i-αppli Content Developer's Guide for DoJa-5.x/5.x LE

Functional Descriptions

Version 3

May 27th, 2008
NTT DOCOMO, INC.

This document targets the Java Application Runtime Environment compatible with the DoJa-5.x Profile (referred to as i-αppli Runtime Environment in this document). The i-áppli specification can be identified by its profile version. The profile version evolution up to this point and the major mobile phone models included are as follows.

DoJa-1.0 Profile:	Digital mova 503i series and the initial FOMA mobile phone model
DoJa-2.0 Profile:	mova 504i series
DoJa-2.1 Profile:	FOMA mobile phone models sold in the winter of 2002 and later
DoJa-2.2 Profile:	FOMA mobile phone models sold in the spring of 2003 and later
DoJa-3.0 Profile:	mova 505i series
DoJa-3.5 Profile:	FOMA 900i series
DoJa-4.0 Profile:	FOMA 901i series
DoJa-4.1 Profile:	FOMA 902i series
DoJa-5.0 Profile:	FOMA 903i series, FOMA 904i series
DoJa-5.1 Profile:	FOMA 905i series, FOMA 906i series

In addition, the DoJa-5.x Profile includes the DoJa-5.x LE (Limited Edition) Profile as a derived specification in which some functions are omitted for entry models. The difference in specification between the DoJa-5.x Profile and the DoJa-5.x LE Profile is described in an appendix in this document.

The profile version of the i-αppli Runtime Environment installed in each device will be announced by NTT DOCOMO separately.

Please be aware that, even in the case of functions existing from previous profiles, the content of this document does not apply to mobile phones which have support only for earlier profiles. The profile described in this guide contains clarification of undefined specifications for pre-existing functions.

Contents

The Purpose of this Document

The "i-appli Content Developer's Guide" contains information required for the design and development of Java-based applications, called i-appli, which provide new services for i-mode mobile phones and is provided to i-mode content providers. The guide's objective is to provide a simple model for creating and deploying an i-áppli application. Furthermore, this guide also contains details about processes developers should adhere to in order to create intuitive, easy to use i-appli in addition to information related to this technology. This preface provides an overview to the Developer's Guide and contains the following topics:

- Intended Reader
- How to Use this Guide
- Notation Rules
- Related Documents

Intended Reader

The "i-appli Content Developer's Guide" contains information targeted for developers who intend on creating new services utilizing i-appli technology. Therefore, it is necessary for service developers, server and client program programmers, and service implementation directors to read this guide. Reader knowledge of the following is assumed for this guide.

- Basic knowledge of network applications using the Java programming language.
- Basic knowledge of the specifications of i-mode mobile phones where developed i-áppli will run.

How to Use this Guide

The following table shows where to find specific information.

Details	Refer to
i-αppli Runtime Environment.	Chapters 1 – 2
Design considerations	Chapter 3
Guide to specifications	Chapters 4 – 14
Development process	Chapters 15 – 16

Notation Rules

The following table describes the typographical conventions this guide uses.

Convention	Description
Courier	Courier text denotes code examples, classes, interfaces, methods, and data types.
Bold text	Bold text denotes menu and softkey options.
[ ]	In explanations of syntax etc., items surrounded by brackets ([ ]) indicate options. In actual designations, these can be set or omitted as needed.
< >	In explanations of syntax etc., items surrounded by angle brackets (< >) indicate places that would be replaced by setting the appropriate value.

In this guide, considerations specific to the DoJa-3.0 and DoJa-4.0 profiles are indicated by [DoJa-3.0] and [DoJa-4.0]. Unless otherwise noted, considerations specific to the older profiles are applicable in the new profiles.

Related Documents

Documents related to this guide are shown below.

i-αppli Content Developer's Guide for DoJa-5.x/5.x LE: API Reference
i-αppli Content Developer's Guide for DoJa-5.x/5.x LE: i-appli Options/i-appli Extensions
i-αppli Content Developer's Guide for DoJa-5.x/5.x LE: Optional/Extension API Reference
i-αppli Development Kit for DoJa-5.0 APIs User's Guide
Java 2 Platform, Micro Edition (J2ME) Connected Limited Device Configuration (CLDC) specification (http://java.sun.com/products/cldc/)

For profiles that are DoJa-4.0 or later, the version of CLDC is 1.1. For profiles that are DoJa-3.5 or before, the version of CLDC is 1.0.

Please note that the above indicated URL may be changed without notice.

Chapter 1 Introduction

An i-αppli-ready mobile phone combines telephone access, Internet access, and the ability to download and run Java applications (i-αppli) from any i-mode content provider's Web site. This document provides programming guidelines for software engineers developing i-αppli.

i-appli compatible mobile phones contain support for the Java 2 Platform Micro Edition (J2ME) Connected, Limited Device Configuration (CLDC). The J2ME CLDC defines a minimum required complement of Java virtual machine specifications and class libraries for small network-connected devices. The CLDC is based on the K Virtual Machine (KVM). The KVM is a compact, highly portable Java virtual machine specifically designed to run on memory-/CPU-/power-constrained devices like the mobile phone (unless specifically noted, the term "mobile phone" is used in this document to refer to the i-αppli-ready mobile phone). As a member of the Java 2 family, KVM shares much of the infrastructure of the larger editions of the Java runtime environment; but in order to meet the requirements of small devices, it is optimized for these devices' constraints. The J2ME/CLDC platform, though compact, still provides the Java virtual machine, associated libraries, and APIs DOCOMO content providers need in order to make rich, exciting i-αpplis that the i-mode user can install on demand.

The write-once, run-anywhere aspect of Java means that you, the content provider, can write your i-αppli once and easily run it on any mobile phone from any manufacturer, regardless of CPU or operating system. Java provides a method for transferring applications across the Internet in secure ways, so that i-mode users can download, and install i-αpplis whenever they want, just by connecting to an i-mode content provider's Web site. Using Java, i-mode content developers can develop a wide range of applications, including games and robust e-commerce services. Implementing Java in i-mode mobile phone allows you to deploy newer, more dynamic services.

This chapter will discuss the overall features and functions of the application runtime environment (the i-appli Runtime Environment) on an i-appli enabled mobile phone.

1.1 Advantages of the i-appli Runtime Environment

As a Java content provider, you will write i-αppli in the Java programming language. The i-appli Runtime Environment provides an API (the i-appli API) to control processes such as those involving the mobile phone's user interface, communications, text conversion, graphics, multimedia, and data storage. The i-appli API is made up of three different categories of API groups.

- αppli Standard APIs

  These APIs and their operations are designated with common specifications and are installed as standard on all models.

- αppli Optional APIs

  Although these APIs and their operations are designated with common specifications, it is the responsibility of the manufacturer to decide whether or not they will be installed. When using this category of API, it must be presumed that they will not be installed on certain models.

- αppli Extension APIs

  Even in cases where identical functions are being realized, there is a possibility that differences may exist in these APIs and their operations depend on the manufacturer in question. When using this category of API, it must be presumed that they will not be installed on certain models, and also that usage methods may vary from model to model. (Even in the case of models with identical functionality, there is no guarantee of compatibility at the source code level.)

[DoJa-2.0]

The classification of API categories for the i-αppli Runtime Environment differs from that for DoJa-1.0 Profile. Details regarding all APIs from the above-mentioned three categories are published in the i-αppli Content Developer's Guide for DoJa-2.0 Profile or later.

Note that this document, the i-αppli Content Developer's Guide, deals mainly with explanations of J2ME/CLDC APIs and the i-αppli standard APIs. For details regarding the i-appli optional APIs and the i-appli extension APIs, refer to the "i-αppli Content Developer's Guide: i-αppli Options and Extensions", and the "i-αppli Content Developer's Guide: Optional and Extension API Reference".

The i-appli Runtime Environment has the following functions.

- Dynamic delivery of i-αpplis
- Java Application Manager (JAM)
- A simple sandbox model security feature. This model provides only safe Java API sets such as the java.lang or java.util packages to developers.

The i-αppli Runtime Environment consists of four layers of APIs.

- APIs defined by J2ME/CLDC
- i-αppli APIs (i.e., i-αppli standard APIs)
- i-αppli APIs (i.e., i-αppli optional APIs)
- i-αppli APIs (i.e., i-αppli extension APIs)

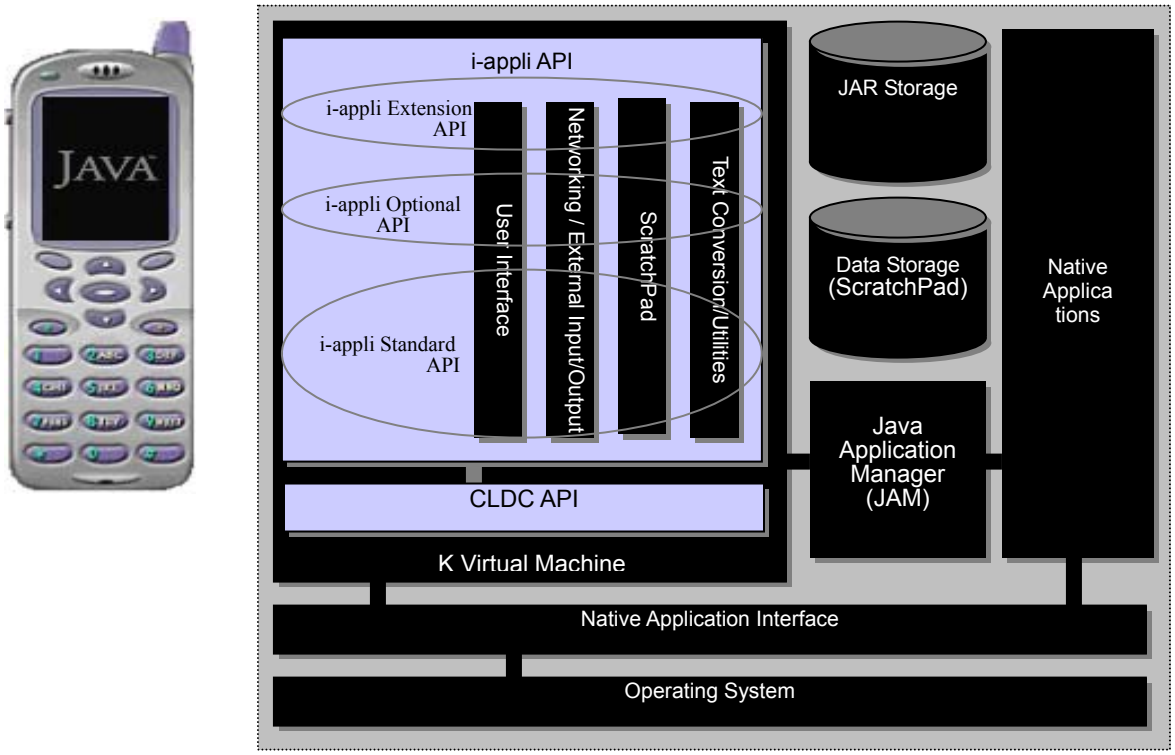


Figure 1: The i-αppli Runtime Environment on an i-αppli compatible mobile phone

Notes:

- Most i-αpplis will use only the J2ME/CLDC and the i-αppli standard API. However, certain specialized i-applis may use the i-appli optional APIs or i-appli extension APIs to enhance their impact or visual style for the user. Please be aware that, depending on the manufacturer of the phone in question, such i-αpplis which use these APIs may not run at all on certain phone models.
- The i-αppli optional and extension APIs include items which may be adopted in the i-αppli standard APIs of future profiles. Conversely, it also includes items which may be removed in future profiles.

J2ME/CLDC is a version of the standard Java platform for compact network connection devices with limited resources. Such devices have the following common features.

- A small amount of total memory.
- Power consumption is limited. Usually operating on battery power.
- Connected to some type of network. Often connected to a wireless network in which connections can be lost easily; bandwidth is also limited.
- Need for a user interface optimized for the service provided by devices.

Mobile phones, two-way pagers, personal digital assistants, home appliances, and point-of-sale terminals are typical devices that are supported by the CLDC specification.

J2ME/CLDC is a stable application platform and is implemented in the i-appli Runtime Environment. Like the larger Java editions, J2ME aims at maintaining the qualities for which Java technology has become known:

- Built-in consistency across products in terms of running anywhere, any time, on any device.
- High-level object-oriented programming language with a large developer base and development tools.
- Portability of code.
- Safe network application delivery.
- Upward scalability with J2SE and J2EE.

The i-appli Runtime Environment, which is built on the CLDC, contains APIs which offer the following kinds of functionality.

- Network connectivity using HTTP and HTTPS.
- Components for defining user interface elements.
- Low-level graphics controls.
- ScratchPad used for saving data.
- Stand-by application control which makes i-αpplis resident on mobile phones in stand-by mode.
- Linked operation between i-αppli and other applications (browser, mailer, i-αppli, etc).
- External connection using the infra-red port (IrOBEX, or infrared remote control).

[DoJa-2.0]

Stand-by applications, IrOBEX external connection, and native application linking represent functions which are newly installed with DoJa-2.0 Profile. Detailed descriptions of these functions are provided in Chapter 9, Chapter 10, and Chapter 11 respectively.

