトを取得する方法
- **InternalStorageFolder.getAllFolders()** メソッドにより、そのフォルダ直下に存在する全てのフォルダの **InternalStorageFolder** オブジェクトを配列として取得する方法

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主な方法には、以下の 2 つがあります。

- ファイルの名前を **InternalStorageFolder.getFile()** メソッドの引数に指定し、そのファイルに対応する **InternalStorageFile** オブジェクトを取得する方法
- **InternalStorageFolder.getAllFiles()** メソッドにより、そのフォルダ直下に存在する全てのファイルの **InternalStorageFile** オブジェクトを配列として取得する方法

既存のフォルダやファイルにアクセスする方法には、上述の方法に加えて、取得したいフォルダやファイルに対するパス（パス区切り文字“/”から始まる絶対パス）を指定する方法もあります。パスを指定してフォルダを取得するには **InternalStorage** クラスの **getFolder()** メソッドを、ファイルを取得するには **InternalStorage** クラスの **getFile()** メソッドを使用します。ただし、自アプリ固有フォルダ以外の場所のフォルダやファイルにアクセスすることはできません。

InternalStorageFolder クラスや **InternalStorageFile** クラスはフォルダやファイルの、内部ストレージ内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 **InternalStorageFile** 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、**InternalStorageFile** 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）。

InternalStorageFolder クラスには、上記のフォルダやファイルに対する操作機能以外に、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。

deleteFolder()	： このフォルダを削除します。 また、このフォルダ配下にある全てのフォルダやファイルも削除されます。
equals()	： 引数で指定されたオブジェクトと比較します。
getFolderPath()	： このフォルダのパス名（絶対パス）を取得します。パスの区切り文字は“/”です。 ルートフォルダに対してこのメソッドを呼び出すと常に“/”が返ります。
getParentFolder()	： このフォルダが存在するフォルダに対応する InternalStorageFolder オブジェクトを取得します。
hashCode()	： このフォルダに対応する InternalStorageFolder オブジェクトのハッシュコードを取得します。

注意事項：

- ファイルを新規に作成した後、データを1バイトも書き込まずにそのファイルへのアクセスを終了した場合に、そのファイル（サイズ0のファイル）が削除されるかどうかは機種により異なります。

14.3.3 ファイルの操作と入出力

InternalStorage オブジェクトや **InternalStorageFolder** オブジェクトから取り出した **InternalStorageFile** オブジェクトは内部ストレージ内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などのファイル管理情報を取り扱います。**InternalStorageFile** 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。

deleteFile()	： このファイルを削除します。
equals()	： 引数で指定されたオブジェクトと比較します。
getFilePath()	： このファイルのパス名（絶対パス）を取得します。パスの区切り文字は“/”です。
getFolder()	： このファイルが存在するフォルダに対応する InternalStorageFolder オブジェクトを取得します。
getLength()	： このファイルのデータサイズを取得します。
hashCode()	： このファイルに対応する InternalStorageFile オブジェクトのハッシュコードを取得します。

また、ファイル内のデータにアクセスするには **InternalStorageFile.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。**InternalStorageFile** オブジェクトが内部ストレージ内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。

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

データ入出力オブジェクトを使用したファイルアクセス：

FileEntity.openDataInput() メソッドや **FileEntity.openDataOutput()** メソッドにより、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、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。

【Star-2.0】

内部ストレージ機能は Star-2.0 プロファイルで新設されました。

第 15 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ユーティリティ機能について解説します。

15.1 デジタル署名 API

本項では、デジタル署名 API について解説します。

電子認証サービス「FirstPass」は 2012 年 8 月 31 日をもってサービスを終了しました。そのため、2012 年 9 月 1 日以降、FirstPass で発行されたクライアント証明書を使用して署名することはできません。

なお、携帯電話の中には、コンテンツプロバイダがユーザーに発行したクライアント証明書を使用して署名する機能が搭載されている機種がありますが、これらの機種では 2012 年 9 月以降もデジタル署名 API を利用することができます。詳細は 15.1.2 項を参照してください。

初期の機種を除く FOMA 携帯電話では、ユーザー毎に発行されたクライアント証明書を UIM カードに格納し、インターネットアクセスの際に SSL クライアント認証機能を提供している FirstPass を利用することができます。デジタル署名 API ではこの FirstPass のクライアント証明書を使用して、任意のデータにデジタル署名を施す（PKCS#7 SignedData 形式への変換）機能を提供します。本 API を使用することにより、署名者の認証とデータの完全性の確認を行うことができます。

注意事項：

- デジタル署名 API で署名を行う際は、FirstPass にて発行されたクライアント証明書を使用します。このため、デジタル署名を行う携帯電話では、あらかじめ UIM カードに FirstPass のクライアント証明書がダウンロードされている必要があります。また、FirstPass のサービスを受けるためには、バージョン 2 以降の UIM カードを使用している必要があります（バージョン 1 の UIM カードでは FirstPass のサービスを受けることはできません）。現在携帯電話に挿入されている UIM カードのバージョン情報は、`com.docomostar.system.PhoneSystem` クラスの static メソッド `getProperty()` の引数に、`PhoneSystem.UIM_VERSION` を指定することにより調べることができます。
- デジタル署名 API で施された署名を検証するには、FirstPass における認証局の証明書（ドコモルート CA 証明書）を使用する必要があります。携帯電話上では、ドコモルート CA 証明書は工場出荷時に組み込まれています。また、サーバー側でドコモルート CA 証明書を利用するには、別途 NTT ドコモにサービスの申し込みを行う必要があります。

- FirstPass の詳細や、サーバー側でドコモルート CA 証明書を利用するための申し込みなどについては NTT ドコモの以下のサイトを参照してください。
<http://www.nttdocomo.co.jp/service/safety/firstpass/index.html>

15.1.1 デジタル署名 API のクラス構成

デジタル署名 API を構成するクラスは、`com.docomostar.security` パッケージで提供されます。以下に、本機能のクラス構成を示します。

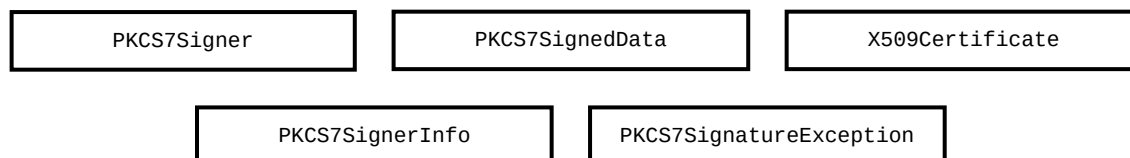


図 28 デジタル署名 API のクラス構成

各クラスの機能概要は以下の通りです。

クラス	機能概要
PKCS7Signer	デジタル署名付きデータを生成する機能を提供するクラスです。このクラスのオブジェクトに署名対象データを与えることでデジタル署名を行います。デジタル署名付きデータは、次に記載する PKCS7SignedData クラスで表現されます。
PKCS7SignedData	デジタル署名付きデータを表現するクラスです。このクラスのオブジェクトからは、デジタル署名付きデータのバイナリデータ表現を取り出したり、デジタル署名付きデータに含まれているコンテンツ（署名前のデータ）を取り出したりすることなどができます。
X509Certificate	X509 証明書を表現するクラスです。デジタル署名付きデータには、署名者の証明書情報が含まれています。デジタル署名 API では、デジタル署名付きデータ（ PKCS7SignedData オブジェクト）に含まれる証明書情報を本クラスのオブジェクトとして取得することができます。
PKCS7SignerInfo	デジタル署名付きデータに含まれる署名者情報を表現するクラスです。このクラスのオブジェクトは、 PKCS7SignedData オブジェクトから <code>getSignerInfos()</code> メソッドを使用して取り出すことができます。
PKCS7SignatureException	デジタル署名に関する処理の例外を定義します。デジタル署名実行時にユーザーに求められる暗証番号入力力がキャンセルや失敗した場合、またはデジタル署名の検証結果が失敗だった場合などに発生します。

15.1.2 デジタル署名 API の利用

以下に、デジタル署名 API の利用方法の概要を示します。デジタル署名 API の利用は、主に以下の 2 つに大別されます。

- ・ 署名対象データへのデジタル署名の実施（**PKCS7Signer** クラスの利用）
- ・ デジタル署名付きデータの操作（**PKCS7SignedData** クラスの利用）

署名対象データへのデジタル署名の実施

署名対象データにデジタル署名を行う手順を以下に示します。

- 1) **PKCS7Signer** オブジェクトを、コンストラクタを使用して生成します。
- 2) **PKCS7Signer** オブジェクトの設定を行います。設定項目には、ダイジェストアルゴリズムの選択と、署名対象データのコンテンツタイプの設定の 2 つがあります。ダイジェストアルゴリズムの設定は、

`setDigestAlgorithm()` メソッドに引数 "SHA-1" もしくは "MD5" を指定することで行います。デフォルトのダイジェストアルゴリズムは SHA-1 です。またコンテンツタイプの設定は、`setContentType()` メソッドの引数に `PKCS7Signer` で定義される定数 `DATA` もしくは `SIGNED_DATA` を指定することで行います。通常のデータに署名を行う際は `DATA` を指定しますが、署名対象データ自身がデジタル署名付きデータである場合（二重にデジタル署名するような場合）には `SIGNED_DATA` を指定します。

- 3) `update()` メソッドを使用して、`PKCS7Signer` オブジェクトに署名対象データを設定します。`update()` メソッドを複数回呼ぶことで署名対象データを複数回に分けて設定したり、`reset()` メソッドを使用してそれまでに設定した署名対象データを破棄することができます。
- 4) 署名対象データを設定し終わった後で、`sign()` メソッドを呼び出します。このメソッドは署名対象データにデジタル署名を実施し、その結果として `PKCS7SignedData` オブジェクトを返します。デジタル署名処理において、ダイジェストアルゴリズムは 2) のフェーズで SHA-1 もしくは MD5 を選択することができますが、暗号方式は RSA 固定となります。

上記により取得した `PKCS7SignedData` オブジェクトは、次に示すデジタル署名付きデータの操作方法に従って操作することができます。

デジタル署名付きデータの操作

デジタル署名付きデータ（`PKCS7SignedData` オブジェクト）に対しては、以下のような操作を行うことができます。

- デジタル署名付きデータの検証

`verify()` メソッドを使用して、デジタル署名付きデータの検証を行うことができます。検証を行うことにより、署名者の認証とデータの完全性の確認を行うことができます。

- デジタル署名付きデータのバイナリデータ表現の取得

`getEncoded()` メソッドを使用して、デジタル署名付きデータを DER エンコードされた ASN.1 形式のバイナリデータとして取得することができます。このメソッドを使用することで、デジタル署名付きデータを、HTTP 通信などの手段により外部に送信可能なバイト配列として取得することができます。

`PKCS7SignedData` オブジェクトは、このバイナリデータを引数としてコンストラクタを呼び出すことでも生成することができます。このことを利用して、署名されたデータを別の携帯電話に送信し、送信先の i アプリでオブジェクト化した上で検証などの操作を行うといったことも可能です。

- デジタル署名付きデータに含まれているコンテンツ（署名前のデータ）の取得

`getContent()` メソッドを使用して、デジタル署名付きデータから署名前のコンテンツデータを取得することができます。

- デジタル署名付きデータに含まれている証明書情報の取得

`getCertificates()` メソッドを使用して、デジタル署名付きデータに含まれている証明書の情報を取得することができます。証明書の情報は `X509Certificate` クラスのオブジェクトとして表現されます。デジタル署名付きデータに複数の証明書が含まれている場合、それら全てについて証明書情報を取得することができます。`X509Certificate` オブジェクトからは、証明書の発行者と被認証者についての情報（一般名や国名、組織名など）や証明書の有効期間終了日といった情報を得ることができます。

- デジタル署名付きデータに含まれている署名者情報の取得

`getSignerInfos()` メソッドを使用して、デジタル署名付きデータに含まれている署名者情報を取得することができます。署名者の情報は `PKCS7SignerInfo` クラスのオブジェクトとして表現されます。`getSignerInfos()` メソッドは、デジタル署名付きデータに含まれている全ての署名者情報を返します。`PKCS7SignerInfo` オブジェクトからは、署名者情報の発行者識別名の設定値（一般名、組織名、部門名）やシリアルナンバーの情報を得ることができます。

以下に、デジタル署名 API で取り扱うデータの相関関係を示します。

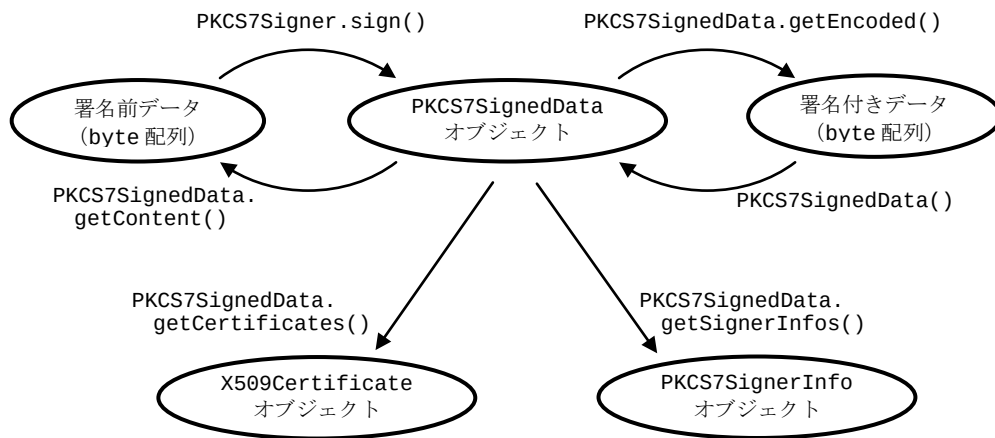


図 29 デジタル署名 API で取り扱うデータの相関関係

携帯電話の中には、FirstPass で提供されるクライアント証明書以外に i モードサイト（携帯電話側では i モードブラウザ）を介してコンテンツプロバイダがユーザーにクライアント証明書を配布し、ユーザー側でそのクライアント証明書を利用可能とする機能が搭載されている機種があります。

i アプリではそのようにしてダウンロードされたクライアント証明書でデジタル署名を行えるようにするため、以下のような API が i アプリオプション API として規定されています。