[DoJa-3.0]

The infrared remote control feature and application linking between i-αppli are newly introduced features of DoJa-3.0 Profile. The infrared remote control feature will be explained in detail in Chapter 12.

1.1.1 Optimized Support for Standalone and Client-Server Applications

i-αppli services support programming of both stand-alone and client/server configurations.

Standalone i-applis contain all application files and data on the mobile phone itself; there is no need for any operations involving a server. Many games, calculators, and utilities can be written as standalone i-αpplis.

Standalone i-applis do not need to connect to a server to fulfill the purpose of that particular i-appli; however, it will need to connect to a server when it is downloaded to a mobile phone. Chapter 16 discusses issues related to making your i-αppli available to mobile phones over the Internet.

While of course standalone i-applis are useful in their own right, the most popular i-applis are usually client-server type applications. Client-server i-applis can utilize the powerful processing and data mining capabilities of servers to extend the limited processing power of the mobile phone platform.

The i-appli Runtime Environment includes APIs to establish network connections based on the HTTP and HTTPS protocols, which can in turn be used by the i-appli to connect to the content provider's server through the Internet. It also includes an IrOBEX API for using 2 physically close cellular phones as clients and/or servers via the infrared port.

HTTP(S) Connections

For i-αpplis which use HTTP(S) connection, the processing capabilities of the client computer (in this case, a mobile phone) can be combined with the processing capabilities and/or data access functionality of a remote server. Mobile phones are limited in terms of processing power and memory capacity; however, they can function as the front-end processors for servers with extensive memory and powerful processing capabilities. Mobile phones have limited data input capabilities and display sizes, but if you design and create your i-applis with the proper design and programming methodologies you can allow i-appli compatible mobile phone users to access information, perform online business transactions, and participate in other remote services with ease.

The main purpose of the server in client-server i-applis is to search for the requested data in a databse, and return the results of that search to the client which requested that information. In certain cases, however, where this is unsuitable or impossible as a result of restrictions in the client’s processing power or memory capacity, the server may also carry out a portion of the calculation processing.

For more details regarding the creation of client-server i-αpplis using HTTP(S) connection, refer to Chapter 5.

IrOBEX External Connections

IrOBEX external connection (hereinafter referred to simply as OBEX) realizes a simple client-server model where the infrared ports integrated into mobile phones are used to carry out the communication of data over short distances between these phones. Containing both client and server APIs for OBEX external connections, the i-αppli Runtime Environment provides support for the communication of data between mobile phones. This functionality can be used to create business-card exchange applications, schedule exchange applications, and other similar i-αpplis which transfer small amounts of data between mobile phones.

The hardware specifications of the connecting device and the surrounding environment may make it impossible to carry out the communication of data using OBEX external connection methods.

Note that when not expressly noted in the following section, the terms “client” and “server” will refer to situations where both the client and server can communicate using HTTP(S). For OBEX, these will be referred to as the OBEX client and the OBEX server.

1.1.2 Support for Local Storage of Persistent Data

The i-appli Runtime Environment supports a local storage mechanism called a ScratchPad. Under this mechanism the storage area for persistent data (data which remains stored even after the i-appli has finished running) is allocated within the mobile phone's internal storage device. If a standalone i-appli needs to store persistent data it would use the ScratchPad for this task, but client-server i-applis often store only a portion of such data to the ScratchPad, opting to store the rest of the data on the server.

On the server side, persistent data is almost always stored in a database. Client data is sent to the server via an HTTP request, which interacts via a servlet or a CGI script. The servlet or CGI script then acts as a middle-man between the HTTP request and the database access protocols. This means there will always be an HTTP server on the server side.

Consider the nature of the persistent data before you decide where to store it. Store persistent data using the ScratchPad only if the user can afford to lose it; otherwise, store it on the server side. This is because data stored using the ScratchPad could be lost due to hardware failure or other problems with the mobile phone.

The Java Application Manager (JAM: see the next section for details) is one of the security features of i-appli compatible mobile phones. Under JAM, the areas of ScratchPad memory allocated for each

individual i-appli cannot be accessed by other i-applis. Futhermore, only one i-appli is allowed to run on the mobile phone at once. Multiple i-applis are not allowed to share data in the application runtime memory space. If you need to share data between multiple i-applies, you would need to set up a server to act as a storage space for such shared i-appli data.

1.2 The Java Application Manager

A mobile phone implementing the i-αppli Runtime Environment provides a component for managing the Java i-αpplis installed on that mobile phone. This component is called the Java Application Manager (JAM), and it is responsible for:

- Displaying existing i-αpplis stored on the mobile phone.
- i-appli runtime management (launching/forced termination, mediating between the i-appli Runtime Environment and other applications, etc.).
- Installing and updating i-αpplis.
- Deleting i-αpplis stored on the mobile phone. Once an i-appli has been saved on a mobile phone, it can only be deleted through the specific instruction of the user to do so.

JAM is an independent native component from the KVM, and cannot be controlled from the i-αppli program logic (hereafter called the application program).

Should an i-αppli fail to terminate after the user quits out of it, the user can instruct JAM to force termination by pressing the mobile phone's Hang-Up key.

[DoJa-2.0]

With DoJa-2.0 Profile or later, the forced termination of i-αpplis is incorporated into the Hang-Up key. Note that with DoJa-1.0 Profile, key assignments for forced termination depend on the manufacturer in question.

1.3 The i-αppli Mobile Phone Java Security Model

In addition to the security features inherent to CLDC, the Java Application Manager (JAM) provides the following security features specifically for the i-αppli Runtime Environment:

- Only one i-αppli can run at a time on a mobile phone. This ensures that one i-αppli cannot interfere with or access data from another. Nevertheless, KVM supports multi-threading, thus allowing multi-threaded programming where communication and user-interface processing can both be carried out in parallel within a single i-αppli.
- JAM is a native component independent of the KVM and as such this cannot be controlled from an i-appli. The JAM manages the downloading of i-αpplis. The result is that the mobile phone protects the user from abusive or buggy behavior caused by downloaded i-αpplis.
- i-αpplis only have limited access to storage on the mobile phone (the ScratchPad). This means that no i-αppli is authorized to access the ScratchPad memory allocated to another i-αppli. As such, this prevents a downloaded i-appli from performing any illegal activities such as collecting personal information saved on the ScratchPad for another i-appli without the user's knowledge and transferring it to another server or overwriting it with false information.
- i-applis cannot access native data which contains personal information such as the phone book. Furthermore, in order to prevent web browsing and voice-call transmission against the wishes of the user, the confirmation of the user will always be sought upon API calling for functions which access other applications on the mobile phone such as browsers or dialers. This allows the user to feel safe when downloading and executing a new i-appli.

  [DoJa-2.0]

  As a result of security issues, no support for linked operation between i-αpplis and other applications was provided with DoJa-1.0 Profile. In the case of DoJa-2.0 or later, whenever linked operation is to be carried out between these types of applications, a process is included which will always seek the user's approval, thus ensuring support for safe application linking by users.

- The i-appli Runtime Environment's networking functions provide an easy to use interface for HTTP and/or HTTPS communication supported by i-appli compatible mobile phones. i-applis can only communicate with the server that particular i-appli was downloaded from. This network security feature prevents a potentially malicious i-appli from sending information to a different server unknown to the user.

[DoJa-3.0]

Since Doja-3.0 Profile a new system is included in which certain certified i-appli have reduced restrictions in regards to this security feature. These i-appli are called trusted i-appli and go by the service name "i-appli DX". Trusted i-applis can operate under a relaxed set of security conditions in which they may access native data such as the phone book or communicate with servers other than the one they were downloaded from. However, even trusted i-appli are not granted full access to data which may reveal a user's identity, such as that found in the phone book. In order to prevent such information from leaving the mobile phone access to this kind of data is subject to certain restrictions.

These features (trusted API) can only be used in the services listed on the i-mode menu registered on the iMode server. Other i-applis may not use the trusted API functions.

This document does not go into detail about trusted i-applis.

1.4 Installation and Life Cycle of an i-appli

The only way to run an i-αppli on a mobile phone is to download (install) it from a content provider’s i-mode compatible Web site. Once an i-appli has been downloaded, the JAM will allocate all the memory required to store it. Both an i-appli's JAR file and ScratchPad will be stored in the mobile phone's nonvolatile memory.

The next section will discuss the life cycle of an i-apply, from downloading and execution to deletion.

1.4.1 Downloading the i-αppli

The general procedure for downloading i-αpplis is given below. Note that some details of how a given mobile phone will carry out a given action are manufacturer-specific.

1. The user locates the desired i-αppli’s download link using the i-mode browser.

The i-appli download will only begin when triggered by an action in the i-mode browser. The Web page (HTML content) for the download typically displays a description of the i-αppli and a link that the user clicks to start the download process. The link contains a reference to an application descriptor file (a file with a “.jam” extension; also called an ADF or JAM file. In this document, they will be called ADFs). The ADF is a text file in SJIS encoding, and records information about the i-αppli associated with that ADF with one key and value pair on each line.

One purpose of this file is to allow the JAM to decide whether the selected i-αppli can be successfully downloaded to the mobile phone. By determining whether or not the i-appli can be downloaded successfully beforehand, we can eliminate the cost of actually downloading the i-appli to the mobile phone. The ADF is small (around 100 – 300 bytes), while an i-appli is anywhere from several kilobytes to several tens of kilobytes. So it is advantageous in terms of communication costs to download the ADF and check for available space and compatibility before downloading the entire i-appli.

Section 15.5.1 details the contents of the ADF, and Chapter 16 describes the Web server standards for delivering i-αpplis to mobile phones.

2. The user clicks the link to start the installation process.

This user operation makes the browser pass the URL showing the location of the ADF to the JAM. From this point, the JAM will handle both the downloading and installation of the i-appli.

3. Inspect the contents of the ADF and determine whether to allow or deny the installation of the i-appli.

The JAM will check to ensure that the designated ADF has specified all of the required keys. Then, based on the contents of the ADF (for example, the JAR file size, ScratchPad size, target model information, etc.) the JAM will check to see if

the i-appli can be installed on that particular mobile phone. If there are no problems at this stage, the process continues to the next step.

In the following cases, the i-appli will be considered to be already installed on the mobile phone and the i-appli cannot be newly installed. (However, as shown in Section 1.4.3, the i-appli can be updated.)

- When an i-appli with the same URL and ADF as the i-appli you are attempting to install is already installed on the mobile phone (From DoJa-3.0 Profile onward, if an instruction is given to download this type of i-appli the mobile phone will search for updates for that i-appli and switch to the i-appli's update process if an update is possible.)
- When an i-αppli is already installed on the mobile phone with the same name (ADF AppName key) and a JAR file URL (ADF PackageURL key) corresponds to the ADF of a URL different from the i-αppli to be installed.

4. The i-appli will then be downloaded.

If the JAM decides that the i-αppli can be installed on the mobile phone, it obtains the JAR file's URL using the PackageURL key (if the PackageURL key is a relative URL, the ADF location is used as the base URL) and starts the HTTP download process. If a network error occurs or the user decides to cancel during the download process, the JAM will delete the partially downloaded i-appli and the mobile phone will be reverted back to the state it was in before the download process began.

5. The JAM then updates the list of installed i-αpplis.

The JAM will add the i-appli that was installed as described in the previous section to the list of installed i-applis. It will also store the ADF data and JAR file in nonvolatile memory.

If the i-appli is configured in the ADF to use HTTP communication or some other function which requires the user's permission, the user will be asked whether or not to allow the usage of these functions at this stage. Furthermore, if the ADF settings are such that i-αpplis can operate as a stand-by application, then the user will be asked whether or not the application is to be registered as such.

The i-αppli will be installed into the mobile phone during the above steps 1 through 5, and it will then be ready for use (i.e., execution).

[DoJa-3.0]

From DoJa-3.0 Profile onward, functionality to launch an i-appli immediately upon downloading is supported in order to enable one-click downloading and launching of i-applis. (This type of i-appli is referred to as a "download and immediate launch i-appli".) Because i-αppli that launch immediately after download require no user confirmation, they are restricted from using functions which require user confirmation.

For details regarding i-αppli that launch immediately after download, see chapter 16.

[DoJa-5.0]

From DoJa-5.0 Profile onward, i-applis can be downloaded through the ToruCa Viewer on mobile phones which support ToruCa's version 2.0 format by placing an i-appli download tag (see Chapter 16) in the HTML content portion of the ToruCa (details).

1.4.2 Launching the i-αppli

The procedure for launching an i-αppli and the user interface are manufacturer-specific. A typical scenario follows:

- A list of installed i-αpplis is displayed in response to a user command.

- The user selects the i-αppli they wish to launch from the list.
- The JAM specifies the main class name as found in the ADF's AppClass key and launches the Java Virtual Machine.
- The i-appli screen will appear on the mobile phone's display.

Furthermore, the above-mentioned user operations do not only result in launching the i-appli; the i-appli Runtime Environment also supports the following types of launch variations.