CertificateStore クラス (com.docomostar.system パッケージ) :

クライアント証明書のネイティブ保存領域を操作するためのクラスです。このクラスの static メソッド `selectEntryId()` を呼び出すことで、携帯電話に保存されているクライアント証明書の中から任意の 1 つをユーザーが選択し、そのクライアント証明書のエントリ ID を i アプリに知らせることができます。

PKCS7Signer.sign(int) メソッド :

`CertificateStore.selectEntryId()` で取得したクライアント証明書のエントリ ID を指定することで、そのクライアント証明書を利用してデジタル署名を行います。

15.2 マナーモード中の音声出力設定

携帯電話がマナーモードや公共モード（ドライブモード）設定中の状態では、携帯電話本体のスピーカから音声は出力されません。Star-2.0 プロファイルに対応する携帯電話では、マナーモードや公共モード設定中のためスピーカで音声を出力できない状態において、ユーザーの許可の下、その i アプリからの音声に限り一時的に出力できるようにすることができます。

マナーモードや公共モード中に i アプリからの音声出力を許可するには、

`StarApplicationManager` クラスの `allowAudioOutput()` 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、ユーザーに i アプリからの音声出力を許可するかどうかを確認するダイアログが表示されます。このダイアログでユーザーの許可が得られる

と、この i アプリからの音声出力が可能となります。その後、i アプリはレジュームし、**allowAudioOutput()**メソッドは **true** を返します。ユーザーの許可が得られない場合、**allowAudioOutput()**メソッドは **false** を返します。

また、マナーモードや公共モード中に i アプリからの音声出力が許可されているかどうかを確認することもできます。マナーモードや公共モード中に i アプリからの音声出力が許可されているかどうか確認するには、**StarApplicationManager** クラスの **isAllowedAudioOutput()**メソッドを使用します。このメソッドは **allowAudioOutput()**メソッドによって音声出力が許可されているかどうかを **boolean** 値で返します。

注意事項：

- **allowAudioOutput()**メソッドによって可能となる音声出力は以下のものです。電話着信時やメール着信時などの着信音は出力されません。
 - 標準サウンド
 - メディアプレゼンタによる再生
 - i アプリ音楽プレイヤー機能
 - i アプリ-Flash 連携機能における、Flash コンテンツ(SWF ファイル)再生時の音声出力
- **allowAudioOutput()**メソッドによる音声出力の許可設定はこのメソッドを呼び出した i アプリが終了するまで有効となり、一度許可された音声出力設定を i アプリ起動中に再度禁止することはできません。

第 16 章

i アプリのビルド、テスト、 およびパッケージ化

この章では、i アプリの開発サイクルについて概要を説明します。NTT ドコモでは、PC 上で i アプリを開発するための開発環境（以降 i アプリ開発環境とします）を提供していますが、ここでは i アプリ開発環境および付属のツールの個々の詳細な情報には触れません。i アプリ開発環境の詳細については、i アプリ開発環境のユーザーズガイドを参照してください。

開発者は、Java 2 SDK, Standard Edition（以降 J2SE SDK とします）を利用したことがあり、J2SE SDK に含まれるツールやドキュメントの知識があるものとします。

なお、開発した i アプリをテストのために実際の携帯電話にインストールするには、i アプリを配布するための Web サイト（インターネットに接続されている必要があります）を設定する必要があります。i アプリの配布については、第 17 章を参照してください。

16.1 i アプリ開発環境のインストール

i アプリ開発環境を使用するには、NTT ドコモが配布する i アプリ開発環境のアーカイブファイル入手し、PC 上でインストール作業を行います。i アプリ開発環境のインストール手順の詳細については、i アプリ開発環境のユーザーズガイドを参照してください。

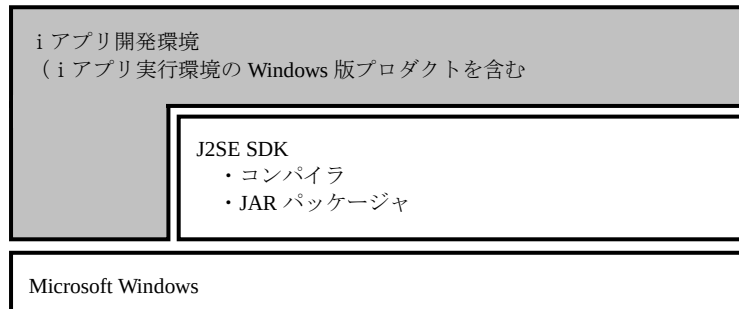
なお、i アプリ開発環境をインストールするには、i アプリ開発環境のほかに J2SE SDK を用意する必要があります。J2SE SDK をインストールした上で i アプリ開発環境をインストールしてください。

16.2 i アプリ開発環境のコンポーネント

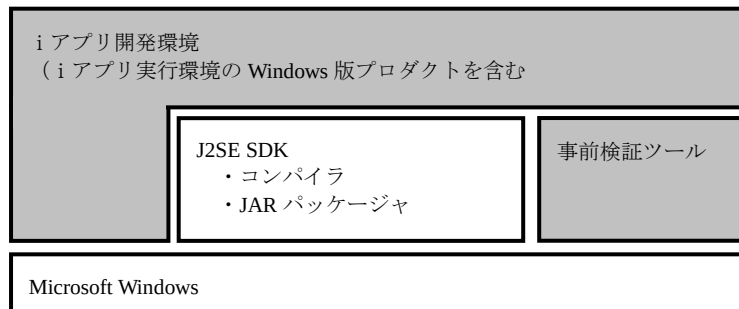
i アプリ開発環境は、PC 上で i アプリを開発するための簡易な開発環境です。これを使用することで、PC 上で i アプリをビルド、シミュレート実行することができます。

i アプリ開発環境は以下のようなコンポーネント構成を取っています。

●Star-1.3 プロファイル以降の i アプリ開発環境



●Star-1.2 プロファイル以前の i アプリ開発環境



注意：ハッチングされたコンポーネントが NTT ドコモより提供されます。i アプリ開発環境がサポートする動作環境（Microsoft Windows や J2SE SDK のバージョンなど）は、i アプリ開発環境のバージョン（対応プロファイル）により異なります。詳細については、i アプリ開発環境のユーザーズガイドを参照してください。

図 30 i アプリ開発環境の構成

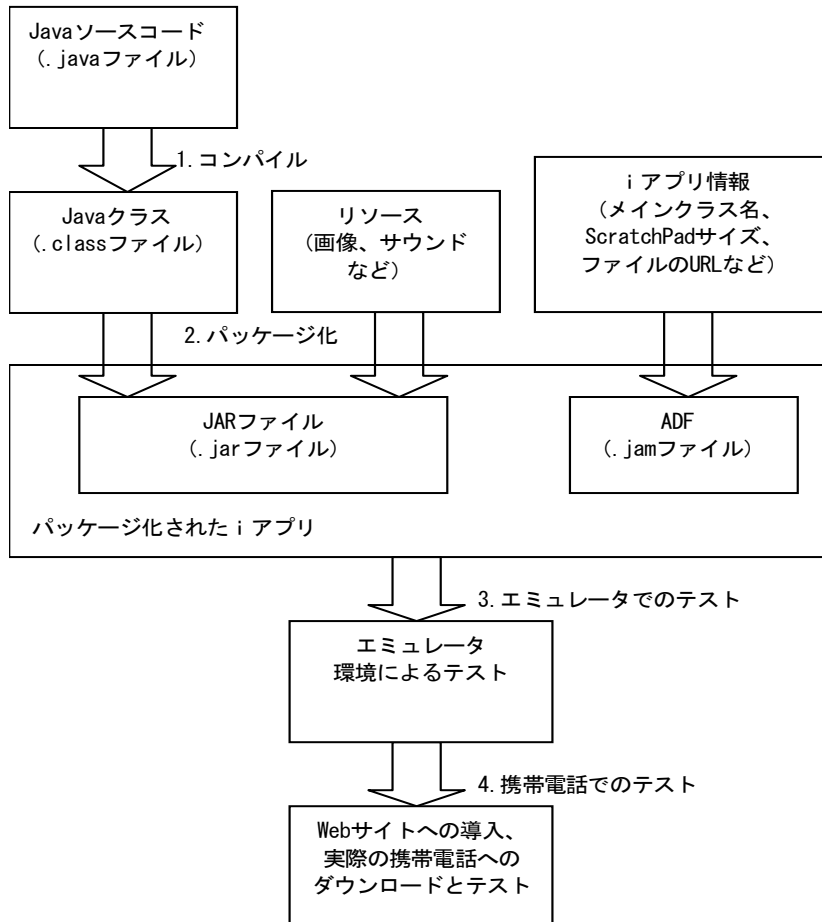
i アプリ開発者は、i アプリ開発環境を使用して i アプリのビルド（コンパイルや Jar パッケージング）とテスト実行を行うことができます。なお i アプリのビルドは、各ビルド用コマンドをコマンドラインから呼び出して行うこともできます。本書では、i アプリ開発環境をエミュレータと表記することがあります。

注意事項：

- Star-1.2 プロファイル以前の i アプリ実行環境では、i アプリのビルド時にアプリケーションクラスの事前検証処理を行う必要があり、i アプリ実行環境に事前検証ツール（preverify）が含まれていました。この点について、Star-1.3 プロファイル以降の i アプリ実行環境では内部仕様が一部変更となり、これに伴い i アプリビルド時の事前検証プロセスが不要になります（事前検証が施された i アプリも実行は可能です）。
- Star-1.3 プロファイル向け以降の i アプリ開発環境は、ビルド時に事前検証プロセスを行わないようになっています。このため、Star-1.3 プロファイル向け以降の i アプリ開発環境でビルドした i アプリは Star-1.2 プロファイル以前の i アプリ対応携帯電話では動作しないため注意してください。Star-1.2 プロファイル以前の i アプリ対応携帯電話で動作させる i アプリは、Star-1.2 プロファイル向け以前の i アプリ開発環境でビルドしてください。

16.3 開発サイクルの概要

次に、一般的な開発サイクルを示します。



※Star-1.2 プロファイル以前の i アプリを開発する場合は、「2.パッケージ化」の前にクラス（.class ファイル）に対して事前検証処理を行う必要があります。

図 31 開発サイクル

コンパイル – ソースファイルをクラスファイルにコンパイルするには、J2SE SDK の Java 言語コンパイラ（javac）を使用します。このときコンパイラは、i アプリ実行環境に属していないパッケージやクラス、メソッドがアプリケーションプログラムで使用されていないか検査します。

パッケージ化 – すべてのアプリケーションクラスファイルとリソースファイルを JAR ファイルとしてパッケージ化するには、J2SE SDK のアーカイブツール（jar）を使用します。jar コマンドの使用方法は、J2SE SDK における jar コマンドの標準的な使用方法と同じです。携帯電話に保存される JAR ファイルサイズを最小限に保つため、非圧縮オプションは指定しないようにしてください。

エミュレータでのテスト – i アプリの機能が正しく動作するかどうかをエミュレータ環境で確認します。

実際の i アプリ対応携帯電話でのテスト – i アプリ（JAR ファイル、ADF）およびダウンロードナビゲート用 HTML コンテンツを Web サイトに配置し、実際の携帯電話にダウンロードします。そして、携帯電話で i アプリを実行して機能が正しく動作するか確認します。使用可能なメモリ容量など、携帯電話によってハードウェア・リソースの容量が異なる場合があるので、ターゲットとする全てのメーカーの携帯電話で i アプリの動作確認を行うようにしてください。

コンパイル、パッケージ化の一連の流れ（ビルド作業）は個々のツールを独立して使用する以外に、エミュレータの持つ i アプリビルド機能を使用して一括して行うこともできます。エミュレータの i アプリビルド機能については、i アプリ開発環境のユーザーズガイドを参照してください。

注意事項：

- Star-1.2 プロファイル以前の i アプリ実行環境で動作させる i アプリは、JAR ファイルとしてパッケージ化する前のアプリケーションクラスファイルに対して事前検証処理を行う必要があります。Star-1.3 プロファイル以降ではアプリケーションクラスファイルに対する事前検証処理は不要となっており、i アプリ開発環境はビルド時に事前検証プロセスを行わないようになっていますので、Star-1.2 プロファイル以前の i アプリ対応携帯電話で動作させる i アプリは Star-1.2 プロファイル向け以前の i アプリ開発環境でビルドしてください。

16.4 通信先アプリケーションの作成

クライアント/サーバー型 i アプリでは、i アプリの通信相手となるサーバー側アプリケーションを作成する必要があります。サーバー側アプリケーションは、クライアント側アプリケーション（i アプリ）と HTTP プロトコルで通信するものであればどのような技術を使用したものでも構いません。

また i アプリが OBEX 外部接続を使用する場合、通信先となる OBEX アプリケーションを用意する必要があります。i アプリ対応携帯電話間で OBEX 外部接続を利用するケースでは、i アプリ API を使用して OBEX クライアント、OBEX サーバーとも開発することができます。携帯電話とその他の機器を接続する場合、通信先アプリケーションの開発については、通信先となるプラットフォームで利用する OBEX アプリケーション開発環境のドキュメントを参照してください。

16.5 i アプリのパッケージ化

図 31 に示したように、i アプリは、ADF と JAR ファイル（クラスファイルおよびソースファイルを含んでいる）の 2 つのファイルから構成されます。

16.5.1 ADF の作成

ADF は、SJIS コードで記述されたテキストファイルです。このファイルは、“=”で分離されたキーと値のペアからなる行で構成されます。ファイルの編集には、任意のテキストエディターを使用することができます。このファイルには、次のキーのエントリ（行）を指定します。