Timer launch

i-appli launch via application linking from the browser, mailer, external device (infrared port, etc.), or another i-appli

Automatic launch following a return to the stand-by condition through registration as a stand-by application

Launch immediately after download

See the section on the LaunchAt key in Section 15.5.1 for more details about timer launch. For details about application linking, see Chapter 11, and see Chapter 9 for information regarding stand-by applications. For details regarding i-αppli that launch immediately after download, see chapter 16.

[DoJa-2.0] / [DoJa-3.0]

Only user-operated launch and timer launch were supported in DoJa-1.0 Profile. From DoJa-2.0 Profile onward, i-appli launching from the browser, mailer, and external devices via application linking and stand-by launching for stand-by applications were added. Further, launching i-αppli from other i-αppli and launching immediately after download were added in DoJa-3.0 Profile.

1.4.3 Updating an i-αppli

i-appli updates will be performed when an instruction is given by the through a menu or some other means from the list of installed i-applis to update the i-appli (the actual procedure will vary depending on the manufacturer).

When an i-appli is installed the JAM saves the URL to its ADF. When the user requests that the i-appli be updated, the JAM accesses this URL again and retrieves a new ADF. Then, the JAM decides whether the i-αppli needs to be updated based on the content of this ADF. It only begins the update process if it decides that an update is actually necessary.

To update an i-αppli (for example, to fix bugs or add new features), the content provider must do the following:

- Set a date for the ADF's LastModified key which is later than that of the previous version. Configure other keys to appropriate values as necessary. However, please do not set the ScratchPad size (the SPsize key) of the application to be updated to a smaller value than the previous version. (When managing ScratchPad segmentation, the size of individual segments should not be smaller than those in the previous version.) As long as the size of the updated i-αppli's ScratchPad is at least as large as that of the previous version, the updated i-αppli will inherit the contents of the previous ScratchPad.
- Post the updated JAR file and ADF file on the content provider's Web site. The name of the new ADF must be the same as that of the old one. Be aware that if you change the URL of the ADF, users who have already downloaded that i-appli to their mobile phones will no longer be able to update it.

The JAM will not update the i-appli if the ADF's LastModified key has not been updated, even if an explicit update request is made.

Under the DoJa-3.0 Profile, a function was added which allows i-applis to use the API to recommend an update to the user in addition to the standard method of updating via user control. This functionality is

achieved through one kind of application linking, in which the native component for updating i-αppli is called from the i-αppli. See Chapter 11 regarding the i-αppli update API.

[DoJa-3.0]

For models of phone without the i-appli update API (DoJa-2.0 Profile or earlier), it will be necessary to notify the user about updates through a Web site or e-mail.

1.4.4 Deleting the i-αppli

Each manufacturer has a procedure for removing i-αpplis at the request of the user. When the user requests that an i-appli be deleted, the JAM will delete the JAR file as well as the ScratchPad assigned to the i-appli on the mobile phone.

1.4.5 Moving the i-appli Between the Mobile Phone and External Memory

Some FOMA 902iS series or newer model phones support the ability to move i-applis (the i-appli and all related data such as its ScratchPad and ADF) between the mobile phone and an external memory device as an optional feature. The user can use this functionality to backup i-applis on their mobile phone to an external memory device or to restore i-applis on an external memory device back to their mobile phone (or to a different mobile phone).

However, in order to protect the content creator's rights, the following restrictions are placed on moving i-applis in this way.

When an i-appli is moved to an external memory device from a mobile phone, the i-appli that was moved is deleted from that mobile phone. Furthermore, the i-appli is saved in an encrypted format on the external memory device, and the contents of that i-appli's files cannot be loaded or viewed on a PC or other device.

When an i-appli is restored from an external memory device to a mobile phone (even a mobile phone other than the one that originally downloaded the i-appli), the UIM card which was used in the mobile phone when that i-appli was downloaded must be present in the destination mobile phone. Additionally, the UIM card which was used to download the i-appli must be present in the destination mobile phone whenever it wants to execute that i-appli.

An API is provided for mobile phones which support this feature which allows an i-appli to determine whether or not it was moved through an external memory device to its present location. For details regarding this API, refer to the IApplication class (in particular, the methods isMoved(), isMovedFromOtherTerminal(), and clearMoved()) in the API Reference.

When an i-appli is moved to a mobile phone from an external memory device, the destination mobile phone will perform the same checks as it would if it were downloading said i-appli (i.e., does the same i-appli already exist, can the necessary resources for the i-appli, ScratchPad, FeliCa chip, etc. be allocated, etc.). If one of these conditions is encountered, the i-appli cannot be moved to the destination mobile phone.

Chapter 2
i-αppli Runtime Environment

This chapter describes the makeup of the functions and APIs of the Java Application Runtime Environment (i.e., the i-αppli Runtime Environment) implemented on i-αppli-capable mobile phones.

2.1 i-appli Runtime Environment

The i-appli Runtime Environment consists of the mobile phone's native applications, the Java Application Manager, the i-appli API, the K Virtual Machine, and the CLDC API. The following diagram shows the makeup of the i-αppli Runtime Environment.

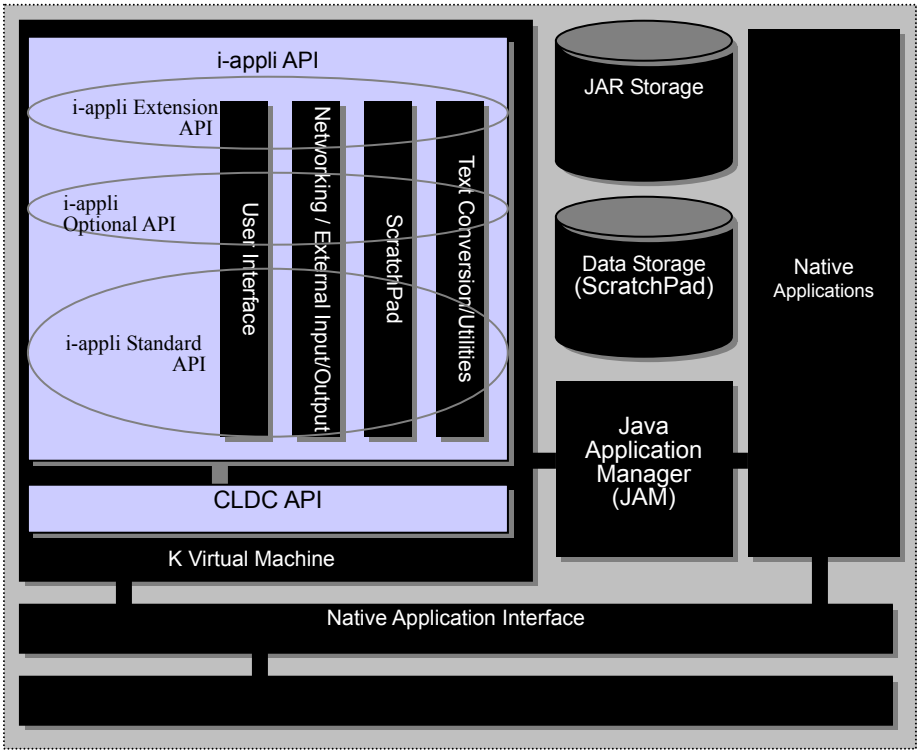


Figure 2: The i-appli Runtime Environment

2.1.1 Native Applications and the Operating System

Ordinary telephones and traditional browser-based i-mode services use the operating system and native applications (e.g. the Browser, networking components, and Dialer) on the mobile phone to operate. The i-αppli Runtime Environment also includes functionality for the linked operation of these native applications using the JAM.

2.1.2 Java Application Manager

The Java Application Manager (JAM) manages individual i-αpplis stored as JAR files, and communicates with the K Virtual Machine.

The main features of the Java Application Manager are shown in Section 1.2.

2.1.3 K Virtual Machine

The K Virtual Machine is a version of a Java Virtual Machine that was re-designed for small embedded devices. It is designed to run on memory-/CPU-/power-constrained devices.

2.1.4 CLDC API

The CLDC APIs include a subset of classes inherited from Java 2 Standard Edition (J2SE). The CLDC APIs share many of the features included in the larger editions of the Java runtime environment, but in order to meet the requirements of small devices, it is optimized for their constraints.

The CLDC APIs are shown in Section 2.2.2.

[DoJa-4.0]

> In the i-αppli Runtime Environment in the DoJa-4.0 Profile and later, the CLDC version is 1.1 (it is 1.0 in the i-αppli Runtime Environment in the DoJa-3.x Profile and before). There are some additional functions, such as supporting floating point numbers and such related API in CLDC-1.1.

2.1.5 i-appli APIs

The i-appli APIs are designed for use on i-appli compatible mobile phones. While the CLDC provides the application infrastructure that can be shared on small embedded devices, the i-αppli APIs build on the CLDC base and offer application infrastructures tailored to i-αppli services. The following individual elements comprise the i-αppli APIs:

- User Interface APIs
- Networking APIs
- SJIS text processing
- ScratchPad local storage
- Application linked operation API and native functionality call API
- Infrared port control (External-connection and infrared remote control via OBEX)

Refer to Section 2.2.3 for more information about the APIs included in the i-appli APIs. Chapters 4 onward discuss the exact procedures for using each function in detail.

The i-appli APIs can be divided into three major categories.

- αppli standard APIs and their operations are designated with common specifications and are installed as standard on all models.

- Although i-αppli optional APIs and their operations are designated with common specifications, it is the responsibility of the manufacturer to decide whether or not they will be installed.
- Even in cases where identical functions are being created, there is a possibility that differences may exist in i-αppli extension APIs and their operations depending on the manufacturer in question.

For an i-αppli that is required to run on all models, i-αppli optional APIs and i-αppli extension APIs should not be used.

2.2 Overview of APIs in the i-αppli Runtime Environment

The remainder of this chapter describes the APIs supported in the i-αppli Runtime Environment. This section is subdivided as follows:

- 2.2.1 Network Architecture
- 2.2.2 CLDC API
- 2.2.3 i-αppli API

The CLDC APIs contain the subset class from the standard J2SE library, and although mapping to J2SE is functionally possible, these APIs also contain CLDC-specific classes which have been redesigned for small devices. This issue was settled through the use of the Java Community Process (JCP).

Conversely, the i-appli APIs are a specialized group of APIs for the i-mode service and are regulated by NTT DOCOMO.

This section will only discuss the i-appli standard API category of i-appli APIs. For details regarding the i-appli optional APIs and the i-appli extension APIs, refer to the "i-αppli Content Developer's Guide: i-αppli Options and Extensions", and the "i-αppli Content Developer's Guide: Optional and Extension API Reference".

2.2.1 Network Architecture

The i-αppli network architecture is nearly identical to the network architecture of conventional i-mode services. Application-level communication between a mobile phone and a content provider's site is conducted using HTTP. Below is a conceptual diagram of the network architecture:

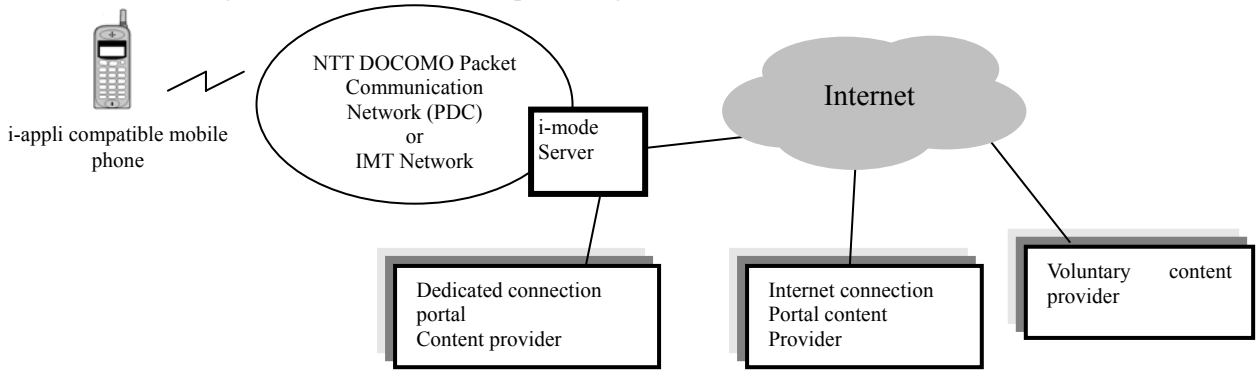


Figure 3: Conceptual Diagram of the Network Architecture

i-αppli service network operations can be divided into the following two categories:

- Downloading and installing i-applis
- Server access from a running i-appli

As with conventional i-mode services, i-αppli service network operations are conducted via an i-mode server. Even on a mobile phone, HTTP is used for the application-level communication protocol, and so the i-mode server does not conduct any processing along the lines of protocol conversion. Even when i-αpplis are downloaded, the file is sent from the content provider-side Web server to the mobile phone as-is.

Additionally, i-appli compatible mobile phones support secure socket layer (SSL) server authentication on the mobile phone side. By utilizing the networking features and SSL support of an i-appli compatible mobile phone, you can perform secure HTTP (HTTPS) communication. In this case, the secure zone is that between mobile phone to the content provider-side Web server.