AppType: 必須キー

JAR パッケージ（本体の JAR ファイル）に含まれる i アプリの種別（フルのみアプリ、ミニのみアプリ、フル+ミニアプリ）を指定します。このキーが指定されていない場合は、DoJa アプリと判定されます。

フルアプリの場合は“FullApp”、ミニアプリの場合は“MiniApp”を指定します。フル+ミニの場合はこれらをカンマ区切りで列挙します。

指定例：AppType = MiniApp,FullApp

フル+ミニアプリの場合に値を列挙する順番は、i アプリダウンロード直後の i アプリ起動、バージョンアップ後の i アプリ起動、および i アプリ一覧からの i アプリ起動において意味を持ちます。これらの起動時は、このキーの先頭に書かれている方の i アプリが優先的に起動されます。

注意事項：

- i アプリを更新する際、AppType キーの値を変更することはできません。i アプリの更新時に AppType キーの値が変更されている場合、携帯電話は更新処理を中止します。ただし、フル+ミニアプリにおいて値を列挙する順番を変更した場合には i アプリを更新することができます。

AppName: 必須キー

i アプリ名（最大 16 バイト）。携帯電話で i アプリ一覧を表示させた際に、ここに指定された名前が表示されます。SJIS で日本語名を指定することができます。

AppVer: オptional キー

i アプリのバージョン（最大 10 バイト）。開発者はこのキーを利用して i アプリのバージョン管理を行うことができます。ASCII 文字のみ指定可能です。

PackageURL: 必須キー

i アプリの JAR ファイルの URL（最大 255 バイト）。相対パスを指定した場合、そのベースディレクトリは ADF の位置となります。ASCII 文字のみ指定可能です。なお、URL に指定するホスト名はシンボリックでなければなりません（"www.nttdocomo.co.jp"など）。数値 IP アドレス（"127.0.0.1"など）は使用できません。

InnerPackageName: オptional キー（フル+ミニの場合のみ必須）

1 つの JAR パッケージ内に複数の実行ファイル(JAR ファイル)を持つフル+ミニアプリにおいて、フルアプリ、ミニアプリそれぞれのファイル名を指定するために使用します。フルのみアプリ、ミニのみアプリではこのキーは使用できません。ASCII 文字のみ指定可能です。

指定例：InnerPackageName = Inner1.jar,Inner2.jar

各々の JAR ファイルの i アプリ種別は、AppType キーに指定された i アプリ種別の並び順に対応しているものと見なされます。つまり、AppType キーの項のような指定例が行われている場合、このキーに指定された 1 つ目の JAR ファイルがミニアプリ、2 つ目の JAR がフルアプリと見なされます。

AppSize: 必須キー

JAR ファイルのサイズ。バイト単位の整数値を指定します。JAR ファイルサイズには、3.2 項で解説した制限があります。

InnerAppSize: オptional キー（フル+ミニの場合のみ必須）

1 つの JAR パッケージ内に複数の実行ファイル(JAR ファイル)を持つフル+ミニアプリにおいて、フルアプリ、ミニアプリそれぞれのファイルサイズを指定するために使用します。フルのみアプリ、ミニのみアプリの場合は、このキーは使用できません。ファイルサイズには、3.2 項で解説した制限があります。

指定例：InnerAppSize = 29484,543832

各サイズは、InnerPackageName キーに指定された JAR の並び順に対応しているものと見なされます。つまり、InnerPackageName キーの項のような指定例が行なわれている場合、上記例では Inner1.jar のサイズが 29484 バイト、Inner2.jar のサイズが 543832 バイトと見なされます。

AppClass: 必須キー

i アプリの起動に使われるメインクラス名（フルパッケージ名付き・最大 255 バイト）。フルのみアプリとミニのみアプリでは 1 つのクラス名を指定します。またフル+ミニアプリでは、それぞれに対応するクラス名を 2 つ、カンマ区切りで指定します。ASCII 文字のみ指定可能です。

指定例：AppClass = App1Main,App2Main

フル+ミニアプリで指定する各クラス名は、AppType キーに指定された i アプリ種別の並び順に対応しているものと見なされます。つまり、AppType キーの項のような指定例が行われている場合、App1Main がミニアプリのメインクラス、App2Main がフルアプリのメインクラスと見なされます。

AppParam: オプショナルキー

メインクラスの起動パラメータ。パラメータ間はスペースで区切ります（最大 255 バイト）。これらのパラメータをアプリケーションプログラムから取得するには、com.docomostar.StarApplicationManager.getArgs() メソッドを使用します。ASCII 文字のみ指定可能です。省略時は起動パラメータがないものとします。

ConfigurationVer: オプショナルキー

アプリケーションプログラムが前提としている、参照コンフィグレーションバージョン。このキーが指定されていない場合は、その i アプリは CLDC の全てのバージョンでサポートされている API のみを使用していることを示します。このキーの設定値は以下のフォーマットに従います。

CLDC-x.x (x は 0~9 の数字。本バージョンでは"1.1"をサポート)

ProfileVer: オプショナルキー

i アプリ実行環境のプロファイルバージョン。このキーが指定されていない場合は、その i アプリが i アプリ実行環境のすべてのバージョンで動作することを示します。このキーの設定値は以下のフォーマットに従います。

Star-x.x (x は 0~9 の数字)

本バージョンでは以下をサポートしています。

- Star-1.0 (2008 年秋以降発売の Star プロファイル搭載携帯電話に対応)
- Star-1.1 (2009 年春以降発売の Star プロファイル搭載携帯電話に対応)
- Star-1.2 (2009 年秋以降発売の Star プロファイル搭載携帯電話に対応)
- Star-1.3 (2010 年春以降発売の Star プロファイル搭載携帯電話に対応)
- Star-1.5 (2010 年秋以降 (2011 年春以降も含む) 発売の Star プロファイル搭載携帯電話に対応)
- Star-2.0 (2011 年秋以降発売の Star プロファイル搭載携帯電話に対応)

SPsize: オプショナルキー

ScratchPad のサイズ。バイト単位の整数値を指定します。i アプリが使用可能な ScratchPad サイズには、3.2 項で解説した制限があります。省略時は ScratchPad を使用しないものとします。

ScratchPad は内部的に最大 16 個まで分割管理することができます。ScratchPad を分割するには、SPsize キーの値として、個々のバイト数をカンマ区切りで列挙します (カンマの前後に空白は含めないでください)。ScratchPad にアクセスする際は、URL に ScratchPad の番号 (0~最大 15 まで) を指定しますが、その番号はこのキーで指定されたサイズの並びに対応します。

("SPsize = 1024,2048" と記載されている場合、0 番目の ScratchPad のサイズが 1024 バイトに、1 番目の ScratchPad のサイズが 2048 バイトになります。)

ScratchPad を分割管理する場合は、分割された個々の合計サイズがサイズ制限の範囲内に収まるようにしてください。

ミニのみアプリの場合において、ScratchPad を分割するよう指定されている場合はダウンロードできません。

LastModified: 必須キー

i アプリの最終変更日時。i アプリの更新は、このキーの内容に基づいて行われます。フォーマットは以下の通りです。

Dow, DD Mon YYYY HH:MM:SS TimeZone

Dow : 曜日	DD : 日	Mon : 月	YYYY : 年
HH : 時間	MM : 分	SS : 秒	TimeZone : タイムゾーン

タイムゾーンは"GMT"および"JST"に対応しています。また、"+0900"というようにタイムゾーンを時差で指定することもできます。タイムゾーンが指定されていない場合、"JST"が指定されたものとします。

指定例: **Fri, 25 Feb 2000 12:04:25 JST**

UseNetwork: オプションルキー

ネットワーク機能を使用する場合に、通信方式を指定します。Star プロファイルでは"yes"が唯一の有効な値です (HTTPS 使用時も"yes"を指定します)。この指定がない i アプリは、ネットワーク接続を要求することはできません。

TargetDevice: オプションルキー

ターゲットとする携帯電話の機種名を指定します。すべての機種をターゲットとする場合はエントリ自体を指定しません。このキーに記述する機種名には、ユーザーエージェントに含まれている名前と同等のものを使用します。ターゲットとなる機種が複数存在する場合は、カンマ区切りで列挙します。

例: **X01A,Y01A,Z01A**

なお、2008 年秋以降に発売される NTT ドコモの携帯電話では、それまでのシリーズの概念がなくなります。これに伴い、2008 年秋以降に発売される携帯電話では、ターゲットとなるシリーズ全機種の一括指定 (all 指定) は廃止されます。

LaunchAt: オプションルキー

i アプリに自動起動を行わせる場合、その起動タイミングを指定します。先頭の 1 文字は形式指示子です。i アプリ実行環境では、形式指示子としてインターバルを表す"I"をサポートします。

✧ 時間間隔指定形式 (形式指示子: I)

フォーマット: **I h**

(h: 時間 有効桁数は 2 桁 "I"と数字の間にはスペースが 1 つ必要です)

指定例: **I 6** (6 時間毎に起動する)

時間間隔の起点は、i アプリが携帯電話にダウンロードされた時刻となります。

i アプリが自動起動するタイミングで他のアプリケーションが実行されていたり、通話中であつたりした場合、その i アプリは自動起動できません。その場合、携帯電話は各機種固有の方法で、自動起動できなかった i アプリがあることを利用者に通知します。

なおメーカーによっては携帯電話の機能として、i アプリを特定の時刻に自動起動するようユーザーによる登録設定が行える機能を搭載していることがあります。このような機能で自動起動される場合、i アプリの起動形態はタイマー起動 (StarApplication.LAUNCHED_FROM_TIMER) ではなく通常起動 (StarApplication.LAUNCHED_FROM_MENU など) となります。

注意事項:

- ミニのみアプリでこのキーが指定されていた場合はアプリをダウンロードすることはできません。
- フル+ミニアプリが自動起動で起動される場合、フルアプリとして起動されます。

UseTelephone: オプションルキー

通話機能を利用する i アプリであることを宣言します。値に"call"が指定されている場合、その i アプリは通話発信 (com.docomostar.system.Phone.call() メソッド) を使用することができます。この指定がない i アプリは、通話発信を利用することはできません。

UseBrowser: オプションルキー

ブラウザ機能を利用する i アプリであることを宣言します。値に"launch"が指定されている場合、その i アプリは com.docomostar.system.Launcher.launch() メソッドを使用してブラウザを起動することができます。この指定がない i アプリは、ブラウザを起動することはできません。。

LaunchByMail: オプションキー

この i アプリがメールからの起動を許可することを宣言します。値には、起動を許可するメールの送信元メールアドレスを 1 つだけ ASCII 形式で指定します (最大 50 バイト)。このキーの値が、後方一致比較の結果として送信元メールアドレスに全て含まれていた場合、そのメールから i アプリを起動することを許可します。なお、このキーに値 "any" が指定されている場合、送信元メールアドレスの制限は行いません。

注意事項:

- ミニのみアプリでこのキーが指定されていた場合はアプリをダウンロードすることはできません。

LaunchByBrowser: オプションキー

この i アプリがブラウザからの起動を許可することを宣言します。値には、起動を許可する Web ページの URL を 1 つだけ ASCII 形式で指定します (最大 255 バイト)。このキーの値が、前方一致比較の結果としてブラウザで表示中の URL に全て含まれていた場合、その Web ページから i アプリを起動することを許可します。このキーに値 "any" が指定されている場合、URL の制限は行いません。

なお、トルカビューアからの i アプリ起動を行う場合は LaunchByBrowser キーの代わりに LaunchByToruCa キーを使用します。前方一致比較の適用先が Web ページの URL ではなくトルカ (詳細) の取得先 URL となる他は、LaunchByToruCa キーの使用方法は LaunchByBrowser キーと同じです。

また、MyFACE からの i アプリ起動を行う場合は LaunchByBrowser キーの代わりに LaunchByFaceApp キーを使用します。前方一致比較の適用先が Web ページの URL ではなく MyFACE のダウンロード元 URL となる他は、LaunchByFaceApp キーの使用方法は LaunchByBrowser キーと同じです。

注意事項:

- ミニのみアプリでこのキーが指定されていた場合はアプリをダウンロードすることはできません。

AllowPushBy: オプションキー

この i アプリが、外部インタフェースからの起動を許可することを宣言します。外部インタフェースからの起動には、赤外線ポートからの vTrigger オブジェクト受信による起動、およびコード認識機能における 2 次元バーコードからの起動の 2 種類があります。

<赤外線ポートからの vTrigger オブジェクト受信による起動>

AllowPushBy キーの値には、以下の形式のパラメータを 1 つだけ ASCII 形式で指定します (接頭辞 "Irda:" を含めて最大 255 バイト)。

Irda:<command>

<command>には、i アプリ起動を許可する vTrigger オブジェクトに指定される起動コマンドを指定します。このキーの値が、i アプリに対して起動要求を行った vTrigger オブジェクトの起動コマンドと完全一致した場合、その vTrigger オブジェクトによって i アプリを起動することを許可します。

なお、<command>には空白およびタブを含めることはできません。

<コード認識機能における 2 次元バーコードからの起動>

AllowPushBy キーの値には、以下の形式のパラメータを 1 つだけ ASCII 形式で指定します (接頭辞 "Code:" を含めて最大 255 バイト)。

Code:<command>

<command>には、i アプリ起動を許可する 2 次元バーコードに指定される起動コマンドを指定します。このキーの値が、i アプリに対して起動要求を行った 2 次元バーコードの起動コマンドと完全一致した場合、その 2 次元バーコードによって i アプリを起動することを許可します。

なお、<command>には空白およびタブを含めることはできません。

注意事項:

- ミニのみアプリでこのキーが指定されていた場合はアプリをダウンロードすることはできません。

AppTrace: オプションキー

i アプリ開発のフェーズにおいて、携帯電話上で簡易なデバッグ機能を提供するためのキーです。このキーに値"on"を指定すると、携帯電話上で以下の機能が有効となります。

- キャッチされない例外により i アプリが異常終了した場合、どのような例外により i アプリが異常終了したかを表示します。
- i アプリから標準出力または標準エラー出力に出力した文字列をロギングし、i アプリ終了後に表示させることができます。ただしロギング可能な文字列長には制限があり（最大 512 バイト）、新しく書き込まれたものから順に制限の範囲内のものが表示されます。なお、`java.lang.Throwable.printStackTrace()` を呼び出した結果はこのログには保存されません。
- アプリケーションマネージャの i アプリ情報表示機能が詳細モードとなります。

i アプリの開発が完了した際には、本キーは ADF から削除してください。

DrawArea: オプションキー

この i アプリがフルアプリとして実行された場合に前提としている画面サイズ（i アプリの描画エリアサイズ）を指定します。大きな描画エリアを持つ携帯電話で小さい描画エリアを前提とした i アプリを実行すると、ディスプレイに対してセンタリングされた、指定サイズの矩形領域内で i アプリが動作します。矩形領域の外側の部分に i アプリから描画を行うことはできません（その部分の表示色はメーカーにより異なります）。また VGA 対応などの高精細ディスプレイをもつ機種では、描画エリアが一定サイズの場合にアスペクトを保ったまま整数倍に拡大表示が行われます。

すべての機種で整数倍に拡大表示が行われる描画エリアサイズは、「240x240」「240x320」「320x240」です。それ以外の描画エリアサイズが指定された場合は、機種によっては拡大表示が行われない場合があります。

キーの値は以下のように指定します。

`<width>x<height>` （`<width>`は横、`<height>`は縦のドット数 "x"は小文字のみ有効）

例： `DrawArea = 120x120`

`<width>`の設定可能最小値は 96、`<height>`の設定可能最小値は 72 です。また、このキーで指定された描画エリアサイズは、i アプリをダウンロードしようとしている携帯電話の描画エリアサイズ以下である必要があります（横または縦のいずれも、ダウンロードしようとしている携帯電話のサイズ以下でなければなりません）。これらの条件を満たさない設定が行われている場合、その i アプリをダウンロードすることはできません。

なお、このキーが指定されていない場合は、480 x 480 が指定されたものとして i アプリを実行します。

Pallet: オプションキー

このアプリがミニアプリとして実行された場合のパレットフォーマットを指定します。

キーの値は以下のように指定します。

`<PalletSize>:<PalletDrawArea>:<PositionX>:<PositionY>:<PalletFace>`

- `<PalletSize>`

以下のフォーマットで指定します。

`<width>x<height>` （`<width>`は横、`<height>`は縦のドット数 "x"は小文字のみ有効）

指定可能なサイズは 320x240、240x320、160x160、440x80 の 4 つです。

省略された場合は、160x160 が指定されたものとして動作します。

- `<PalletDrawArea>`

パレット上での描画領域のサイズを、以下のフォーマットで指定します。

`<width>x<height>` （`<width>`は横、`<height>`は縦のドット数 "x"は小文字のみ有効）

`PalletSize` で指定したサイズ以下のサイズのみ指定することができます。

省略された場合は、`<PalletSize>`と同様のサイズが指定されたものとして動作します。

- **<PositionX>,<PositionY>**

パレット上に、**PalletDrawArea** で指定された描画領域を配置する位置を指定します。パレットの左上を原点とした、x、y 方向の座標値で指定します。描画領域がパレットからはみ出すような位置を指定することはできません。

省略した場合は、それぞれ 0,0 が指定されたものとして動作します。

- **<PalletFace>**

パレットのフェイスとなる画像のファイル名を指定します。

フェイス画像のファイルは、ミニアプリの JAR ファイル内のルートディレクトリの位置に置くものとします。（フル+ミニアプリでは、内包されるミニアプリ JAR ファイル内のルートディレクトリの位置に置きます。）

省略された場合は、フェイス指定なしとなります。

指定例：

全て記載する場合

 Pallet=440x80:240x60:120:0:face.gif

省略形式で記載する場合

 Pallet=440x80::120::face.gif

 Pallet=440x80:440x60:0:0:

このキーが指定されていない場合は、それぞれの値の指定を省略したものとして動作します。

パレットについての詳細は、本書 9.2 項を参照してください。

GetSysInfo: オプションルキー

この i アプリが、携帯電話のアイコン情報を参照することを宣言します。このキーに値"yes"が指定されている i アプリがダウンロードされると、携帯電話は i アプリにアイコン情報の参照を許可するかユーザーに確認を求めます。このキーが指定されていない i アプリやユーザーから許可を得ていない i アプリは、アイコン情報を参照することはできません。

なおアイコン情報の取得には、**PhoneSystem.getAttribute()**メソッドを使用します。

詳細は、本書 第 8 章を参照してください。

LaunchApp: オプションルキー

アプリケーション連携により他の i アプリを起動（連携モードまたはランチャーモード）する i アプリであることを宣言します。値に"yes"が指定されている場合、その i アプリは **com.docomostar.system.Launcher.launch()**メソッドを使用して他の i アプリを起動することができます。この指定がない i アプリは、他の i アプリを起動することはできません。

注意事項：

- ミニのみアプリでこのキーが指定されていた場合はアプリをダウンロードすることはできません。

LaunchByApp: オプションルキー

アプリケーション連携により他の i アプリから起動されようとする場合の動作を制御します。値に"deny"を指定することで、その i アプリが他の i アプリからランチャーモードで連携起動されることを拒否します。

注意事項：

- ミニのみアプリでこのキーが指定されていた場合はアプリをダウンロードすることができません。

AccessUserInfo: オプションルキー

アプリケーション連携により、携帯電話のユーザーデータ領域にアクセスする i アプリであることを宣言します。値に"yes"が指定されている場合、その i アプリは以下のユーザーデータ操作を行うことができます。

- ・ 電話帳エントリ、電話帳グループの登録
- ・ ブックマークの登録
- ・ スケジュールの登録、スケジュールの起動
- ・ 画像の登録・選択・取得

GetUtn: オプションルキー

この i アプリが、携帯電話の個体識別情報を参照することを宣言します。このキーに値"terminalid"が指定されている場合は携帯電話本体の識別情報を、また値"userid"が指定されている場合は UIM カードの識別情報を参照することができます。携帯電話本体の識別情報と UIM カードの識別情報を両方参照する場合は、これら 2 つをカンマ区切りで両方指定します(カンマの前後に空白は含めないでください)。

このキーが指定された i アプリがダウンロードされると、携帯電話は i アプリに個体識別情報の参照を許可するかユーザーに確認を求めます。このキーが指定されていない i アプリやユーザーから許可を得ていない i アプリは、個体識別情報を参照することはできません。また、UIM カードを搭載していない携帯電話では、UIM の識別情報を取得することはできません。

AppIcon: オプションルキー

携帯電話ネイティブの i アプリ一覧表示画面 (IC カードメニューなども含みます) においては、各 i アプリの名称の他にその i アプリ独自のアイコンを表示できます。**AppIcon** キーは、アイコンとして使用する画像を指定するために使用します (アイコンを持たない i アプリについては、システムが持つデフォルトのアイコンが表示されます)。

アイコンデータの画像ファイルは、アプリケーション JAR ファイル内のルートディレクトリの位置に同梱します (フル+ミニアプリでは、2 つの JAR ファイルを包含する全体の JAR ファイルのルートディレクトリに配置します)。**AppIcon** キーでは、この画像ファイルのファイル名を以下の書式で指定します。

AppIcon = <ファイル名 1> [, <ファイル名 2>]

ファイル名 1 には縦横ドットサイズが 48×48 ドットの画像ファイルのファイル名を、またファイル名 2 を指定する場合は 96×96 ドットの画像ファイルのファイル名を指定します (カンマの前後に空白は含めないでください)。メーカーにより、アイコンとしてサポートする画像のサイズがこれら 2 通りに分かれていますが、**AppIcon** キーにファイル名を 2 つ指定している場合には、その i アプリがダウンロードされた機種に応じて適切なサイズのものが選択されます。

なお、アイコンデータには、アニメーション GIF やインターレース GIF を除く GIF87a、GIF89a データ、および JPEG データを使用することができます。またファイル名には ASCII 文字のみ使用可能であり、パス区切りを表す"\"や"/"は使用できません。

【Star-2.0】

Star-2.0 プロファイルに対応する携帯電話では、アイコンデータに使用できる画像ファイルとして PNG データが追加されました。

AppMainTitle: オプションルキー

携帯電話の i アプリ一覧表示画面の表示スタイルの 1 つとして、グラフィカル表示形式が規定されています。グラフィカル表示形式の i アプリ一覧画面では、ユーザーの選択操作でフォーカスが当たっている i アプリについて、**AppIcon** キーのアイコンよりも大きな (160×160 ドット) 画像でメインタイトル画像を表示することができます (メインタイトル画像を持たない i アプリについては、システムが持つデフォルトの画像が表示されます)。**AppMainTitle** キーはメインタイトル画像として使用する画像を指定するために使用します。

メインタイトル画像の画像ファイルは、アプリケーション JAR ファイル内のルートディレクトリの位置に同梱します (フル+ミニアプリでは、2 つの JAR ファイルを包含する全体の JAR ファイルのルートディレクトリに配置します)。**AppMainTitle** キーでは、この画像ファイルのファイル名を以下の書式で指定します。

AppMainTitle = <ファイル名>

ファイル名には縦横ドットサイズが 160×160 ドットの画像ファイルのファイル名を指定します。

なお、メインタイトル画像のデータには、アニメーション GIF やインターレース GIF を除く GIF87a、GIF89a データ、および JPEG データを使用することができます。またファイル名には ASCII 文字のみ使用可能であり、パス区切りを表す“/”や“\”は使用できません。

【Star-2.0】

Star-2.0 プロファイルでは、メインタイトル画像ファイルとして従来の縦横 160 ドットのものに加えて、縦横 320 ドットものを指定することができます。ただし、縦横 320 ドットのメインタイトル画像ファイルを指定する場合にも従来の縦横 160 ドットのメインタイトル画像ファイルを指定する必要があります。従来の縦横 160 ドットのメインタイトル画像ファイルに加えて、縦横 320 ドットのメインタイトル画像ファイルが指定されている場合は、縦横 320 ドットのメインタイトル画像ファイルが使用されます。

縦横 320 ドットのメインタイトル画像を指定する場合の書式は以下のとおりです。

AppMainTitle = <ファイル名 1>,<ファイル名 2>

ファイル名 1 には縦横ドットサイズが 160×160 ドットの画像ファイルのファイル名を、またファイル名 2 には 320×320 ドットの画像ファイルのファイル名を指定します（カンマの前後に空白は含めないでください）。

また、Star-2.0 プロファイルに対応する携帯電話では、メインタイトル画像に使用できる画像ファイルとして PNG データが追加されました。

DeniedMultiApp: オptional キー

FOMA 携帯電話の中には、複数の端末アプリケーション（例えばブラウザと通話機能など）を同時に起動できるものがあります。機種によっては i アプリと音楽プレーヤー、もしくは i アプリとデジタルテレビを同時に起動して、音楽やテレビ音声を聞きながら i アプリを利用するといった使い方をサポートするものもあります。しかし、音楽プレーヤーやデジタルテレビ機能はそれら単体でも相応の CPU パワーを必要とし、i アプリとそれらを同時に利用している場合には i アプリに十分な CPU パワーを提供できない場合があります（このような状況が問題となる i アプリは、ユーザには一般に i アプリの動作が重くなったり、動作速度が不安定になったりするように見えます）。

このため、CPU パワーをフルに使用する必要がある i アプリについて、**DeniedMultiApp** キーを指定することで、音楽プレーヤーやデジタルテレビなどとの同時実行を抑止することができます。

DeniedMultiApp キーの書式は以下の通りです。

DeniedMultiApp = <アプリケーション名> [, <アプリケーション名> ...]