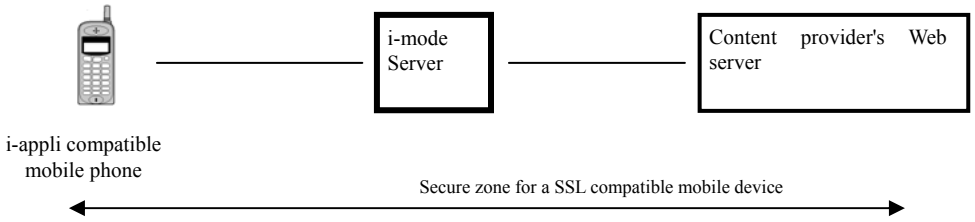


Figure 4: SSL Support of i-mode Services

In FOMA mobile phones, except for the first model, a client certificate, which has been issued for each client, can be stored in a UIM card and FirstPass, which provides SSL client authentication functionality during Internet access, can be used. In FirstPass supported FOMA mobile phones, i-αppli can utilize client authentication that uses FirstPass client certificate. In this case, a DOCOMO root CA certificate must be installed so that the Web server side can support FirstPass. For details on FirstPass, see the following URL:

http://www.nttdocomo.co.jp/service/anshin/firstpass/index.html

Mobile phones which do not support FirstPass (including all PDC mobile phone models) do not support client authentication.

2.2.2 CLDC API

The majority of the class libraries included in CLDC are a subset of the larger editions of the Java execution environment (J2SE) to ensure upward compatibility and portability of Java applications. While upward compatibility is a very desirable goal, J2SE libraries have strong internal dependencies that make it difficult to create subsets in such important areas as security, input/output, User Interface, networking and storage. These dependencies are a natural consequence of the design evolution and reuse that have occurred as the Java libraries have developed over time. Unfortunately, these dependencies make it very difficult to take just one part of the libraries without including several others. For this reason, some of the libraries have been redesigned, especially in the areas of networking and I/O.

J2SE Classes Supported in CLDC

CLDC provides a large number of classes inherited from Java 2 Standard Edition (J2SE). The functions of these classes and the modes of use for inherited methods are not changed. The classes do not add any public or protected methods or fields that are not available in the corresponding J2SE classes. This is because the rules for J2ME configurations mandate that each class that has the same name and package name as a J2SE class, must be a strict subset of the corresponding J2SE class, and do not allow independent methods and fields to be added.

System Classes

Standard Java class libraries include several classes that are intimately linked with the Java virtual machine. Additionally, several standard Java tools assume the presence of particular classes in the system. For example, the standard Java compiler will create code that uses specific methods of String and StringBuffer.

```
java.lang.Object
java.lang.Class
java.lang.Runtime
java.lang.System
java.lang.Thread
java.lang.Runnable (interface)
java.lang.String
java.lang.StringBuffer
java.lang.Throwable
```

Data Type Classes

CLDC supports the following Collection classes from the java.util package. These classes are a strict subset of the equivalent Java 2 Standard Edition classes.

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

[DoJa-4.0]

CLDC-1.1 installed in the i-αppli Runtime Environment in the DoJa-4.0 Profile and later supports the floating point number, and a Double class and a Float class are added accordingly in data type classes.

Reference Object Classes

CLDC supports the following Reference object classes from the java.lang.ref package. CLDC-1.1 supports a subset of WeakReference objects.

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

[DoJa-4.0]

CLDC-1.1 installed in the i-αppli Runtime Environment in the DoJa-4.0 Profile and later supports a subset of the Reference object classes in J2SE, and the above package and classes are added accordingly.

Collection Classes

CLDC supports the following Collection classes from the java.util package.

```
java.util.Vector
java.util.Stack
java.util.Hashtable
java.util.Enumeration (interface)
```

I/O Classes

CLDC supports the following classes from the java.io package. The classes Reader, Writer, InputStreamReader, and InputStreamWriter are required for internationalization.

```
java.io.InputStream
java.io.OutputStream
java.io.PrintStream
java.io.ByteArrayInputStream
java.io.ByteArrayOutputStream
java.io.DataInput (interface)
java.io.DataOutput (interface)
java.io.DataInputStream
java.io.DataOutputStream
java.io.Reader
java.io.Writer
java.io.InputStreamReader
java.io.OutputStreamWriter
```

Micro Edition Classes

CLDC supports the following classes from the javax.microedition.io package.

```
javax.microedition.io.Connector
javax.microedition.io.ConnectionNotFoundException
javax.microedition.io.Connection (interface)
javax.microedition.io.ContentConnection (interface)
javax.microedition.io.Datagram (interface)
javax.microedition.io.DatagramConnection (interface)
javax.microedition.io.InputConnection (interface)
javax.microedition.io.OutputConnection (interface)
javax.microedition.io.StreamConnection (interface)
javax.microedition.io.StreamConnectionNotifier (interface)
```

Calendar and Time Classes

CLDC contains a small subset of the J2SE standard classes java.util.Calendar, java.util.Date, and java.util.TimeZone. CLDC supports GMT as a time zone. Additional time zones may be provided by actual implementations. The i-appli Runtime Environment supports JST as an additional time zone. Some mobile phones which have international roaming support may also support time zones for regions/countries outside of Japan.

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

Additional Utility Classes

CLDC provides two additional utility classes. Class java.util.Random provides a simple pseudo-random number generator that is useful for implementing Java applications such as games.The java.lang.Math class provides various numerical calculation methods.

```
java.lang.Math
java.util.Random
```

Error and Exception Classes

Because the CLDC class library must generally maintain a high level of compatibility with the J2SE class library, the classes included in the CLDC throw the exact same exceptions as their J2SE counterparts. Therefore, a large number of Exception classes are included in the CLDC class library.

Error Classes

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

[DoJa-4.0]

java.lang.NoClassDefFoundError was added in CLDC-1.1.

Exception Classes

```
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
```

JVM Features Not Supported on the CLDC Platform

The following features have been eliminated from the standard Java Virtual Machine to provide a smaller footprint for the CLDC virtual machine (KVM).

JNI
The KVM does not support the Java Native Interface (JNI).

Thread Groups
The KVM implements multithreading, but does not have support for thread groups. Thread operations such as starting and stopping of threads can be applied only to individual thread objects. If application developers want to perform thread operations for groups of threads, they must use explicit collection objects at the application level to manage the thread objects as a group.

Finalization

CLDC does not include the `Object` class's `finalize()` method, and therefore the KVM does not support Object finalization. Furthermore, the CLDC does not allow you to define the `finalize()` method.

[DoJa-4.0]

The i-αppli Runtime Environment in the DoJa-4.0 and later can handle floating point numbers via the floating point number support in CLDC-1.1. On the other hand, the i-αppli Runtime Environment in DoJa-3.x and earlier, which uses CLDC-1.0 as its platform, cannot handle floating point numbers.

Generic Connection Framework (javax.microedition.io package)

The requirement for having a small footprint J2ME implementation has led to the generalization of the J2SE network and I/O classes. The target of this new framework is to function as a strict subset for network functionality and input/output functionality in J2SE. While this subset can be easily mapped for general low-level hardware or arbitrary J2SE implementation, it provides improved extensibility, flexibility, and consistency for the support of new devices and protocols. Access methods based on individual concept are not provided for various kinds of communications, but an access method based on the following integrated concept is used at application programming level.

All connections are created using the `Connector` class's class method `open()`. If the connection is successful, the method `open()` returns an object implemented with one of the Generic Connection interface families (sub-interface families whose root is the `Connection` interface). This interface group is made up of a large number of interfaces, as shown in Figure 5.

The `open()` method takes a URL parameter in the general form:

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

The objective of this design is to isolate, as much as possible, the differences between the setup of one access target and another, into a string characterizing the type of access target (in this case the URL string). The parameter to `Connector.open()` is this URL string. The major advantage of this approach is that the majority of the application program does not need to be changed regardless of the type of the connection to be used. This is different from a conventional implementation in J2SE or other application runtime environment. In a conventional implementation, when access target is changed (for example, file input/ouput is changed to socket input/output), the concept applying to the application program is changed significantly.

If the differences associated with a specific initialization of protocols can be identified by a parameter of `Connector.open()`, and can be absorbed by the platform side, it is not necessary to perform processing, which is specific to the protocols, on the application program side. Therefore, the portability of the application program can be improved by employing a mechanism as described above. Application program that obtains parameter character strings from an environment during execution provides advanced portability in terms of platform and protocol. This can eliminate or minimize the amount of rewriting required in the application program when changing from one form of connection to another.

The binding of protocols to an application program is done at runtime. At the implementation level, the protocol portion of the URL parameter to `Connector.open()` determines which of the supported protocol implementations is to be used. It is this runtime binding mechanism that allows the program to dynamically adapt to different protocols at runtime. This is similar to the relationship between application programs and device drivers on a PC or workstation.

Connections to different types of devices will sometimes need to conduct operations specific to that type of device. A file, for instance, can be renamed, but no similar operations exist for a TCP/IP. The Generic Connection framework reflects these different capabilities, ensuring that logically identical operations share the same API.

The new framework is implemented using a hierarchy of sub-interfaces inherited from the Connection interface that group together classes of protocols with the same semantics. The Generic Connection framework addresses six basic interface types, not counting the Connection interface, which is a super-interface for all interfaces:

- Basic serial input: `InputConnection`
- Basic serial output: `OutputConnection`
- Datagram oriented communications: `DatagramConnection`
- Stream oriented communications (sub-interface of `InputConnection` and `OutputConnection`): `StreamConnection`
- Notification mechanism which notifies the server of a client/server connection: `StreamConnectionNotifier`
- Connection to send/receive content (sub-interface of `StreamConnection`): `ContentConnection`

The collection of `Connection` interfaces forms a hierarchy that becomes progressively more capable as the hierarchy inherits from the root `Connection` interface. This arrangement allows content developers to choose the optimal level of cross-protocol portability for the application program being written.The following shows the relationship among major classes in the Generic Connection class hierarchy.

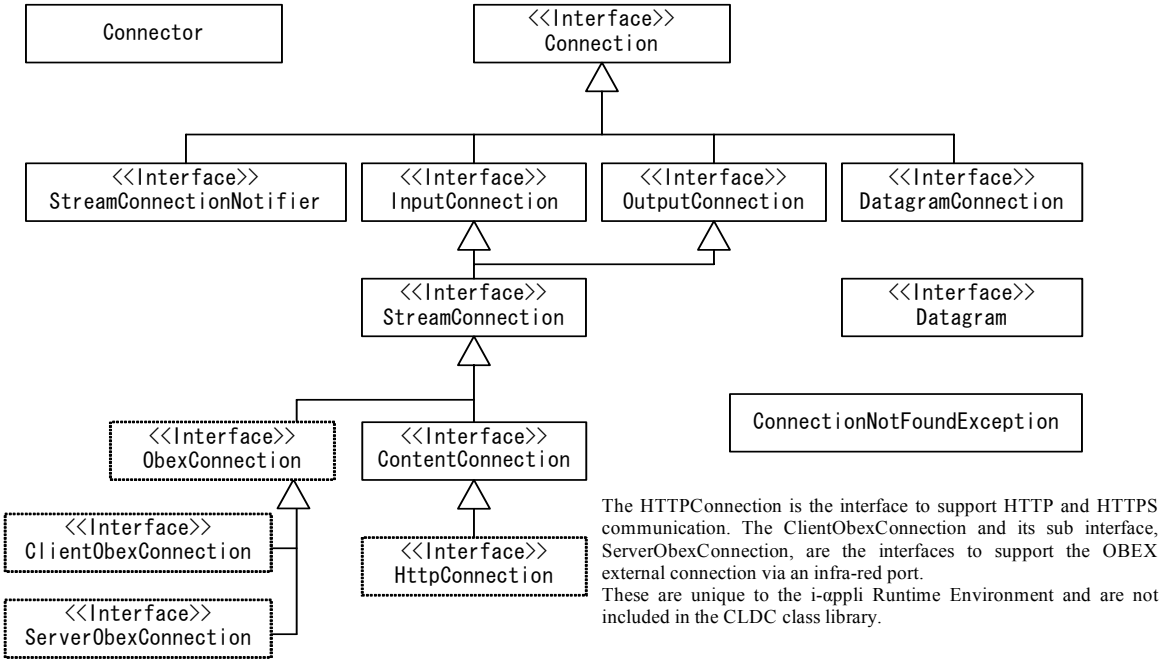


Figure 5: Generic Connection Class Hierarchy

The i-αppli Runtime Environment supports the following interfaces for external access by an i-αppli:

- HTTP(S) Communication :`HttpConnection`
  (sub-interface of `ContentConnection` which is included in the i-appli API)
- ScratchPad :`StreamConnection`
- Resource: :`InputConnection`
- OBEX External Connection: `ObexConnection`, `ClientObexConnection`, `ServerObexConnection`
  (sub-interface of `StreamConnection` which is included in the i-appli API)

The following two items are available as optional parameters of the `Connector.open()` method:

- Access mode (READ, WRITE, READ_WRITE)
- Timeout exception request flag

For the access mode, a value is set appropriate to the operation to be accessed. The timeout exception request flag is set to true for HTTP(S) communications and OBEX external-connections, and false for ScratchPad or resources.

[DoJa-2.0]

Although OBEX external connection via the infrared port is supported for DoJa-2.0 Profile or later, the necessary APIs are also defined in the Generic Connection framework.

2.2.3 i-αppli API

The following sections will cover the classes belonging to the i-appli standard API included in the `io`, `net`, `util`, `lang`, `ui`, `device`, `system`, and `security` sub-packages of the `com.nttdocomo` package. (Some of these classes include methods belonging to the iApplication API.)The sub-packages `graphics3d`, `sound3d` and `util3d` exist in the ui package to provide further 3D graphics and 3D sound functions.

[DoJa-3.0]

The `com.nttdocomo.device` package and `com.nttdocomo.system` package are new to DoJa-3.0 Profile.

[DoJa-4.0]

The `com.nttdocomo.ui.graphics3d` package, `com.nttdocomo.ui.sound3d` package, and `com.nttdocomo.ui.util3d` package are new to DoJa-4.0 Profile.

[DoJa-4.1]

The `com.nttdocomo.security` package is newly added to DoJa-4.1 Profile.

[DoJa-5.0]

The `com.nttdocomo.graphics3d.collision` package is newly added to DoJa-5.0 Profile.

The com.nttdocomo.io Package

CLDC's Generic Connection framework forms the foundation of the i-appli Runtime Environment's input/output package. The `com.nttdocomo.io` package provides the interfaces and Exception classes needed to connect to a network resource specified in URL format and perform input and output. This interface supports network connections through the HTTP/HTTPS protocol and OBEX external connections via infrared port. This package includes supplementary CLDC input/output classes to efficiently handle text streams.

See Section 2.2.2 for more details about the Generic Connection framework.

The package contains the following interface:

Name	Description
`ClientObexConnection`	Interface that defines access to the infrared port in the capacity of an OBEX client. By specifying the URL “obex:/irclient” for the method javax.microedition.io.Connector.open(), an implementation instance for this interface may be obtained. ClientObexConnection is a sub-interface of ObexConnection, the common interface for OBEX servers and OBEX clients within the i-αppli Runtime Environment. [DoJa-2.0] This interface is newly added with DoJa-2.0 Profile.
`HttpConnection`	Interface that defines access to a HTTP connection. An implementation instance of this interface can be obtained by specifying an HTTP or HTTPS URL in the javax.microedition.io.Connector.open() method. HttpConnection is a sub-interface of the Generic Connection framework's javax.microedition.io.ContentConnection interface.
`ObexConnection`	Interface that defines a common interface for OBEX servers and OBEX clients. It is not possible to use directly implemented instances of this interface in application programs. This interface defines response status and operation codes constant, in addition to common accessor methods for OBEX servers and OBEX clients. ObexConnection is a sub-interface of the Generic Connection framework's javax.microedition.io.StreamConnection interface. [DoJa-2.0] This interface is newly added with DoJa-2.0 Profile.
`ServerObexConnection`	Interface that defines access to the infrared port for OBEX servers. By specifying the URL “obex:/irserver” for the method javax.microedition.io.Connector.open(), an implementation instance for this interface may be obtained. ServerObexConnection is a sub-interface of ObexConnection, the common interface for OBEX servers and OBEX clients within the i-αppli Runtime Environment. [DoJa-2.0] This interface is newly added with DoJa-2.0 Profile.

The package contains the following classes.

Name	Description
BufferedReader	Supports a buffered text input stream. This class offers functionality equivalent to a subset of the J2SE java.io.BufferedReader class. [DoJa-3.0] This class is newly added with DoJa-3.0 Profile.
ConnectionException	This class throws an exception when an I/O processing error is detected in the com.nttdocomo.io package. Exception details are made available by the getStatus() method. [DoJa-4.0] There have been name changes for some exception statuses defined in this class to integrate with other exception classes in DoJa-4.0. Old: RESOURCE_BUSY -> New: BUSY_RESOURCE Old: NO_RESOURCE -> New: NO_RESOURCES Former names are defined in DoJa-4.0 to sustain the compatibility, but its use is deprecated.
PrintWriter	Supports a text output steam for outputting the formatted object expression. This class offers functionality equivalent to a subset of the J2SE java.io.PrintWriter class. [DoJa-3.0] This class is newly added with DoJa-3.0 Profile.

Chapter 5 describes client-server programming using HttpConnection in detail. Also, for more details regarding OBEX external-connection programming, refer to Chapter 10.

The com.nttdocomo.net Package

The package contains the following classes.

Name	Description
URLEncoder	This class contains a utility method for encoding a String into URL encoded format (i.e., an encoding format used with the x-www.form-urlencoded MIME type).Upon encoding, the String is converted to SJIS encoding.
URLDecoder	This class contains a utility method for decoding from URL encoded format to a String. Input data is assumed to be based on SJIS encoding.

These classes have the same capabilities as the classes with the same names in package java.net2 of Java 2 Standard Edition.

The com.nttdocomo.util Package

This package contains the following classes and interfaces:

Name	Description
Base64	A utility class to provide functionality for encoding a byte string to a Base 64 format string and functionality for decoding a Base 64 format string to a byte string. [DoJa-3.5] This class is newly added with DoJa-3.5 Profile.
EventListener	All event listener interfaces addressed by the i-αppli API inherit this interface.

JarFormatException	Exception class indicating that a Jar file format error has occurred. This exception occurs when an attempt is made to handle data of an invalid format in the JarInflater class. [DoJa-3.0] This class is newly added with DoJa-3.0 Profile.
JarInflater	Provides functionality for expanding and extracting entries contained in Jar format file images. The Jar file image can use either a byte array or byte stream. [DoJa-3.0] This class is newly added with DoJa-3.0 Profile.
MessageDigest	Provides functionality for requesting a message digest (hash value) from any byte data. The MD5 and SHA-1 algorithms are supported in DoJa-3.0 Profile. [DoJa-3.0] This class is newly added with DoJa-3.0 Profile.
Phone	Representing a dialer, the Phone class provides a means for accessing the mobile phone's native voice-call functionality. When this voice-call functionality is called from an i-αppli, the JAM will ask the user to confirm whether a voice-call transmission can be carried out. It is not possible to omit this confirmation process. This class also has methods for retrieving the dialer's unique identification information (information unique to the hardware, such as the serial number). [DoJa-2.0] This class is newly added with DoJa-2.0 Profile. [DoJa-2.1] Methods for retrieving unique identification information were added in DoJa-2.1 Profile.
ScheduleDate	Class for expressing the schedule time. This class is used to set the schedule time to the native application from i-αppli. With this class, schedules can be entered for native applications, such as the scheduler and alarm settings. [DoJa-3.0] This class is newly added with DoJa-3.0 Profile.
TimeKeeper	The i-appli API contains a timer function which notifies the i-appli in the form of an event when a certain amount of time has passed. Timer classes (com.nttdocomo.util.Timer and com.nttdocomo.ui.ShortTimer) implement this interface. The timer's resolution (smallest timer interval that can be handled) depends on the manufacturer. [DoJa-4.0] The specification of the timer's resolution was specified in the DoJa-4.0 Profile. By using the newly added method getMinTimeInterval(), the minimum interval to generate a timer event in a model can be checked. In addition, by using the method getResolution(), you can also checked how many msec of a timer event can be generated from the value of getMinTimeInterval(). Therefore, by regarding the value of the method getMinTimeInterval() as x and the value of the method getResolution() as y, a timer event can be generated by an interval of x+ny msec (where n is an integer) in the model. When an application program specifies a value which does not fit in this expression as an interval of the timer, the minimum value to fit in this expression is selected by rounding up.
Timer	This is the timer class. This class supports one-shot timer and interval timer. The TimerListener object can be registered for receiving timer events. The timerExpired() method of the listener object is called when the timer event occurs. The period for which the timer is valid spans from the launching of the i-αppli to its termination, and timer operation will continue even if the screen is replaced. However, the timer will stop if i-αppli execution is interrupted or, in the case of a stand-by application, goes into the sleep mode.

TimerListener	This interface is designed to implement a listener which retrieves the timer event broadcast by com.nttdocomo.util.Timer. Note that event processing for com.nttdocomo.ui.ShortTimer is handled by the Canvas class' processEvent() method, and therefore does not use this interface.

The com.nttdocomo.lang Package

This package contains the following exception class:

Name	Description
IllegalStateException	If calling of operations or methods is carried out in conditions for which such an action is not permitted, IllegalStateException will be thrown as an execution exception. [DoJa-2.0] This class is newly added with DoJa-2.0 Profile.
IterationAborted Exception	This exception class is used to represent a situation in which an iterative process being performed on an array or some other data structure is interrupted due to some exception being thrown. You can use this class' methods to find out the direct cause of failure for the iterative process and the details of the exception thrown. [DoJa-5.0] This class is newly added with DoJa-5.0 Profile.
MemoryManager	This class provides utility functions for monitoring the usage status of application runtime memory (heap memory). [DoJa-5.0] This class is newly added with DoJa-5.0 Profile.
UnsupportedOperation Exception	UnsupportedOperationException is an execution exception that is thrown when an unsupported operation or method is executed.

The com.nttdocomo.ui Package

This package contains the following interfaces.

Name	Description
Audio3DListener	An interface to define an event listener to notify an event related to the 3D sound control to an application program. A motion complete event is defined in SoundMotion (moving location) in this profile. [DoJa-4.0] This interface is newly added with DoJa-4.0 Profile.
Audio3DLocalization	This is an interface to represent location. There are some methods (classes) to represent location in the 3D sound control function, and any classes implement this interface. [DoJa-4.0] This interface is newly added with DoJa-4.0 Profile.
ComponentListener	The listener interface for receiving component events. Classes that process component events must implement this interface. Listener objects are registered to the Panel by using the Panel.setComponentListener() method. When a component event takes place, the componentAction() method of the listener object is invoked, passing as arguments the type of event, the component in which the event took place, and event parameters.

FocusManager	The focus manager class is an object that manages the transfer of focus between components placed on the Panel. This is the interface implemented by the focus manager class. When no specific indications are carried out for the focus manager, the default focus manager class will be used. In all previous profiles, no focus manager other than the default focus manager was defined.
Interactable	Interface implemented for components (e.g. Button, ListBox, and TextBox) that can be manipulated by the user. This interface defines a method for setting component focus and for setting whether components can be operated.
KeyListener	A listener interface for receiving individual key operations in the high-level API. Implementation objects of this panel must be registered on the Panel in order to be used. This interface defines two methods, corresponding to key press events and key release events: keyPressed() and keyReleased().
LayoutManager	The layout manager is an object that manages the layout of components on the Panel. This is the interface implemented by the layout manager class. The i-appli Runtime Environment defines the default layout manager when no particular specifications are made regarding the layout manager and the HTML layout manager which handles HTML-style layouts. [DoJa-2.0] The HTML layout manager is newly added with DoJa-2.0 Profile.
MediaData	An implementation object of MediaData is obtained by using the MediaManager.getData() method. The actual object to be returned is implementation specific. The MediaData interface was created for manufacturers to implement new proprietary media data.
MediaImage	MediaImage represents static or dynamic images. An implementation object of MediaImage is obtained by using the MediaManager.getImage() method. MediaImage conducts playback using the VisualPresenter object. See Section 4.4 for the data that can be used.
MediaListener	This interface is implemented by the media listener in order to receive events from the media presenter that presents multimedia data. The mediaAction() method defined by this interface can receive such media presenter events as start playback, terminate, and stop.
MediaPresenter	Defines the interface that an object presenting media data must implement. The presenter class presents media data according to its actual content. Supported media data types are terminal dependent. If media data not supported by the terminal is set, a UIException is thrown.
MediaResource	Defines the interface which must be inherited by the media data interfaces (MediaData, MediaImage, MediaSound).The use() method must be used before a MediaResource instance obtained by using MediaManager can actually be used, and the unuse() method must be called after use. The dispose() method should be called when the MediaResource instance is no longer needed, in order to discard that instance. [DoJa-5.0] In DoJa-5.0 Profile, methods to control the memory management method for MediaResource and its sub-interfaces (media interfaces) were added.
MediaSound	MediaSound represents a sound. An implementation object of MediaSound is obtained by using the MediaManager.getSound() method. MediaSound is played using an AudioPresenter object. See Section 4.4 for the formats which can be used as sound data.
SoftKeyListener	This interface is used to listen for the two soft keys on the mobile phone being invoked in the high-level API. Implementation objects of this panel must be registered on the Panel in order to be used. This interface defines two methods corresponding to soft key press events and soft key release events: softKeyPressed() and softKeyReleased().

This package contains the following classes.