アプリケーション名には、その i アプリと同時に起動することを抑止するアプリケーション名を指定します。現在のプロファイルでは、アプリケーション名には **music**（音楽プレーヤー）、**dtv**（デジタルテレビ）の 2 つが規定されています。また **all** を指定することで、音楽プレーヤーとデジタルテレビの両方を対象とすることもできます。

VmOption: オptional キー

仮想マシンに対する動作オプションを指定します。現在のプロファイルではキーの値として、仮想マシンに対し、アプリケーションプログラムが **System.gc()** を呼び出した契機で強制的にガベージコレクション (GC) を起動することを示す **gc:on** のみが規定されています。

指定例: **VmOption** = **gc:on**

携帯電話の i アプリ実行環境の実装によっては Generational GC など従来とは異なった GC を搭載し、アプリケーションプログラムからの **System.gc()** 呼び出しによる GC 起動を抑止しているものがあります。このような GC は一般的にアプリケーションの実行性能を向上させる効果があるものの、特定のアプリケーションではメモリのフラグメンテーションを引き起こしやすくなったり、アプリケーション実行中に一時的な動作停止を引き起こす場合があるなど好ましくない影響を与えることもあります。

このような問題が発生するアプリケーションでは、本キーに **gc:on** を指定することで、仮想マシンに従来同様の GC 動作を行わせることができます（ただし、全体的なアプリ実行パフォーマンスは低下します）。

RestrictedAudioOutput: オプションルキー

Bluetooth 機能を搭載する携帯電話の中には、i モーションやミュージックなどの音声を Bluetooth の A2DP (Advanced Audio Distribution Profile) に対応するヘッドセットなどで聞くことができますが、このような携帯電話の中には i アプリのサウンドの Bluetooth ヘッドセット出力をサポートするものがあります。しかし、A2DP の機構上、i アプリから出力されたサウンドは少しタイムラグを伴って Bluetooth ヘッドセットから出力されるという特性があります。

このタイムラグがサービス上問題になる i アプリでは、**RestrictedAudioOutput** キーを指定することで、i アプリから再生を指示したサウンドを Bluetooth ヘッドセットへ出力することを抑止することができます。現在のプロファイルではキーの値として、Bluetooth ヘッドセットへの音声出力を抑止することを示す "bluetooth" のみが規定されています。

指定例: **RestrictedAudioOutput** = bluetooth

StartingAppImage: オプションルキー

i アプリを起動する際、携帯電話は起動処理としてアプリケーション JAR ファイルからクラスファイルを抽出しクラスをロードします。この処理では、アプリケーション JAR ファイルに格納されるクラスファイル数やクラスファイルのサイズに応じた起動処理時間を必要とします。**StartingAppImage** キーを使用することで、携帯電話が起動処理を行っている間、i アプリ独自の画像を表示させることができます (このキーが設定されていない i アプリについては、システムが持つデフォルトの画面が表示されます。)

このキーに指定する画像ファイルは、アプリケーション JAR ファイル内のルートディレクトリの位置に同梱します (フル+ミニアプリでは、2 つの JAR ファイルを包含する全体の JAR ファイルのルートディレクトリに配置します)。**StartingAppImage** キーでは、この画像ファイルのファイル名を以下の書式で指定します。

StartingAppImage = [<ファイル名 1>][,<ファイル名 2>]

ファイル名 1 には縦画面表示における起動時画像ファイルのファイル名を、またファイル名 2 には横画面表示における起動時画像ファイルのファイル名を指定します (カンマの前後に空白は含めないでください)。

どちらの画像ファイルにも任意の縦横ドットサイズのものを指定することができます。画像ファイルの縦横サイズが、携帯電話のディスプレイの縦横サイズに一致するかそれよりも大きい場合、ディスプレイの中心位置と画像の中心位置が一致するように起動時画像が全画面表示されます。このとき、ディスプレイの縦横サイズを越える部分は表示されません。

画像ファイルの縦横サイズがディスプレイの縦横サイズよりも小さい場合は、アプリの描画領域の中心位置と画像の中心位置が一致するように起動時画像が表示されます。このとき、アプリの描画領域の縦横サイズを越える部分は表示されません。

なお、起動時画像ファイルには、インターレース GIF を除く GIF87a、GIF89a データ、BMP データ、JPEG データおよび PNG データを使用することができます。またファイル名には ASCII 文字のみ使用可能であり、パス区切りを表す“\”や“/”は使用できません。

【Star-2.0】

StartingAppImage キーは Star-2.0 プロファイルにて新設されました。

以下に ADF の記述例を示します。ADF は下記に示すとおり、1 つのキーエントリを 1 行で記述しなければなりません。各行の行末は、最終行も含め CR LF (0x0d0a) とします。

```
AppType = FullApp,MiniApp
InnnerPackageName=Full.jar,Mini.jar
AppClass = FullClass,MiniClass
AppName = バンキングデモ
ConfigurationVer = CLDC-1.1
ProfileVer = Star-1.0
AppVer = 1.0
PackageURL = http://www.nttdocomo.co.jp/BankingDemo.jar
AppSize = 17556
LastModified = Fri, 25 Feb 2000 12:04:25
```

以下に i アプリの種別毎に、指定可能なキーを示します。

表 ADF キーと i アプリの種別の対応表

キー値	フルのみ アプリ	フル+ミニ アプリ	ミニのみ アプリ
AppType	○	○	○
AppName	○	○	○
AppVer	○	○	○
PackageURL	○	○	○
InnerPackageName	×	○	×
AppSize	○	○	○
InnerAppSize	×	○	×
AppClass	○	○	○
AppParam	○	○	○
ConfigurationVer	○	○	○
ProfileVer	○	○	○
SPsize	○	○	○
LastModified	○	○	○
UseNetwork	○	○	○
TargetDevice	○	○	○
LaunchAt	○	○	×
UseTelephone	○	○	○
UseBrowser	○	○	○
LaunchByMail	○	○	×
LaunchByBrowser	○	○	×
AllowPushBy	○	○	×
AppTrace	○	○	○
DrawArea	○	○	—
Pallet	—	○	○
GetSysInfo	○	○	○
LaunchByApp	○	○	×
LaunchApp	○	○	×
AccessUserInfo	○	○	○
GetUtn	○	○	○
AppIcon	○	○	○
AppMainTitle	○	○	○
DeniedMultiApp	○	○	○
VmOption	○	○	—
RestrictedAudioOutput	○	○	—
StartingAppImage	○	○	○

○ … ダウンロード可能。
 — … ダウンロード可能。記述は無視される。
 × … ダウンロード不可。

16.5.2 JAR ファイルの作成

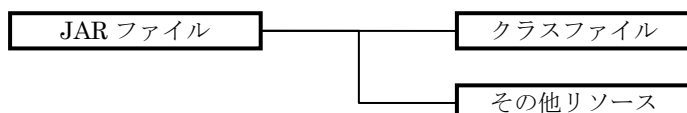
JAR は、アプリケーションを構成するクラスファイルとリソースファイルを圧縮形式で格納するためのアーカイブ機能です。JAR ファイルでは、そのサイズが圧縮前のファイルの合計サイ

ズと比較しておよそ 40%から 50%削減されます（JAR ファイルに含まれる内容により圧縮率は変動します）。

i アプリも JAR ファイルにパッケージングされた形で携帯電話に配布されます。これによって携帯電話に必要な i アプリ保存容量が減るだけでなく、i アプリのダウンロード時間も少なくなります。JAR ファイルにパッケージングされた個々のファイルは、必要に応じてアプリケーションマネージャによりアンパックされます。

- フルアプリ、あるいはミニアプリ

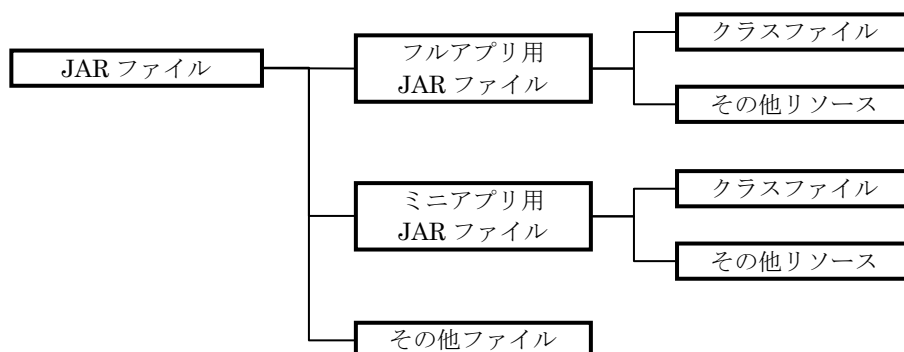
フルアプリ、あるいはミニアプリでは、以下の構成で JAR ファイルにパッケージングします。



上記の中の「その他リソース」には、7.2 項記載の方法で i アプリが読み込むリソースファイルや、ADF の **AppIcon** キーや **AppMainTitle** キーで指定するアイコン画像ファイル、**Pallet** キーで指定するフェイス画像があります。

- フル+ミニアプリ

フル+ミニアプリでは、以下の構成で JAR ファイルにパッケージングします。



フル+ミニアプリの場合、ADF の **AppIcon** キーや **AppMainTitle** キーで指定するアイコン画像ファイルは、「その他ファイル」の位置に格納しますが、i アプリが読み込むリソースファイルや **Pallet** キーで指定するフェイス画像は、内包される各々の JAR ファイルの「その他リソース」の位置に格納する必要があります。

なお、フル+ミニアプリにおいて実行タイプがフルアプリの場合、i アプリが使用するクラスファイルやリソースファイルはまずフルアプリ用 JAR ファイルから検索され、見つからない場合はミニアプリ用 JAR ファイルから検索されます。これに対し、実行タイプがミニアプリの場合はミニアプリ用 JAR ファイルのみが検索対象となります。

16.6 i アプリのテスト

i アプリのテストは、エミュレータおよび実際の i アプリ対応携帯電話を使って行います。

開発の初期段階では、i アプリのテストやデバッグをエミュレータを使って行う方が効率的であり、コスト的にも有利です。エミュレータでは、ユーザーインターフェースや各種入出力、アプリケーションロジックなどのデバッグ作業において、アプリケーションプログラムを修正した結果を PC 上ですぐに確認することができます。

しかしエミュレータ上では、プログラムの実行速度やネットワークの帯域幅などに関して、ハードウェアの動作を完全にエミュレートできるわけではありません。また実際のハードウェアには、メモリ容量などの機種固有の制限事項もあります。したがって、実際の i アプリ対応携帯電話で最終的なテストをして、i アプリの動作が適切かどうかを確認することは不可欠となります。

なおエミュレータの使用方法の詳細については、i アプリ開発環境のユーザーズガイドを参照してください。

エミュレータ環境には、i アプリのテストに関して、さらにいくつかの制約があります。エミュレータでは、i アプリの実行環境は提供されますが、実際の携帯電話に備わっている i アプリ管理機能 (アプリケーションマネージャの持つ機能) はエミュレートされません。たとえば、エミュレータで実行できない 1 つの動作に i アプリの更新があります。これは、アプリケーションマネージャの i アプリ管理機能がエミュレータに実装されていないためです。テストできない動作はその他に次のものがあります。

- **LastModified**、**TargetDevice**、**LaunchAt** に関して、実際の動作をエミュレートする機能はありません。
- エミュレータでは、**ConfigurationVer** キーのチェックは行われません。
- エミュレータは、開発コンピュータのハードディスク (エミュレータのプロジェクトディレクトリ) にある i アプリ JAR ファイルを読み込んで実行するため、**PackageURL** に指定された値は使用されません。実際に携帯電話に i アプリを配布する際には、**PackageURL** には JAR ファイルをダウンロードするための正しい URL が指定されている必要があります。

実際の携帯電話上でのテストのために i アプリを携帯電話にインストールするには、インターネットに接続された Web サーバーに JAR ファイル、ADF、i アプリダウンロード用 HTML を配置する必要があります。これらの詳細については第 17 章を参照してください。

注意事項：

- i アプリがクライアント/サーバー構成を取っている場合、テストに利用する Web サーバーには、サーバー側アプリケーションも適切に導入されている必要があります。

16.7 i アプリ Bookmark

2011 年春以降に発売される携帯電話には、コンテンツパッケージ機能が搭載されたものがあります。コンテンツパッケージは、画像や i モーション、フル楽曲といった i モードの様々な種類のコンテンツを組み合わせるパッケージ化したものです。コンテンツパッケージ機能では、コンテンツパッケージに含まれる各コンテンツを連続してダウンロードすることや、ダウンロードしたコンテンツパッケージ内の各コンテンツを一覧表示したり、各コンテンツの種類に応じた連携動作 (画像表示や i モーション再生など) を行わせたりすることができます。

コンテンツパッケージ機能に対応した機種では、i アプリをコンテンツパッケージに含めることができます。ただし、コンテンツパッケージに含めることができる i アプリは、通常の i アプリとは異なり、アプリケーション本体を構成する Java クラス(class ファイル)を含みません。コンテンツパッケージに含まれる i アプリは、Java クラスの代わりに i アプリのダウンロードサイトの URL を保持し、i アプリのダウンロードサイトのブックマークのような役割を果たします。本項では、このような i アプリを i アプリ Bookmark と表記します。

i アプリ Bookmark はこれまでの i アプリのダウンロードの方法とは異なり、コンテンツパッケージのダウンロード機能を利用して携帯電話にダウンロードされます。

コンテンツパッケージ機能によってダウンロードされた i アプリ Bookmark は、i アプリ一覧画面に表示されますが、そのままでは i アプリとして実行することはできません。ユーザーが i アプリ Bookmark を選択すると、その i アプリ Bookmark が保持する i アプリのダウンロードサイト URL の内容が表示され、そこから i アプリ (i アプリ本体) をダウンロードできます。また、ブラウザやメールなど携帯電話上のネイティブアプリケーションから連携起動する i アプリが i アプリ Bookmark だった場合も i アプリのダウンロードサイト URL の内容が表示されます。

i アプリ Bookmark に対応する i アプリ本体をダウンロードすると、i アプリ一覧画面上の i アプリ Bookmark は i アプリ本体に置き換えられます。

なお、i アプリ Bookmark には i アプリのダウンロードサイトの URL だけでなく、i アプリ一覧画面などに表示する際のアプリ名やアイコン画像などを含めることもできます。i アプリ Bookmark の構成については次項を参照してください。

注意事項：

- i アプリ Bookmark およびコンテンツパッケージ機能は、Star-1.5 プロファイルに対応した携帯電話の中でも 2011 年春以降に発売される機種に順次搭載されます。また、コンテンツパッケージ機能に関する仕様は別途 NTT ドコモより公開されます。