Name	Description
AnchorButton	The AnchorButton component represents HTML anchor-type buttons (i.e., underlined text which can operate as a button). You can also specify an image to be used as a headline before the text. You can also specify text which spans across multiple lines. In situations where processing is to be carried out in response to pressing an anchor button, Listener registration of the object implementing ComponentListener is to be carried out using Panel's setComponentListener() method. The Listener will then accept BUTTON_PRESSED events when the anchor button is manipulated. [DoJa-2.0] This class is newly added with DoJa-2.0 Profile.
Audio3D	This is a controller to control the 3D sound for each AudioPresenter. It enables/disables the 3D sound control or controls the location by using the Audio3D object. The number of virtual sound sources you can use in your i-appli (the number of 3D sound controlled resources) depends on the manufacturer. See Section 13.2 for details. [DoJa-4.0] This class is newly added with DoJa-4.0 Profile.
AudioPresenter	This class is used to play back sound media data (MediaSound).The AudioPresenter can be used when using both high level API and low level API.
Button	The Button component represents a labeled button. In situations where processing is to be carried out in response to pressing a button, Listener registration of the object implementing ComponentListener is to be carried out using Panel's setComponentListener() method. The Listener will then accept BUTTON_PRESSED events when the button is manipulated.
Canvas	The Canvas component represents a rectangular area of blank space on the screen. i-applis can draw in this area or use them to trap user input events in this area. The developer will typically define a subclass inheriting from Canvas. The i-αppli overrides the paint() method of the Canvas class to draw on the Canvas.
Component	A Component is a user interface object having a graphical representation that can be displayed on the screen and is used to interact with the user. Component has sub-classes such as Button, ListBox, and Textbox. Component belongs to the high-level API, and is used in combination with the Panel.
Dialog	A Dialog is top-level window with a title, a text message, and at least one button. A Dialog can be displayed on top of either the Panel or a Canvas.
Display	Display is a class that manages screen real estate. All methods of Display are static, and no instances are ever created. The Display class changes screen content (Panel or Canvas) and delivers input events to the displayable object currently shown on the screen. This class receives low-level events and passes the appropriate displayable object to that event. This object determines the way the event is processed. Display also contains device information, such as the width/height of the display area, as well as a variety of constants for use with keys and event processing.
EncodedImage	This class stores the result of encoding the content drawn in a Canvas or Image object via the ImageEncoder class into a binary image format (JPEG, etc.). Using this class an i-appli can extract a byte image from drawn data or re-convert that data to a media image and extract that data. [DoJa-3.0] This class is newly added with DoJa-3.0 Profile.
Font	Shows the font used when displaying text in the Graphics context, a Component, or a Dialog. Which fonts are supported depends on the manufacturer. [DoJa-3.0] Features for setting a font to Component and Dialog objects were added in DoJa-3.0 Profile.

Frame	Frame is an abstract class that represents the screen. The content that can be shown on the display of a mobile phone at any one time corresponds to a Frame instance. Frame has sub-classes such as Canvas, Dialog, and Panel. To switch screens, call the Display.setCurrent() method with the Frame object for the desired screen to change to set as the parameter. (Note, however, that in the case of Dialog only, display is achieved by calling the Dialog.show() method.)
Graphics	This class carries out rendering for Canvas and Image, and it also encapsulates the corresponding operations. Note that Image rendering can only be carried out for Image instances where the corresponding image has been newly created using the method Image.createImage(). [DoJa-2.0] Functionality for the acquisition of Graphics instances from Image instances and for rendering are newly added with DoJa-2.0 Profile.
HTMLLayout	This layout manager is used to create HTML-style layouts. Although similar to the default layout manager in terms of layout creation in accordance with the sequence of addition to Panel, this manager can use its methods to make indications for HTML carriage return, paragraph separation, and alignment. [DoJa-2.0] This class is newly added with DoJa-2.0 Profile.
IApplication	The IApplication class is the i-αppli superclass. The i-appli's main class (the class which is first called when an i-appli is launched) must be a sub-class of IApplication. This class manages the life-cycle from start to finish of an i-αppli.
Image	The abstract class Image represents static images. A MediaImage instance is created from a static (GIF or JPEG) image, and an implementation instance of Image can be obtained by calling the getImage() method. The Image.createImage() method can be used to create new Image instances. See Section 4.4 for the data formats which can be used as static images. [DoJa-2.0] The createImage() method was added in DoJa-2.0 Profile.
ImageButton	The ImageButton component is used to create buttons which may be pasted onto static images. In situations where processing is to be carried out in response to pressing an image button, Listener registration of the object implementing ComponentListener is to be carried out using Panel's setComponentListener() method. The Listener will then accept BUTTON_PRESSED events when the image button is manipulated. [DoJa-2.0] This class is newly added with DoJa-2.0 Profile.
ImageEncoder	Provides functionality for encoding the rendered content of the Canvas or Image as a JPEG etc. format byte image. Results of encoding are returned as an EncodedImage instance. The common encoding format that can be used on all phone models is JPEG. [DoJa-3.0] This class is newly added with DoJa-3.0 Profile.
ImageLabel	An ImageLabel component is used to display a static image on the Panel. The Image instance can be set to ImageLabel, and placed on the Panel.
ImageMap	ImageMap provides a function to create an apparent large image by lining up several types of small images. It is suitable to configure a large background image with small-sized image data. [DoJa-3.5] This class is newly added with DoJa-3.5 Profile (moved from the i-appli Option API).
Label	A Label component is used to display text on the Panel. A Label represents a single line of read-only text. The text can be changed by the application program, but a user cannot edit it directly.

ListBox	The ListBox component presents the user with a scrolling list of text items. The developer can choose the appearance and capabilities of the ListBox by making the appropriate specifications when creating the instance.
MApplication	MApplication constitutes a template class for stand-by applications. This class inherits the class IApplication, and the main class for the stand-by application (of the class initially called upon startup of the stand-by application) must be a sub-class of this class. The MApplication class manages the life-cycle of stand-by applications through launch, termination, and execution conditions. [DoJa-2.0] This class is newly added with DoJa-2.0 Profile.
MediaManager	A class for managing MediaResources. The MediaManager is the factory class for media data (MediaData, MediaImage, and MediaSound). Media data can be created by specifying a URL location string, stream or byte array. [DoJa-3.0] Creation of media data by specifying a stream or byte array was added in DoJa-3.0 Profile.
Palette	This is a class to represent a color palette and to be used with PalettedImage. [DoJa-3.5] This class is newly added with DoJa-3.5 Profile (moved from the i-appli Option API).
PalettedImage	This is an image which is created by GIF data and can specify a color palette. [DoJa-3.5] This class is newly added with DoJa-3.5 Profile (moved from the i-appli Option API).
Panel	Panel is a container class for displaying components on the screen in the high-level API.A Panel displays space that any component can attach to. Only one Panel can be displayed on the screen at a time.
PhoneSystem	This class is used to access the native resources of the platform. The platform's native resources include icon information (mail icon, etc.) shown on the display. [DoJa-2.0] Icon information referencing was added in the DoJa-2.0 profile.
ShortTimer	This is a relatively low-overhead timer meant for use in the low-level API. This class supports one-shot timer and interval timer. ShortTimer is obtained by using the getShortTimer() static method of this class. Canvas is passed as the argument to the getShortTimer() method, and the timer event passes this Canvas to the processEvent() method. ShortTimer only operates when the corresponding Canvas is being shown on the display. If a frame change occurs while ShortTimer is operating, the timer automatically stops.
Sprite	This class represents each sprite moving on the screen by the sprite function. Sprite cannot be displayed by itself. Therefore, it is used in combination with SpriteSet. [DoJa-3.5] This class is newly added with DoJa-3.5 Profile (moved from the i-appli Option API).
SpriteSet	This class plays a role to combine multiple sprites by the sprite function. An application program assigns sprites in the required number to a sprite set and draws it. [DoJa-3.5] This class is newly added with DoJa-3.5 Profile (moved from the i-appli Option API).
TextBox	The TextBox component is used to display and input text. The TextBox is the only component which allows the user to enter any character string.
Ticker	The Ticker component is used to display streaming text that automatically scrolls to the left. The component displays text in a ticker tape style (equivalent to the MARQUEE tag in i-mode HTML).

UIException	UIException is a runtime exception class that is thrown when an error is detected while a UI package, a native function using the UI package, or similar software is running.
VisualPresenter	The VisualPresenter component is a presenter class used for playing the visible media (MediaImage) data in the high-level API.

The com.nttdocomo.ui.graphics3d Package

This package includes the following classes and interfaces to provide 3D graphics rendering functions.

Name	Description
ActionTable	This is a class to maintain an action (animation data) of a 3D model. The animation data is created using commercially-available 3D authoring tools and the like.
DrawableObject3D	This is the base class for a rendering-enabled 3D object. Each object of the Figure class, the Group class, the Primitive class deriving this class can be targeted for rendering via the Graphics3D.renderObject3D() method. In addition, this has some methods such as a method to easily recognize collisions of two objects in the 3D space, a blend function, and a distortion correction function for textures.
Figure	This is a class to maintain 3D model data. This object can render and be specified to the method Graphics3D.renderObject3D() as a rendering target.
Fog	This is a class to maintain data to add a fog effect. This supports a linear fog mode and an index fog mode.
Graphics3D	This is an interface to define methods which must provide graphics contexts to support 3D Graphics Rendering functions.
Group	This is a class to represent a group that is a 3D object aggregate. This object can render and be specified to the method Graphics3D.renderObject3D() as a rendering target. It can assign multiple Figure objects or Primitive objects to it and render at once. It can also assign Fog objects or Light objects to it to affect rendering in a group.
Light	This is a class to maintain light source data. It supports an environment light source, a parallel light source, a point light source, and a spot light source.
Object3D	This is the base class for all 3D objects. A 3D object indicates all objects which affect the rendering results of 3D graphics. 3D objects are roughly divided into two: objects such as Figure or Primitive, which can be directly targeted for rendering and objects such as Light or Fog, which affect the rendering results.
Primitive	This is a class to maintain an array of primitives (geometric shapes, such as points, lines, or planes) that stores vertex data for use in rendering primitives. This object can render and be specified to the method Graphics3D.renderObject3D() as a rendering target.
Texture	This class stores texture data. This can handle both a texture used for model mapping and a texture used for environment mapping.

The com.nttdocomo.ui.graphics3d.collision Package

This package includes the following classes and interfaces to handle collision detection in 3D space for use in unison with the API's 3D graphics functions.

Name	Description
AABBox	This class represents a Box which is parallel to world coordinate axes.
AABCapsule	This class represents a Capsule whose central axis is parallel to the world coordinate Y-axis.

AABCylinder	This class represents a Cylinder whose central axis is parallel to the world coordinate Y-axis.
AbstractBV	An abstract class that represents a three dimensional shape. This class implements the BoundingVolume interface. Three dimensional shapes are provided as concrete classes by inheriting this class.
AbstractShape	An abstract class which becomes the basis for all shapes; implements the Shape interface. Non three dimensional shapes are provided as concrete classes by inheriting this class. Three dimensional shapes are provided as concrete classes by inheriting the AbstractBV class, which in turn inherits this class.
AxisAlignedBV	Among three dimensional shapes used for comparison decisions, this interface is implemented by all three dimensional shapes that are placed parallel to world coordinate axes.
BoundingVolume	Among the shapes used for comparison decisions, this interface implements three dimensional shapes. This interface inherits the Shape interface. This package supports rectangular solids, cylinders, spheres, and capsule shapes.
Box	This class represents a rectangular solid and is one of the solid shapes used for comparison decisions.
BVBuilder	This class is used to create a BVFigure object or a BoundingVolume object (any of the three dimensional objects which implement the BoundingVolume interface) which can be attached to a BVFigure object from a Figure.
BVFigure	A Figure which has a bone structure and is made up of multiple BoundingVolume objects. Normal Figures (Figure class) make a shape by attaching polygons to them, but a BVFigure makes a shape by attaching BoundingVolume objects to it.
Capsule	This class represents a capsule shape and is one of the solid shapes used for comparison decisions.
Collision	This class provides functions related to collision detection. Collision detection, intersection detection, distance calculation, and other such functions are provided by this class.
CollisionObserver	This interface should be implemented by any observer class designed to receive notification about collisions and information about those collisions when a collision is detected in the Collision class. Applications which want to receive such notifications must create an observer class that implements this interface, create an object from that class, and assign it to a created Collision object.
Cylinder	This class represents a cylinder and is one of the solid shapes used for comparison decisions.
IntersectionAttribute	This class represents intersection information that is to be passed to the CollisionObserver's notification method when an intersection is detected between a Figure and a Ray.
Line	This class represents a line and is one of the non-solid shapes used for comparison decisions.
Plane	This class represents an infinite plane and is one of the non-solid shapes used for comparison decisions.
Point	This class represents a point and is one of the non-solid shapes used for comparison decisions.
Ray	This class represents a ray and is one of the non-solid shapes used for comparison decisions.
Shape	This interface is implemented by all shapes used in collision detection. The collision detection methods supplied by this package do not directly determine when Figures collide with each other. Rather, it determines when simple shapes placed at nearly the same location of these Figures collide with each other instead. The Shape interface is implemented by all shapes used for these comparison decisions.
Sphere	This class represents a sphere and is one of the solid shapes used for comparison decisions.

`Triangle`	This class represents a triangle and is one of the non-solid shapes used for comparison decisions.
`ViewVolume`	This class provides functionality to determine which `BoundingVolume` objects can be seen in the current view frustum (the field of view from the camera's perspective). By using this class to determine which objects are not visible within the camera's current field of view, you can find out only which objects actually require rendering on the screen at that time.

The com.nttdocomo.ui.sound3d Package

This package includes the following classes and interfaces to provide 3D sound functions. These classes and interfaces are used by combining with a class com.nttdocomo.ui.Audio3D which is a controller to control 3D sound.

Name	Description
`CartesianListener`	A class to represent a listener located on the Cartesian coordinate system when representing the location by the `CartesianPosition` class. Setting the direction of the listener required to represent location on the Cartesian coordinate system will be set by using this class.
`CartesianPosition`	When controlling 3D sound, this class represents the relative position between a listener and the virtual sound source (i.e. the sound's orientation) using the Cartesian coordinate system consisting of the x, y, and z axes.
`PolarPosition`	Represents a relative position between a listener and a virtual sound source by using the polar coordinates system. The system assumes that a listener faces a certain direction at the origin and represents a position of the virtual sound source by specifying the azimuth, elevation, and the distance from there.
`SoundMotion`	The `SoundMotion` class is used to represent "moving location". It enables a user to feel that a virtual sound source moves along a route by setting an elapsed time since beginning of moving and the location at the time as route information.
`SoundPosition`	An interface to represent fixed location represented by the position of a listener and the position of a virtual sound source. A class implementing this interface represents a relative position of both by specifying a position of the listener and the position of a virtual sound source.

The com.nttdocomo.ui.util3d Package

This package provides the utility functions to perform math or vector operations required to use 3D graphics functions or 3D sound functions.

Name	Description
FastMath	A high speed math utility with the premise of using with 3D-related functions. Mobile phones supporting this profile are installed with CLDC-1.1, and floating point number operations supported by KVM can be used in this version, but the floating point number operations are internally replaced with an integer operation in this class. Therefore, this might cause a larger error than a normal floating point number operation, but it performs faster calculations.
Transform	Handles the matrices for conversion of 3D affines. This class represents a conversion of affines by a matrix with four rows and four columns.
Vector3D	This is a class to represent four-dimensional vectors. The inner product, outer product, and normalization utility methods are provided. This is used to represent a position or a direction on a coordination system.

The com.nttdocomo.device Package

This package contains the following classes for controlling the mobile phone’s peripheral devices.

Name	Description
Camera	Calls the mobile phone's camera feature, and provides a means for accessing functionality for shooting still and moving images. The actual shooting is performed by the user operating the shutter with respect to the camera feature activated by the i-αppli. Support for taking movies depends on the manufacturer.
CodeReader	Provides a code recognition function to read a barcode (JAN code or 2D barcode) by using a camera device installed in a mobile phone. [DoJa-3.5] This code recognition function is newly added with DoJa-3.5 Profile (moved from the i-appli Option API).
DeviceException	DeviceException is the runtime exception class thrown when an error is detected, mainly during the operation of the device package and native functionality that use the device package.
IrRemoteControl	Provides infrared remote control functionality. This class acts as a transmitter in order to send the remote control signal indicated in the IrRemoteControlFrame class from the infrared port.
IrRemoteControlFrame	Indicates the data frame indicating the external device control details sent from the infrared remote control feature. In general the manufacturer of that external device decides independently what control is performed when a given kind of data-frame is received. It is necessary for the programmer to design the application so that the data frames transmitted comply with the specifications of the targeted external device.

The com.nttdocomo.system Package

This package contains the following classes and interfaces for utilizing the phonebook feature, bookmarks, scheduler and other native functionality of the mobile phone from i-αppli. This package contains many classes and methods the use of which are permitted only in the trusted i-αppli. These features will not be explained here.

Name	Description
ApplicationStore	Provides functions that access the mobile phone's native i-appli management functions. By using this class, the entry ID, which is required when starting another i-αppli that has been downloaded to the mobile phone (launcher mode), can be retrieved.
Bookmark	Provides functions that access the mobile phone's native bookmark management functions. Using this class, bookmark entries can be newly added to the mobile phone.
ImageStore	Provides functions that access the mobile phone's native image data management functions. Using this class, images can be read from and newly added to the mobile phone's image data storing area.
InterruptedOperation Exception	Exception indicating that the user's operation of a native application function called by the i-appli was interrupted by an outside event, such as an incoming phone call.
MovieStore	Provides functions that access the mobile phone's native movie data (iMotion) management functions. Using this class, new movie data can be registered in the movie data storage area in a mobile phone. Note that acquisition of movie data from the movie data storage area corresponds to i-αppli optional API and whether it is supported or not depends on the manufacturer. [DoJa-4.0] / [DoJa-4.1] This class was newly added to DoJa-4.0 Profile, however, in DoJa-4.0 Profile, this function can be used only in Trusted i-αppli. In DoJa-4.1 Profile and later, general i-αpplis can use this class.
PhoneBook	Provides functions that access the mobile phone's native phonebook management functions. By using this class, phonebook entries can be newly added to the mobile phone.

PhoneBookConstants	Interface which encompasses all constants related to phonebook management functions. This interface is implemented in the PhoneBook and PhoneBookParam classes. The developer need not create this interface's implementation class.
PhoneBookGroup	Provides functions that access the phonebook group management functions of the mobile phone's native phonebook group management functionality. By using this class, new phonebook groups can be newly added to the mobile phone. New phonebook entries can be added to the phonebook groups created with this class by using the PhoneBook class.
PhoneBookParam	Indicates phonebook entries for use as new additions. Phonebook entry data, such as name and phone-number, are assigned to this object and it is added to the mobile phone with the PhoneBook class.
Schedule	Provides functions that access the mobile phone's native scheduler. By using this class, new schedules can be added to the mobile phone.
StoreException	Exception indicating an access error, such as insufficient empty area, occurred when attempting to access a data area managed by a native mobile phone application.

Notes:

- Within the com.nttdocomo.system package classes, there are many functions which call a native mobile phone function and newly add user data. The mobile phone's user can disable these functions through the mobile phone settings. When these functions are called, the i-appli Runtime Environment will request the user's confirmation to perform such data registration actions. If the user's permission is not received through this mechanism, that data cannot be added.

The com.nttdocomo.security Package

This package contains classes that provide functions related to securities such as certificates and digital signature.

Name	Description
CertificateException	Exception that indicates when an error related to certificates occurred. An error occurs when the terminal does not have root Certification Authority certificates that are required for certificate verification, or when they become invalid due to the expiration even if the terminal has them.
PKCS7SignedData	Represents data with digital signature. Data that is given digital signature by PKCS7Signer is represented as an object of this class.
PKCS7Signer	Class that provides a function to generate data with digital signature. By providing arbitrary data to this class, digital signature can be implemented for that data.
PKCS7SignerInfo	Class that represents information about the signer included in a digital signature. [DoJa-5.1] This class is newly added with DoJa-5.1 Profile.
SignatureException	Exception that indicates when an error related to digital signature occurred. An error occurs when digital signature processing or verification of digital signature fails.
X509Certificate	Represents a X509 certificate. Inside of an i-αppli, the certificate is represented as an object of this class.

IApplication and MApplication

The com.nttdocomo.ui.IApplication class is the superclass of all i-αpplis. Normally, an i-appli's main class (the first class called when an i-appli is launched, as specified by the AppClass key in the ADF)

must inherit this class. This is similar to the case of Java applets running on a PC browser, which must inherit from the `java.applet.Applet` class.

The IApplication class defines the key methods relating to an i-αppli's lifecycle, from launch to termination:

- The `IApplication.start()` method begins the execution of an i-αppli. The developer overrides this method, and implements its processing classes. At this time, it is possible to call the `IApplication.getArgs()` method to obtain the i-αppli's startup parameter (value specified by the ADF's AppParam key).
- When an i-appli is launched for the first time after being downloaded, or when an i-appli launch request is made via the application linking function, parameters can be passed to the i-appli via HTML or the mail body.
- Call the `IApplication.terminate()` method to terminate an i-αppli's execution.
- When an i-appli is running and a high priority external event such as an incoming phone call occurs, the i-appli's execution will be suspended and a response to the external event will be processed. When handling for the external event completes, the i-αppli automatically restarts. When the i-αppli restarts, the `IApplication.resume()` method is called. The developer can implement any handling needed when resuming i-αppli execution by overriding this method. Further, by using the `IApplication.getSuspendInfo()` method (added in the DoJa-3.0 profile), the reason for previous i-αppli interruptions, and information on the events that fired during the interruption can be retrieved (depending on the manufacturer, it may be possible to retrieve information on events that fire during just that one interruption occurrence). The details of the thread operation flow and thread switch control when an i-appli is running is left to the Java Virtual Machine's thread control. Therefore, the `resume()` method may not necessarily always be called first when resuming execution of an i-appli.
- By using the `IApplication.launch()` method, another native application or i-αppli can be launched from i-αppli. In general, if this operation is successful, the original i-αppli will end execution.

Furthermore, the `com.nttdocomo.ui.MApplication` is inherited and extended from `IApplication` — the class which provides support for the complicated life-cycle unique to stand-by applications. The main class for stand-by applications must inherit from the `MApplication` class.

In addition to the functionality of `IApplication`, the class `MApplication` also includes the following functions.

Stand-by Application Mode Control:

Stand-by applications can adopt a total of three modes — namely, the same Active mode as adopted by other i-αpplis during execution in addition to the modes listed below.

The i-appli is running, but is hooked so that key events and/or soft key events are used by the mobile phone's native stand-by functionality and therefore the i-appli is not notified of these events [Inactive Mode]

The screen display is saved in its current state and the i-appli stops running to save battery power [Dormant Mode]

The MApplication class manages the transition between these states.

System Event Handling:

In the case of stand-by applications, events from the outside are notified to resident i-αpplis as system events. The developer can override system event handlers with the MApplication class so that, regardless of which of the above-mentioned three conditions is currently adopted, it will be possible to receive and process said events.

For more details regarding stand-by applications, refer to Chapter 9.

Notes:

- The application does not have a direct interface between `IApplication` and its subclasses. Instances of these are created by the i-αppli Runtime Environment.
- Whenever you explicitly define the constructor for the i-appli main class, be sure to always make the access specifier `public`. With the i-αppli Runtime Environment, furthermore, it is not possible to use constructors with arguments when creating a main-class instance. If you explicitly define a main-class constructor, be sure to confirm that it does not have any arguments.
- CLDC defines the following methods: `java.lang.Runtime.exit()` and `java.lang.Sysytem.exit()`. However, these methods cannot be used to terminate execution of an i-appli under the i-appli Runtime Environment. (If these functions are called, a `SecurityException` will be thrown.) Call the `IApplication.terminate()` method to terminate an i-αppli's execution.

Text Conversion

i-αpplis process text using the Japanese language character set and the ASCII character set. The `java.io` package provides classes that allow you to convert between Unicode `character` streams and byte streams of non-Unicode text. Use the `InputStreamReader` class to convert `byte` streams to `character` streams, and use the `OutputStreamWriter` class to convert character streams into byte streams. See Chapter 6 for more details on text conversion.

ScratchPad

The mobile phone memory model for i-applis is called the ScratchPad. The `open()` method of the `Connector` class is used to connect to the Scratchpad. Passing the URL for the ScratchPad to this method opens a connection, and returns an implementation instance of the `javax.microedition.io.StreamConnection` interface defined by CLDC. The `StreamConnection` interface is used to read and write to the ScratchPad area. See Chapter 7 for more details on using the ScratchPad.

Chapter 3
Design Considerations

This chapter provides guidance on designing i-αpplis that efficiently use available system resources, are well adapted to the wireless communications environment, and are easy to use.

3.1 Characteristics of i-αppli Compatible Mobile Phones

In the interests of minimizing the physical size and maximizing the battery life of the mobile phone, the mobile phone's display screen, application executable memory, and application storage memory are very small by comparison with personal computers. Also, the processor runs at a slower speed. The Liquid Crystal Display (LCD) screen consumes very little electricity, but has a slower refresh rate than other types of displays. Text entry must be done using the mobile phone keypad, and since more than one character is assigned to each key, entering text will be a tedious procedure for the user. Text input should therefore be kept to a minimum.

When designing services which make use of i-αppli, it will be important to take these platform-specific characteristics into consideration during i-αppli design. In the following sections we will look at how to solve these problems from a design perspective, and provide some guidelines for how to design easy to use i-applis.

3.2 Memory Issues

Below shows an example how the JAM allocates the storage area (storage area for JAR file and ScratchPad) for three i-αpplis.