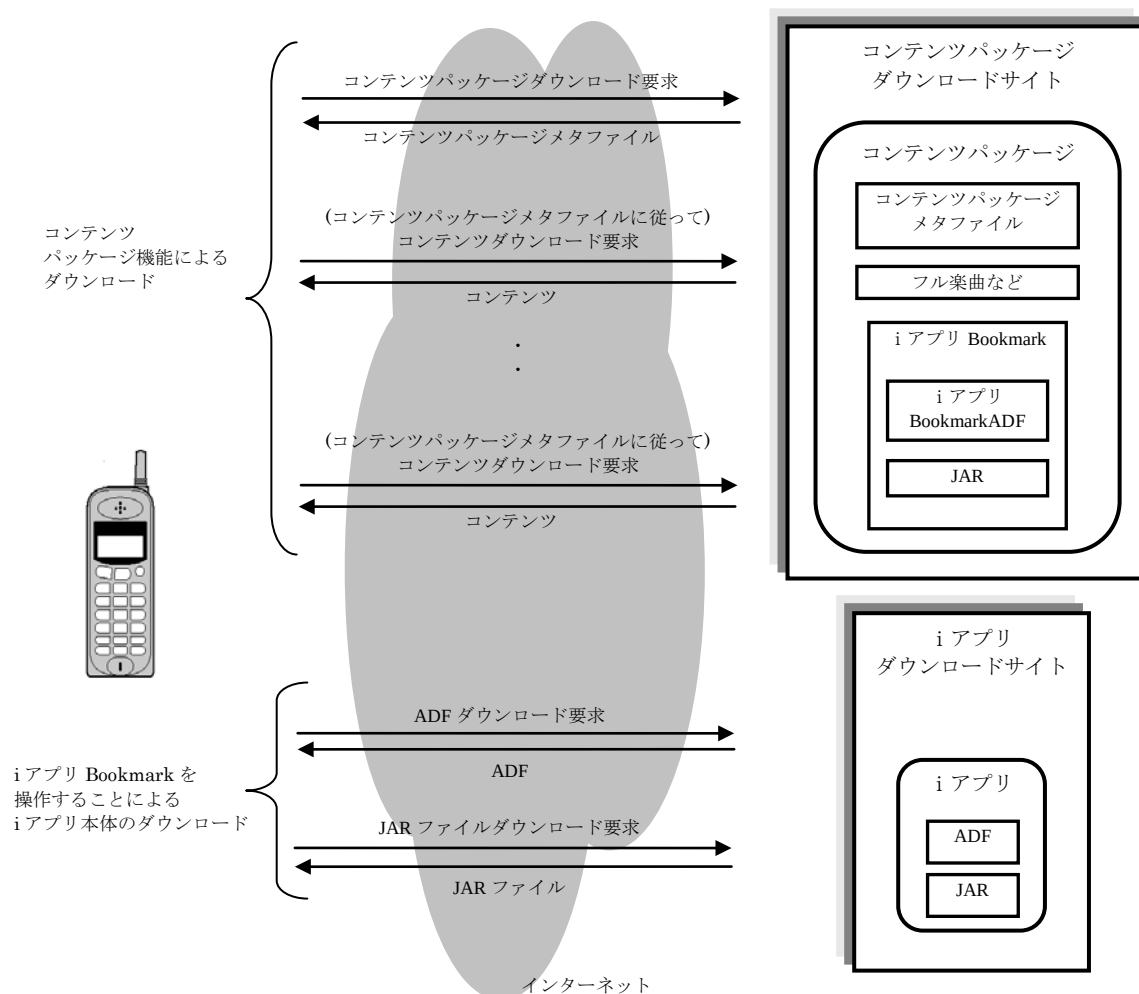
【Star-1.5】 (2011 年春以降発売モデル)

i アプリ Bookmark は Star-1.5 プロファイルで新設されました。

16.7.1 i アプリ Bookmark の構成

i アプリ Bookmark も i アプリと同様に、アプリケーションディスクリプタファイル (“jam” 拡張子を持つテキストファイル。以降 i アプリ BookmarkADF とします) と JAR ファイルから構成されます。i アプリ Bookmark はエミュレータ (i アプリ開発環境) の持つ i アプリ Bookmark 作成機能を使用して作成することができます。エミュレータの i アプリ Bookmark 作成機能については、エミュレータのユーザーズガイドを参照してください。

i アプリ Bookmark をダウンロードするには、コンテンツパッケージのダウンロードサイトおよびコンテンツパッケージが必要になります。また、コンテンツパッケージに含まれるコンテンツの名前やダウンロード用 URL、コンテンツ種別などの情報が記述されたファイル (コンテンツパッケージメタファイル) に、i アプリ Bookmark に関する情報を記述する必要があります。



本項では i アプリ Bookmark を構成する i アプリ BookmarkADF と JAR ファイル、i アプリ Bookmark をコンテンツパッケージに含めるためのコンテンツパッケージメタファイルの記述内容について解説します。

i アプリ BookmarkADF の作成

i アプリ BookmarkADF も i アプリの ADF と同様、複数のエントリ（“=”で分離されたキーと値のペアからなる行）で構成された SJIS コードのテキストファイルです。このファイルには、次のキーのエントリを指定します。

BookmarkURL : 必須キー

i アプリ Bookmark からアクセスする、i アプリのダウンロードサイトの URL（最大 255 バイト）。このキーが指定されている場合に、その ADF は i アプリ本体の ADF ではなく i アプリ BookmarkADF と見なされます。

コンテンツパッケージ内一覧画面や i アプリ一覧画面などからユーザー操作によって i アプリ Bookmark が選択された際に表示する i アプリのダウンロードサイトの URL を絶対 URL 形式で指定します。ASCII 文字のみ指定可能で、大文字小文字は区別されます（たとえば’a’と’A’は異なる文字と判定されます）。また、URL に指定するホスト名はシンボリックリンク（“www.nttdocomo.co.jp”）でなければならず、数値 IP アドレス（“127.0.0.1”）は使用できません。

BookmarkAppURL : 必須キー

i アプリ Bookmark に対応する i アプリの ADF の URL (最大 2048 バイト)。ASCII 文字のみ指定可能で、大文字小文字は区別されます (たとえば 'a' と 'A' は異なる文字と判定されます)。

このキーに指定された URL と完全に一致する URL から ADF が取得される i アプリ (i アプリ本体) が、この i アプリ Bookmark に対応する i アプリと認識されます。

AppName : 必須キー

i アプリ Bookmark 名 (最大 16 バイト)。携帯電話で i アプリ一覧を表示させた際に、ここに指定された名前が表示されます。SJIS で日本語名を指定することができます。

AppIcon : オプションキー

i アプリ一覧表示画面 (IC カードメニューなども含みます) において、各 i アプリの名称と共に表示するアイコン画像を指定します (アイコンを持たない i アプリ Bookmark については、システムが持つデフォルトのアイコンが表示されます)。

アイコンデータの画像ファイルは、i アプリ Bookmark の JAR ファイル内のルートディレクトリの位置に同梱します。

AppIcon キーで画像ファイルを指定する際の書式、アイコンデータとして利用可能な画像サイズや形式は i アプリの AppIcon キーと同様です。詳細は 16.5.1 項の AppIcon キーの説明を参照してください。

AppMainTitle : オプションキー

i アプリ一覧をグラフィカル表示形式で表示した際、i アプリ Bookmark にフォーカスが当たっている場合に表示するメインタイトル画像を指定します (メインタイトル画像を持たない i アプリ Bookmark については、システムが持つデフォルトの画像が表示されます)。

メインタイトル画像の画像ファイルは、i アプリ Bookmark の JAR ファイル内のルートディレクトリの位置に同梱します。

AppMainTitle キーで画像ファイルを指定する際の書式、メインタイトル画像として利用可能な画像サイズや形式は i アプリの AppMainTitle キーと同様です。詳細は 16.5.1 項の AppMainTitle キーの説明を参照してください。

BookmarkPreinType : 必須キー

i アプリ Bookmark をどの i アプリ一覧画面に表示するかを指定します。このキーの値には "Java" あるいは "JavaIC" のいずれかを指定します。このキーに値 "Java" が指定されている場合は i アプリ一覧画面のみに、"JavaIC" が指定されている場合は i アプリ一覧画面に加えて IC カードメニューにも表示されます。

BookmarkSize : 必須キー

i アプリ Bookmark の JAR ファイルのサイズ。バイト単位の整数を指定します。i アプリ Bookmark の JAR ファイルサイズは 100K バイト (102400 バイト) 以内とする必要があります。

PackageURL : 必須キー

i アプリ Bookmark の JAR ファイルの URL (最大 255 バイト)。相対パスを指定した場合、そのベースディレクトリは i アプリ Bookmark ADF の位置となります。

PackageURL キーに URL を指定する際の書式は i アプリの PackageURL キーと同様です。詳細は 16.5.1 項の PackageURL キーの説明を参照してください。

また、i アプリ Bookmark もアプリケーション連携機能をサポートしており、ブラウザやメーラなど携帯電話上のネイティブアプリケーションや外部インタフェースからの連携起動の対象として i アプリ Bookmark

を指定することができます。起動対象として i アプリ Bookmark が指定されると、携帯電話はブラウザを起動し、i アプリ BookmarkADF の **BookmarkURL** キーで指定されたサイトを表示します。ネイティブアプリケーションから i アプリ Bookmark を連携起動の対象として指定する方法は次項を参照してください。

なお、i アプリ Bookmark を起動対象として指定可能なネイティブアプリケーションは、i アプリ BookmarkADF で指定可能なものに限られます。i アプリ BookmarkADF で指定可能なアプリケーション連携機能に関するキーは以下の通りです。

- **LaunchByMail**
この i アプリ Bookmark がメールから起動対象として指定されることを許可することを宣言します。
- **LaunchByBrowser**
この i アプリ Bookmark がブラウザから起動対象として指定されることを許可することを宣言します。
- **LaunchByToruCa**
この i アプリ Bookmark がトルカビューアから起動対象として指定されることを許可することを宣言します。
- **LaunchBySchedule**
この i アプリ Bookmark がスケジューラから起動対象として指定されることを許可することを宣言します。
- **AllowPushBy**
この i アプリ Bookmark が外部インタフェースから起動対象として指定されることを許可することを宣言します。
- **LaunchByFaceApp**
この i アプリ Bookmark が MyFACE から起動対象として指定されることを許可することを宣言します。

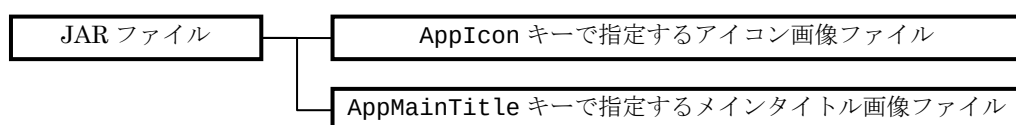
これらのキーの使用方法は i アプリの ADF で利用される同名のキーと同様です。詳細は各キーの説明を参照してください。

以下に、i アプリ BookmarkADF の記述例を示します。i アプリ BookmarkADF は下記に示すとおり、1 つのキーエントリを 1 行で記述しなければなりません。また各行の行末は最終行も含め CR LF (0x0d0a) とします。

```
BookmarkURL = http://www.nttdocomo.co.jp/downloadpage.html
BookmarkAppURL = http://www.nttdocomo.co.jp/iappli.jam
AppName = i アプリ Bookmark
AppIcon = 48x48.gif,96x96.gif
AppMainTitle = 160x160.gif
BookmarkPreinType = Java
BookmarkSize = 5877
PackageURL = http://www.nttdocomo.co.jp/iapplibookmark.jar
```

JAR ファイルの作成

i アプリ Bookmark の JAR ファイルには、i アプリ BookmarkADF の **AppIcon** キーや **AppMainTitle** キーで指定した画像ファイルを格納します。これらのファイルは JAR ファイル内のルートディレクトリの位置に格納します。



i アプリ Bookmark 配布のための Web サーバーへの配置

作成した i アプリ BookmarkADF (.jam ファイル) と JAR ファイル (.jar ファイル) を Web サーバーに配置します。i アプリ Bookmark はブラウザではなく、コンテンツパッケージのダウンロード機能を利用してダウンロードされるため、i アプリを配置する場合とは以下の点で異なります。

- i アプリ Bookmark をダウンロードするための HTML ファイルは不要です。
コンテンツパッケージのダウンロード機能が直接 i アプリ BookmarkADF の取得を行い、その後 i アプリ BookmarkADF の PackageURL キーで指定された URL から JAR ファイルをダウンロードします。
ただし、i アプリ本体をダウンロードするための HTML ファイルは、i アプリ BookmarkADF の BookmarkURL キーで指定する位置に配置する必要があります。また、i アプリ本体の ADF ファイルは、i アプリ BookmarkADF の BookmarkAppURL で指定する位置に配置します。
- i アプリ BookmarkADF の Content-Type を "application/x-iapplibookmark" とします。
Content-type が異なる場合、i アプリ Bookmark をダウンロードすることはできません。特に、i アプリ本体の ADF の Content-Type ("application/x-jam") とは異なることに注意してください。

また、i アプリ BookmarkADF 自身の URL と BookmarkAppURL キーで指定された URL が同一である場合、i アプリ Bookmark をダウンロードすることはできません。

コンテンツパッケージメタファイルの記述

コンテンツパッケージメタファイルは、コンテンツパッケージダウンロード機能において、ダウンロード対象の全コンテンツの名前やダウンロード用 URL、コンテンツ種別などの情報が記述されたファイルです。i アプリ Bookmark もコンテンツパッケージメタファイルの記述内容にしたがってダウンロードされます。コンテンツパッケージメタファイルは、コンテンツパッケージダウンロード機能向けに規定された特定の要素を持つ XML ファイルです。

以下に、i アプリ Bookmark をダウンロードするためのコンテンツパッケージメタファイルの記述例を示します。

```
<launch mode="cont" type="application/x-iapplibookmark">
  <title>i アプリ Bookmark</title>
  <url>http://www.nttdocomo.co.jp/iapplibookmark.jam</url>
  <metasize>310</metasize>
  <size>5877</size>
  <connection>
    <param name="name1" value="value1" />
    <param name="name2" value="value2" />
  </connection>
</launch>
```

i アプリ Bookmark をダウンロードするために使用される各要素は以下の通りです。

- launch 要素
launch 要素はダウンロード対象のコンテンツ種別を指定するために使用されます。i アプリ Bookmark では、launch 要素の mode 属性には "cont" を、type 属性には "application/x-iapplibookmark" を指定します。
- title 要素
title 要素には i アプリ Bookmark の名前を指定します。携帯電話でコンテンツパッケージ内一覧画面を表示させた際に、ここに指定された名前が表示されます。
- url 要素
url 要素には i アプリ BookmarkADF のダウンロード元 URL を指定します。
- metasize 要素
metasize 要素には i アプリ BookmarkADF のサイズをバイト単位で指定します。

- **size 要素**
size 要素には i アプリ Bookmark の JAR ファイルのサイズをバイト単位で指定します。
- **connection 要素**
connection 要素は、コンテンツパッケージからの i アプリ起動時に、`StarApplicationManager.getParameter()` メソッドにより取得できるパラメータを指定する際に使用されます。パラメータ名やパラメータ値は本要素の子要素の param 要素で指定します。
i アプリに渡すパラメータが不要な場合、本要素は省略可能です。
- **param 要素**
param 要素は i アプリに渡すパラメータ名とパラメータ値を指定するために使用されます。param 要素の name 属性にはパラメータ名を、value 属性にはパラメータ値を指定します。

コンテンツパッケージメタファイルの詳細については、別途 NTT ドコモより公開されるコンテンツパッケージ機能の資料を参照してください。なお、コンテンツパッケージメタファイルを作成するためのコンテンツパッケージ作成ツールが NTT ドコモより公開されます。典型的には、i アプリ Bookmark (i アプリ BookmarkADF と i アプリ Bookmark の JAR ファイル) はエミュレータ (i アプリ開発環境) で作成し、コンテンツパッケージメタファイルはコンテンツパッケージ作成ツールで作成します。

16.7.2 i アプリ Bookmark の動作

ダウンロードされた i アプリ Bookmark は、コンテンツパッケージ内一覧画面や i アプリ一覧画面などに表示されます。これらの画面でユーザー操作により i アプリ Bookmark が選択されると、携帯電話はブラウザを起動し、i アプリ BookmarkADF の BookmarkURL キーで指定された i アプリのダウンロードサイトを表示します。

サイトから i アプリがダウンロードされる際、その i アプリの ADF URL と同じ URL が BookmarkAppURL キーに指定されている i アプリ Bookmark が削除され、今回新たにダウンロードした i アプリに置き換えられます。i アプリ Bookmark に代わって携帯電話に保存された i アプリは、次項で説明する、コンテンツパッケージから連携起動が可能であることを除き、通常の i アプリと同様に扱うことができます。

また前項で記載したとおり、ブラウザやメーラなど携帯電話上のネイティブアプリケーションや外部インタフェースから i アプリ Bookmark を起動対象として指定することができます。

ネイティブアプリケーションから i アプリ Bookmark を起動対象として指定するには、ネイティブアプリケーションからの i アプリの連携起動における i アプリの ADF の URL の代わりに、起動対象として指定したい i アプリ Bookmark の BookmarkAppURL キーの値を指定します。ネイティブアプリケーションからの起動指示で指定された URL と同じ URL が BookmarkAppURL キーに指定されている i アプリ Bookmark が携帯電話上にある場合、その i アプリ Bookmark が持つ i アプリのダウンロードサイトが表示されます。対応する i アプリが携帯電話にダウンロードされたことにより既に i アプリ Bookmark が削除されている場合、対応する i アプリが起動されます。

注意事項：

- コンテンツパッケージ内一覧画面や i アプリ一覧画面でユーザー操作により i アプリ Bookmark が選択された場合や、ネイティブアプリケーションや外部インタフェースから指示により i アプリ Bookmark が起動対象になった場合、ブラウザを起動する前に、i アプリ実行環境はユーザーにサイトへアクセスしてよいか確認を行います。ユーザーの許可が得られなかった場合、ダウンロードサイトを表示することはできません。

16.7.3 コンテンツパッケージからの i アプリ起動

i アプリ BookmarkADF の `BookmarkAppURL` キーで指定された値と ADF 取得元 URL が一致する i アプリはコンテンツパッケージから起動することができます。コンテンツパッケージから起動された i アプリの起動種別 (`StarApplication.started()` メソッドの引数に設定される起動種別) は `StarApplication.LAUNCHED_FROM_CONTENTS_PACKAGE` となります。

また、コンテンツパッケージから i アプリを連携起動する際に i アプリにパラメータを渡すこともできます。i アプリに渡すパラメータはコンテンツパッケージメタファイルの `param` 要素で指定します。コンテンツパッケージメタファイルで指定されたパラメータは

`StarApplicationManager.getParameter()` メソッドにより取得することができます。パラメータは最大 16 個指定することができ、パラメータ名とパラメータ値 ("=" を除くパラメータ文字列) の長さの合計は Shift_JIS 換算で最大 20480 バイトまでに制限されます。

注意事項：

- 携帯電話ユーザーは、携帯電話の設定によりネイティブアプリケーションや外部インタフェースからの i アプリ Bookmark の連携起動（ダウンロードサイトの表示）を禁止することができます。またブラウザを起動する前に、i アプリ実行環境はユーザーにサイトへアクセスしてよいか確認を行います。これらの機構によりユーザーの許可が得られなかった場合、ダウンロードサイトを表示することはできません。
- コンテンツパッケージから起動される i アプリはフルアプリ、ミニアプリのいずれでも構いません。なお起動先 i アプリがフル+ミニアプリである場合は、ADF の `AppType` キーの先頭に書かれているほうの i アプリが優先的に起動されます。
- ネイティブアプリケーションや外部インタフェースからの i アプリ Bookmark の連携起動の結果ダウンロードされた i アプリがそのままユーザー操作で起動された場合、起動種別は、`StarApplication.LAUNCHED_AFTER_DOWNLOAD` となります。このとき、ネイティブアプリケーションや外部インタフェースが指定したパラメータは取得できません。
- コンテンツパッケージから起動された i アプリの動作が終了 (`StarApplication.terminate()` により終了) した場合、i アプリを起動する前のコンテンツパッケージ表示状態に復帰します。コンテンツパッケージから起動された i アプリをユーザーが強制終了した場合や i アプリがアプリケーション連携機能によって他のネイティブアプリケーションの機能の起動または呼び出しを行った場合は 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。

第 17 章

i アプリの配布

この章では、i アプリを配布するために Web サーバーをどのように設定する必要があるかについて説明します。

17.1 i アプリ配布のための Web サーバーの構成

前の章で述べたように、i アプリは ADF (.jam ファイル) と JAR ファイル (.jar ファイル) から構成されています。Web サーバーに置かれた ADF をダウンロードするには、i モード HTML の OBJECT タグでこのファイルが参照されていなければなりません。次に、i アプリをダウンロードするための HTML ファイルにおけるタグの記述例を示します。

```
<OBJECT declare id="application.declaration"
data="http://www.nttdocomo.co.jp/java/abc.jam" type="application/x-jam">
<PARAM name="Param1" value="i-mode">
<PARAM name="Param2" value="i アプリ">
</OBJECT>
ダウンロードするには
<A ijam="#application.declaration" href="notapplicable.html">ここ</A>
をクリック。
```

以下に、i アプリのダウンロードのために使用されるタグを解説します。

- A タグおよび ijam 属性

A タグは、ダウンロード対象の i アプリに対応する OBJECT タグを参照するために使用されます。ユーザーがこの A タグを選択することで、i アプリのダウンロード処理が開始されます。

A タグの ijam 属性には、OBJECT タグの id 属性に指定された名前を指定します。A タグに href 属性を併記することで、i アプリ非対応の i モード機種は href 属性を、i アプリ対応機種では ijam 属性を解釈するようになります。また ijam 属性と、ブラウザから i アプリを起動するための ista 属性を同時に有効にすることはできません。ijam 属性と ista 属性を同時に指定した場合、常に ista 属性を解釈し、ijam 属性は無視されます。ブラウザからの i アプリ起動の詳細については、第 11 章を参照してください。

なお、ijam 属性を持つ A タグでは、必ずペアで href 属性を指定するようにしてください。

- OBJECT タグ

OBJECT タグは、i アプリに対応する ADF を参照するために使用されます。OBJECT タグの id 属性には、A タグの ijam 属性で参照される名前 (HTML 内で一意) を指定します。また data 属性には ADF の位置を示す URL を指定します。type 属性には data 属性で示されたデータ (この場合 ADF) のコンテンツタイプを指定します。ADF のコンテンツタイプは "application/x-jam" です。

なお、i アプリの JAR ファイルは、ADF 中に指定された PackageURL キーから参照されます。

OBJECT タグの内部には、下記に示す PARAM タグを記述することができます。

● PARAM タグ

PARAM タグは、i アプリをダウンロードして初回の実行時(*1)に、`StarApplicationManager.getParameter()` メソッドにより取得できるパラメータを指定するために使用されます。

PARAM タグは、OBJECT タグ中に最大 16 個含めることができます。また、name 属性の値の長さと value 属性の値の長さの合計は最大 20480 バイトに制限されます。name 属性の値および value 属性の値には、それぞれ SJIS で日本語テキストを指定することができます。

- (*1) i アプリをダウンロードして初回の実行がアプリケーション連携によるもの（ブラウザ、メール、外部機器および i アプリからの起動）である場合、ここで指定された PARAM タグの内容は i アプリに渡されません。初回実行がアプリケーション連携によるものである場合、起動元となるアプリケーションから渡されるパラメータの方が有効となり起動先の i アプリに渡されます。アプリケーション連携の詳細については第 11 章を参照してください。

i アプリ対応携帯電話は、OBJECT タグの type 属性で ADF のコンテンツタイプを確認します。したがって Web サーバー側では、ADF の拡張子に対応するコンテンツタイプのコンフィグレーションを行う必要はありません。

なお、携帯電話に 100K バイトを超える JAR ファイルをダウンロードできるようにするためには、i アプリの配布に使用する Web サーバーがレンジリクエストによるバイト単位の部分的 GET に対応していなければなりません。レンジリクエストに対して部分的 GET の応答（206 Partial content）を返さない Web サーバーからは、100K バイトを超える JAR ファイルをダウンロードすることはできないため注意してください。

特に、JAR ファイルのダウンロードにおいて CGI を利用しているような場合には注意が必要です。

また、トルカ（詳細）の HTML コンテンツ部分に本項で解説したタグを記述することでトルカビューア経由で i アプリをダウンロードすることもできます。この場合 OBJECT タグ data 属性に記載する ADF URL には相対 URL は指定しないようにしてください。

注意事項：

- Star プロファイルでは、DoJa プロファイルにおけるダウンロード即起動 i アプリをサポートしません。このため Star プロファイルのアプリをダウンロードする際に、ilet 属性が指定されていた場合 i アプリをダウンロードすることはできません。
- MyFACE から i アプリをダウンロードすることはできません。

17.2 サーバー側アプリケーション実行のための Web サーバーの構成

クライアント/サーバー型の i アプリを動作させる場合、サーバー側アプリケーションに適用する技術によっては Web サーバーに特定のコンフィグレーションを行う必要がある場合があります。例えばサーバー側アプリケーションにサーブレットを使用する場合、Web サーバー側ではサーブレットエンジンのインストールとコンフィグレーションを行う必要があります。

使用している Web サーバーのドキュメントおよびサーバー側アプリケーションに適用している技術に関するドキュメントを参照し、Web サーバーに適切なコンフィグレーションを行うようにしてください。

17.3 特定メーカーの i アプリ対応携帯電話へのダウンロード

i アプリオプション機能または i アプリ拡張機能を使用した i アプリや厳密な画面サイズを前提とした i アプリは、特定メーカーの携帯電話でのみ開発者の意図通りに動作します。このような i アプリは、そのターゲットとする機種だけに適切にインストールされることが望まれます。

i アプリのダウンロードを各機種毎に適切にコントロールするために、i アプリ対応携帯電話では次のような機構が利用可能になっています。

1. 携帯電話からの HTTP リクエストには、携帯電話の機種を判別するための **User-Agent** リクエストヘッダが含まれています。Web サーバー側では i アプリのダウンロード用 HTML を表示する際に、**User-Agent** の内容に基づいてユーザーに適切な i アプリを提示するようナビゲートすることができます。
2. 携帯電話側では、ADF の **TargetDevice** キーを参照して、ダウンロードする i アプリがその機種に対応しているかどうかを判定します。特定機種向けの i アプリでは、ADF の **TargetDevice** キーを適切に設定することで、開発者の意図しない機種へのダウンロードを防止することができます。

注意事項：

- Star-1.3 プロファイル向けのエミュレータでビルドした i アプリは、Star-1.2 以前のプロファイルに対応した携帯電話上では動作しません。詳細は 16.2 項の注意事項を参照して下さい。

付録 A

ミニアプリにおける機能制限

ミニアプリ実行環境は複数のミニアプリを同時に実行することができますが、通常の i アプリに比べ利用可能な機能に制限が設けられています。ここでは、ミニアプリ実行環境が i アプリに与える制限について解説します。ミニアプリ実行環境やミニアプリの詳細は、「i アプリコンテンツ開発ガイド 詳細編 第 9 章」を参照してください。

なお、本項で解説する制限によって、全機能がミニアプリ実行環境では使用出来なくなるクラスやインタフェースについては、クラス、インタフェースそのものがミニアプリ実行環境から削除されています。ミニアプリ実行環境においてクラス定義が存在しないクラスについては API リファレンスのトップページからリンクされている注意事項を参照してください。

A.1 ミニアプリで制限される機能

以下の機能は、ミニアプリ実行中はミニアプリの表示状態に関わらず、利用することができません。ミニアプリからこれらの機能を利用しようとした場合、例外が発生します。

- メモリ管理機構へのアクセス
メモリ管理機構へアクセスする API (**MemoryManager** クラス) を利用することができません。
- 高レベル UI コンポーネント
高レベル UI (**Panel**、**Component**) を利用することはできません。
- ダイアログ表示
ダイアログ (**Dialog**) を表示することはできません。
- プレゼンタによる再生
プレゼンタ (**AudioPresenter**、**VisualPresenter** とともに) を利用することはできません。
- i モーション形式データの再生
動画再生 (**NativeMoviePlayer** クラス) することはできません。
- イメージ処理
イメージエンコーダを利用することはできません。
- i アプリ連携起動
連携モードによる i アプリ連携起動機能、ランチャーモードによる i アプリ連携起動機能は利用できません。
- i アプリ更新機能連携起動
i アプリ更新機能を利用することはできません。

- 電話帳管理機能の呼び出し
電話帳グループの新規登録機能、電話帳エントリの新規登録機能を利用することはできません。
- ブックマーク管理機能の呼び出し
ブックマークの新規登録機能を利用することはできません。
- スケジュール管理機能の呼び出し
スケジュールの新規登録機能を利用することはできません。
- 画像管理機能の呼び出し
画像データの新規登録機能、画像データの選択読み込み機能、画像データの ID 指定読み込み機能を利用することはできません。
- カメラ機能の呼び出し
カメラ機能、コード認識機能を利用することはできません。
- 映像データ管理機能の呼び出し
映像データの新規登録機能を利用することはできません。
- データ BOX フォルダ管理機能の呼び出し
フォルダの新規登録機能、フォルダのエントリ ID の選択取得機能を利用することはできません。
- コンテンツ取得保存機能の呼び出し
コンテンツ取得保存機能を利用することはできません。
- 3D グラフィックス・3D サウンド
3D グラフィックス描画機能、3D サウンド制御機能を利用することはできません。
- 内部ストレージ機能
内部ストレージ機能を利用することはできません。
- ユーティリティ
デジタル署名 API を利用することはできません。

A.2 i アプリ API 利用上の注意点

A.2.1 ユーザーインタフェース

(1) 利用可能なマルチメディアデータ

ミニアプリから利用可能なマルチメディアデータは以下のデータのみに制限されます。

- GIF 形式画像データ
- JPEG 形式画像データ
- BMP 形式画像データ
- PNG 形式画像データ

これら以外の、MFi 形式サウンドデータ、SMF サウンドデータ、i モーション形式データ、SWF 形式画像データは使用できません。

なお、ミニアプリではプレゼンタを利用することができないため、GIF 形式画像データのうち、アニメーション GIF を再生することはできません。

(2) ミニアプリの描画領域

ミニアプリが **Canvas** に描画した内容は ADF の **Pallet** キーの内容に従いパレット上に表示されます。**Pallet** キーで指定可能なパレットのサイズに制限があるため、ミニアプリでは全画面表示することはできません。また、**Canvas.getWidth()**メソッド、**getHeight()**メソッドで取得する値、**Canvas** の描画原点についても **Pallet** キーで指定された値に従います。

(3) キーとキーイベント

表示状態が **Selected** 状態以外のミニアプリがキーパッド状態を取得するメソッド

(**Canvas.getKeypadState()**メソッド) を呼び出した場合、常に **0** (どのキーも押されていないことを表す) が返ります。また、**Selected** 状態以外のミニアプリにキーイベント、ファンクションキーイベントが通知されることはありません。

表示状態が **Selected** 状態であるミニアプリにおいては i アプリが取得可能なキーイベントのうち、クリアキー、ファンクションキー3、ファンクションキー4 以外のキーのイベントを取得することができます。クリアキー、ファンクションキー3、ファンクションキー4 は **WidgetView** の操作にアサインされるため、これらのキーのイベントはミニアプリには通知されません。

(4) ファンクションキー/決定キーのラベル設定、上下左右キー矢印表示設定

表示状態が **Selected** 状態以外のミニアプリがファンクションキー1、ファンクションキー2、決定キーのラベル文字列設定 (**Canvas.setSoftLabel()**メソッド)、および上下左右キー矢印表示設定 (**Canvas.setSoftArrowLabel()**メソッド) を呼び出した場合、設定した内容は画面に反映されません。このミニアプリの表示状態が **Selected** 状態に遷移し、表示設定を行った **Canvas** がカレントフレームに設定されたときに表示設定された内容に従い画面に反映されます。

表示状態が **Selected** 状態であるミニアプリがファンクションキー1、ファンクションキー2、決定キーのラベル文字列設定、および上下左右キー矢印表示設定を呼び出した場合、設定した内容は画面に反映されます。

また、ミニアプリ実行中はその表示状態によらず **Canvas.setSoftLabelVisible()**メソッドを呼び出した場合、例外が発生します。

(5) IME(Input Method Editor)起動

表示状態が **Selected** 状態以外のミニアプリが IME を起動しようとした場合、例外が発生します。

A.2.2 通信の制御

複数のミニアプリから同時に接続要求(**HttpConnection.connect()**メソッド)することはできますが、同時に接続可能な **HttpConnection** の数はミニアプリ全体で 1 つだけです。1 つのミニアプリがすでに HTTP 通信中の場合、別のミニアプリからの接続要求は待ち状態に入り、メソッドはブロックされます。実行中の HTTP 通信が終了すると、待ち状態の接続要求のうち 1 つが選択され、それに対する HTTP 通信が開始されます。

待ち状態に入って **connect()**メソッドがブロックしている状態で他のスレッドからその **HttpConnection** オブジェクトに対して **close()**メソッドを呼び出すことで、その HTTP 通信要求の待ち状態を解除し **connect()**メソッドを中断することができます。

【Star-1.3】

Star-1.3 プロファイル以降では、複数のミニアプリから同時に HTTP(S)通信を行うことが可能となる機種があります。ただしこの機能はオプションであり、同時に HTTP(S)通信が行えるかどうかや、行える場合でも同時にいくつのミニアプリから HTTP(S)通信が可能かはメーカーにより異なります。また複数の HTTP 通信を同時に行える機種でも、同じ数の HTTPS 通信を同時に行えるとは限りません。なおこの機能をサポートする場合でも、同じミニアプリから 2 つ以上の HTTP(S)通信を同時に行うことはできません。

【Star-2.0】

Star-2.0 プロファイル以降、フルアプリからの HTTP(S)通信で受信可能なレスポンスボディのサイズは 1024K バイトですが、ミニアプリからの HTTP(S)通信で受信可能なレスポンスボディのサイズは 150K バイトです。

A.2.3 ScratchPad へのアクセス

ミニアプリの表示状態によらず ScratchPad にアクセスすることはできますが、ミニアプリでは ADF の SPSIZE キーで指定された 1 つ目の ScratchPad ("scratchpad:///0") のみアクセスが許可されています。また利用可能な ScratchPad の最大サイズは 200K バイトに制限されています。

フル+ミニアプリでは、それを構成するフルアプリとミニアプリが SPSIZE キーで定義される ScratchPad を共有します。ただし、そのためには

- ScratchPad を分割しない場合 ScratchPad のサイズを 200K バイト以下にする。
- ScratchPad を分割する場合 1 つ目の ScratchPad のサイズは 200K バイト以下にする。

のいずれかの条件を満たす必要があります。

ミニアプリが ADF の SPSIZE キーの 2 番目以降の ScratchPad を指定した場合や利用可能な最大サイズを超える ScratchPad を利用しようとした場合、ScratchPad のオープン処理で例外が発生します。

A.2.4 ローカルリソースへのアクセス

フル+ミニアプリにおいて、ミニアプリからはミニアプリ用 JAR ファイル内に同梱されているリソースのみ利用可能です。なお、フルアプリからは、フルアプリ用 JAR ファイル、ミニアプリ用 JAR ファイルの双方の JAR ファイルに同梱されるリソースを利用することができます。

フルアプリがローカルリソースを使用する際には、まずフルアプリ用 JAR ファイルを検索し、フルアプリ用 JAR ファイル内でリソースが見つかった場合はそのリソースをフルアプリに返します。フルアプリ用 JAR ファイル内に該当するリソースが見つからなかった場合はミニアプリ用 JAR ファイルを検索します。

A.2.5 プラットフォームリソースのアクセス

ミニアプリから制御可能なプラットフォームリソースを以下に示します。

- 機体の折りたたみ状態（折りたたみ型、フリップ型の携帯電話のみ有効）
- メール・メッセージの受信状態
- 電池パック残量
- 電波状態

- エリア情報
- マナーモード設定状況
- 画面可視・不可視状態

これらの属性は全て `com.docomostar.system.PhoneSystem` クラスで定義されており属性値の参照のみ許可されています。

A.2.6 アプリケーション連携

(1) ブラウザ連携起動

表示状態が `Selected` 状態以外のミニアプリが、`Launcher.launch()` メソッドの引数に `Launch.LAUNCH_BROWSER` を指定してブラウザを起動しようとした場合、例外が発生します。

なお、ブラウザを起動する際に `i` アプリを終了させずサスペンド状態を継続する機能 (`Launcher.LAUNCH_BROWSER_SUSPEND`) をサポートする携帯電話がありますが、この機能をサポートする携帯電話であってもミニアプリからこの機能を利用しようとした場合ミニアプリの表示状態によらず例外が発生します。

(2) 同一パッケージ内での `i` アプリ間連携起動

表示状態が `Selected` 状態以外のミニアプリが、同一パッケージ内での `i` アプリ間連携によりフルアプリを起動しようとした場合、例外が発生します。

(3) 通話機能の呼び出し

表示状態が `Selected` 状態以外のミニアプリが通話発信機能を呼び出した場合、例外が発生します。

(4) スケジュール管理機能の呼び出し

表示状態が `Selected` 状態以外のミニアプリがスケジューラを起動しようとした場合、例外が発生します。

(5) アラーム機能の呼び出し

表示状態が `Selected` 状態以外のミニアプリが、アラーム時刻を登録しようとした場合、例外が発生します。

A.2.7 赤外線リモコン

表示状態が `Selected` 状態以外のミニアプリが赤外線リモコン信号を送信しようとした場合、例外が発生します。

表示状態が `Selected` 状態であるミニアプリが赤外線リモコン信号を送信している状態で、ミニアプリの表示状態が `Selected` 状態以外の状態に遷移した場合、信号の送信は停止されます。

注意事項：

- ミニアプリからの赤外線リモコン機能の利用はオプションであり、メーカーによってはサポートされない場合があります。

付録 B

2in1 機能が i アプリに与える影響

904i シリーズ以降の携帯電話では、1 台の携帯電話に 2 つの電話番号、2 つの i モードメールアドレスを割り当てて、ユーザーの利用シーンによって自由に使い分けを行うことができる、2in1 機能と呼ばれる機能を利用できるものがあります(2in1 機能はハイパーマルチナンバー機能、もしくはハイパーマルチ機能と称される場合もあります)。

2in1 機能を利用するためには、ユーザーは NTT ドコモと、通常回線契約に加え 2in1 機能を利用するための契約を行う必要があります。この契約が行われると、ユーザーは通常回線契約での電話番号や i モードメールアドレス (A ナンバー/A アドレス) の他にもう一組の電話番号、i モードメールアドレス (B ナンバー/B アドレス) を利用できるようになります。

2in1 機能は携帯電話の通話・メール機能の動作や、電話帳・発着信履歴などのユーザーデータ管理に強く関わる機能であり、2in1 機能が有効となるよう設定されている携帯電話では一部の i アプリ API に影響を及ぼす場合もあります。本付録では、2in1 機能が i アプリに与える影響について解説します。

なお、携帯電話上で 2in1 機能が有効となるよう設定された状態を活性化状態、無効となるよう設定された状態を非活性化状態と呼びます。また活性化状態には、ユーザーデータの閲覧を制御するためのモードの概念が導入されています。活性化状態中のモードには、以下の 3 つがあります。

- A モード : A ナンバー/A アドレスに関連するユーザーデータを閲覧できるモード
- B モード : B ナンバー/B アドレスに関連するユーザーデータを閲覧できるモード
- デュアルモード : A、B のどちらに関連するユーザーデータも閲覧できるモード (共通モードと呼ばれることもあります)

2in1 活性化 A モードでは、B モードでの発着信履歴や B アドレス宛てのメール、B モード用として属性をつけて保存された電話帳データの閲覧はできません。また 2in1 活性化 B モードでは、A モードでの発着信履歴や A アドレス宛てのメール、A モード用として属性をつけて保存された電話帳データの閲覧はできません。

なお現在の 2in1 サービスでは、B アドレス宛てのメールは通常の i モードメールの形で携帯電話に送られてくるのではなく、Web メール形式で閲覧する形となります。B アドレスにメールが届いたことは、B アドレス宛てとして携帯電話に送付される着信通知メール (i モードメール) にてユーザーに通知されます。

また 2in1 活性化 B モードでは、携帯電話から i モードメールおよび SMS の送信は行えません。
2in1 活性化 B モードでは、メール送信については閲覧時と同様に Web メール形式で行います。

<2in1 機能が i アプリに与える影響>

- 状態やモードによるユーザーデータの登録制限について

i アプリから `PhoneBook.addEntry()` メソッドを使用して電話帳エントリを登録する場合、2in1 非活性化状態、2in1 活性化デュアルモード、2in1 活性化 A モードでは、その電話帳エントリは A モード用の (B モードでは電話帳のデータとして見るできない) 電話帳データとして保存されます。また、2in1 活性化 B モードでは、その電話帳エントリは B モード用の電話帳データとして保存されます。

なお、上記のように保存された電話帳エントリも、携帯電話ネイティブの設定機能をユーザーが操作することで、対応するモードを変更したり、どのようなモードでも利用可能とするよう設定したりすることができます。

- `StarApplicationManager.getSuspendInfo()` メソッドによるサスペンド理由の取得について

`StarApplicationManager.getSuspendInfo()` メソッドを使用することで、直前に発生したサスペンドの理由や、そのサスペンド期間中に発生した事象の情報を取得することができます。サスペンド理由や事象には「通話着信」「メール着信」といった着信系のイベントも含まれますが、2in1 活性化 A モードでは B ナンバー/B アドレスの着信イベント、また 2in1 活性化 B モードでは A ナンバー/A アドレスの着信イベントについて、このメソッドでサスペンド理由などを取得することはできません。