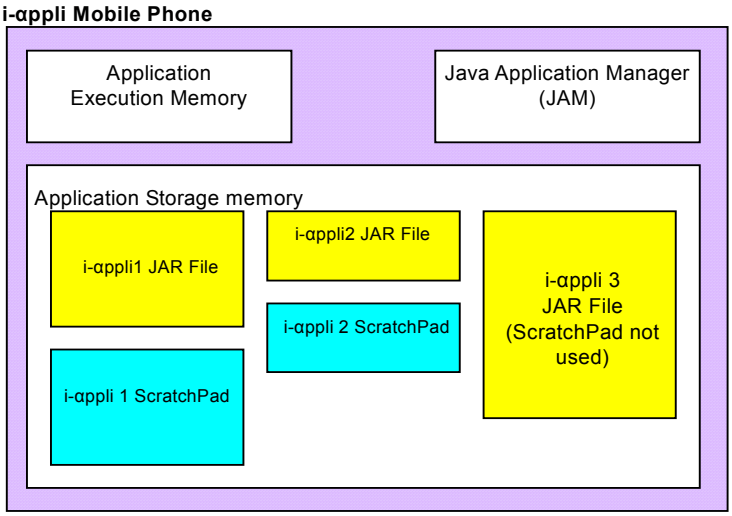


Figure 6: i-αppli Compatible Mobile Phone Memory

For the developer, the following two memory constraints exist.

- Limited application runtime memory. i-αpplis are executed in this area. Objects that are generated during execution are also stored in this area.
- Limited application storage memory. In this area, JAR files for i-αpplis and ScratchPads that corresponds to these i-αpplis are stored.

The absolute limit on application runtime size is the amount of application runtime memory in the mobile phone in which the i-áppli is to execute. The amount of application runtime memory present and how the application runtime memory is used when an i-appli is run depends on the manufacturer.

In addition, the size of application storage is restricted as follows by the generation of a profile (hereinafter 1 K bytes = 1024 bytes). Please be aware that an i-áppli which exceeds this size limit cannot be properly downloaded into a mobile phone.

Profile	PDC Mobile Phones		FOMA Mobile Phones	
	JAR File	ScratchPad	JAR File	ScratchPad
DoJa-1.x	10 Kbytes	5 Kbytes	30 KB	10 Kbytes
DoJa-2.x	30 KB	100 Kbytes	30 KB	200 KB
DoJa-3.x	30 KB	200 KB	100 Kbytes	400 Kbytes
DoJa-4.x	————	————	100 Kbytes	400 Kbytes
DoJa-5.x	————	————	JAR file and ScratchPad together total 1024 KB (*1)	

(*1) Depending on the manufacturer, the JAR file size may be limited to 100 KB and the ScratchPad size limited to 400 KB for download and immediate launch i-appli, which is the same as DoJa-4.x Profile.

Notes:

- A larger ScratchPad size than the specified size might be supported by certain manufacturers. Depending on the region management method employed by the manufacturer, the ScratchPad size for an i-appli that is about to be installed may be limited depending on the total size of all the ScratchPads being used by i-applis that have already been installed. In either case, if a ScratchPad cannot be allocated with the amount of space declared in the i-appli's ADF's SPsize key, that i-appli cannot be downloaded to the mobile phone.

[DoJa-3.0]

In DoJa-3.0 or later profile, a maximum of 16 ScratchPads assigned to a single i-αppli can be internally segment-managed. See Section 7.1 for how to access a ScratchPad that is partitioned and managed. See Section 15.5.1 for how to configure the ADF to partition and manage the ScratchPad.

[DoJa-5.0]

For DoJa-5.0 Profile and later, JAR files over 100KB can be downloaded based on the previously described definition. However, in order for the mobile phone to be able to download a JAR file that exceeds 100 KB in size, the Web server used to distribute the i-appli must be able to support partial GET range requests in bytes. Note that JAR files that exceed 100 KB in size cannot be downloaded from Web servers which give the partial GET response "206 Partial content" in response to a range request.

Be particularly aware of this fact if CGI is being used in any way to download the JAR file.

The range request discussed previously involves sending a request with some conditions using an If-Unmodified-Since header. Please note that as a result of this procedure, if the i-appli download source is performing load balancing across multiple servers and each individual server's system time and/or the timestamps on the JAR files placed on those servers do not match up, the i-appli may not download properly.

3.2.1 Minimizing Memory Usage

This section will discuss two broad categories: application runtime and application storage memory.

Application Runtime Memory (Heap Memory)

Typically, the largest component in application runtime memory is the executable program itself (the classes comprising the i-αppli Runtime Environment and the i-αppli). If the application, with all of the objects it needs, cannot fit in the available runtime memory, then you have no choice but to reduce the size of the executable program, the size of the set of generated objects, or both.
For information on reducing the size of the executable program, see Section 3.2.2.

To reduce the size of the set of generated objects, consider the following:

- Calling the garbage collector often. Particularly with the i-αppli Runtime Environment and its small memory capacity, the fragmentation which occurs in the application runtime memory often leads to cases where the remaining memory cannot be accessed as a single item. (Exceptions such as `OutOfMemoryError`, `UIException`, and `ConnectionException` will occur, and this will manifest itself in unstable i-αppli operation.) In such a case, the garbage collector should be called as soon as possible to resolve any problems.
- Keeping as few objects in memory as possible. Also, minimizing the number of objects and size you generate.
- Consider breaking up your application program into smaller parts. (For example, if you are designing a game with multiple levels, make multiple i-applis, one for each level.)
- Using static when applicable.
- Minimizing stack depth by avoiding excessive nested method calls or recursion.
- Minimizing the number of parameters to methods.
- Minimize the number of classes and the size of individual classes.

- Avoiding the proliferation of threads.
- Creating classes with pre-set size. Dynamically growing classes such as `HashTable`, `ByteArrayInputStream`, `ByteArrayOutputStream`, or `Vector` may lead to memory fragmentation.
- Dispose of any media resources and image that are no longer required (`dispose()`).

If the objects you are generating are as small as you can make them, and you are using application execution memory efficiently but your program sill runs out of memory, consider the following:

- Moving data stored in variables to the ScratchPad.
- Off-load processes which consume a lot of memory from the client to the server.
- If downloading data from a server, avoid doing this in one batch, but rather carry out partial downloads of the smallest possible size.

[DoJa-5.0]

In DoJa-5.0 Profile, the `MemoryManager` class, which is a utility to monitor the usage status of application runtime memory, was added to the `com.nttdocomo.lang package`.

CLDC's java.lang.Runtime class also contains similar functionality in the `freeMemory()` and `totalMemory()` functions, but the `MemoryManager` class also adds the ability to find out the maximum concurrent free space in application runtime memory (that is, the biggest area of space which is not divided up due to fragmentation). Also, if the i-appli Runtime Environment has multiple partitions of application runtime memory (for example, a heap for media data in addition to a heap for Java objects), you can monitor the same information for each of these partitions separately.

Application Storage Memory

Your application consumes two blocks of application storage memory to save the i-αppli:

- The block allocated to the JAR file.
- The block allocated to the ScratchPad.

JAR Files

Make sure the JAR file contains only essential files, and that those files are as compact as possible. If your i-αppli contains different modules for each mobile phone, for instance if using i-αppli optional APIs and i-αppli extension APIs, consider creating a separate i-αppli (a separate JAR file) for each device (see Chapter 16 for details on how to control device-specific downloads). That way, the JAR file downloaded to the mobile phone will contain no extraneous entries.

Also, although you can include resources like images and sound in the JAR file (see Chapter 7), this has a large impact on its size.Consider using the mobile phone’s communication capabilities to obtain infrequently used resources at i-αppli runtime.

Also not that the i-mode service employs a packet billing system for charging communication fees. Reducing the size of the JAR file will make it cheaper for the user to download the i-αppli.

ScratchPad

Do not specify a larger ScratchPad than necessary. And regardless of how much space you allocate to the ScratchPad, use that space as efficiently as possible. You specify your ScratchPad size in the i-αppli’s ADF (see Section 15.5.1 for details). Also note that the available size of the i-αppli’s ScratchPad is subject to the limitations discussed in Section 3.2.

The kind of physical device (hardware) used for the ScratchPad depends on the manufacturer. Note that depending on the model of phone, allocating a larger than necessary ScratchPad could cause a decrease in access speed and impact i-αppli performance.

3.2.2 Class File Size

Compiled Java class files sometimes contain debug information, which increases the size of the file. Reduce class file size by keeping the number of classes defined down to a minimum, and by compiling the code with the `-g:none` option to remove debug information.

Note that if you are building i-αpplis using the i-αppli Runtime Environment build function as provided by NTT DOCOMO, the `-g:none` option will be set during compilation. See Chapter 15 for more information about i-appli development environments.

3.3 Usability and GUI Design

This section presents suggestions for creating i-αpplis that are easy and intuitive to use. In general, it is recommended that on a platform like the mobile phone, complex i-αpplis requiring a lot of keystrokes be avoided in favor of i-αpplis with fewer keystrokes. A few points to remember are:

- Maintain a consistent look and feel throughout the i-αppli.
- The physical screen area is small, so keep the number of UI components shown on a single screen to a minimum. In particular, if a number of UI components supporting user operation (i.e., buttons, etc.) are displayed on a single screen, the requirement for focus-shift operations can result in users considering to be difficult to use.
- UI layout management will be manufacturer dependent, and in most cases the default layout management will be a simple flow layout. Keep the UI layout simple and the i-αppli execution screen appearance uniform to avoid drastic differences between various manufacturers.
- Keep the scrolling to a minimum. Note that while vertical scrolling is supported by Panel, it will be necessary to control screen scrolling from the application program when using Canvas.
- Softkeys are a useful way to simplify user operations. Consider using softkeys instead of Buttons on screens that generate complex user operations. For instance, you could use a softkey to select a component on a screen upon which a large number of input component objects are placed, regardless of which component has focus.
- Text entry is complex and difficult on a mobile phone, so minimize the number of characters the user must enter. Where possible, use an input method that presents choices and lets the user selected from among them.
- Design your screen composition so that commonly performed tasks do not require the user to navigate through many menu levels, or to enter many keystrokes.

3.3.1 Designing for the Target Audience

Everyone uses phones, but not everyone uses computers. Therefore, all the users do not understand the ideas of moving focus between UI components and scrolling intuitively. To be used by wide range of users, the user interface must be able to be used as intuitively as possible. When designing the i-áppli user interface, the target user group must be defined to be expected to use it.

3.3.2 Design Policy for i-αppli Operation Methods

Divide user tasks into categories, based on how commonly they will be performed and how valuable they are to the user. In general, you should make it very easy for the user to perform common tasks. For example, bank balance checking is performed more frequently than bank transfer. Therefore, ideally, the bank balance checking should be performed with fewer key operations than the transfer.

On the other hand, a task that requires several keystrokes – such as entering a Personal Identification Number (PIN) – will certainly be accepted by the user if it performs an important function (in this case, protecting the user's bank account or other assets from unauthorized access). And even a certain amount of user input could be acceptable even if it is a little tedious when first using an i-αppli after it is installed on the mobile phone if it will simplify later activities.

Activities should be consistent. Selecting an item on one screen should be the same as selecting an item on another. Softkeys should be used with a consistent concept throughout the i-αppli in order to avoid user errors and frustration. The most important and often-used tasks should require the fewest keystrokes.

3.3.3 Scrolling

Logically it is possible to design virtual screens bigger than physical screens. However, on such screens, a lot of scrolling is required to view the entire screen or to move the focus between UI components. Since scrolling does not always have short response time as a screen operation, users may feel screens with a lot of scrolling are not easy to use. You should aim to minimize scrolling. Note that the `Panel` only supports vertical scrolling.

[DoJa-2.0]

With DoJa-1.0 Profile, it was left up to individual manufacturers to decide whether or not to support Panel scrolling. With DoJa-2.0 Profile or later, however, a Panel for which the default layout manager or the HTML layout manager has been set will support scrolling in the vertical direction.

3.3.4 Keys and Softkeys

i-αppli can make use of the following standard keys.

- Direction keys and the selection (confirmation) key
- Number keys, 0 through 9, in addition to the # and * keys
- Stand-by application switching key

Certain manufacturers may support other special keys to provide functionality for the i-αppli option category.

Furthermore, i-αppli capable mobile phones also feature two more softkeys in addition to those above. The appearance and location of the softkeys depend on the manufacturer. As stated earlier, using softkeys makes a user interface much more pleasant to use. While you should consider using softkeys in your i-αppli design, you should avoid relying on a specific appearance or location for them.

[DoJa-2.0]

The stand-by application switching key has been newly implemented in DoJa-2.0 Profile in order to support stand-by application functionality. With the exception of cases where the stand-application is in either the sleep or inactive condition, it will be possible for the i-αppli to make direct use of this key (and to obtain key events). If in the sleep condition or inactive condition, events from the stand-by application switching key will be used by the system.

For more details regarding stand-by applications, refer to Chapter 9.

[DoJa-5.0]

For DoJa-5.0 Profile and later, depending on the manufacturer, by using the `DrawArea` key of ADF, a display area may be used wider than the default display area (i.e., when not using the `DrawArea` key). For such models, by using the method `Canvas.setSoftLabelVisible()`, the softkey label display may be switched on and off when displaying the `Canvas`. Note that although the method `setSoftLaelVisible()` is defined in the class Frame, this method cannot be called for frame objects (`Panel` and `Dialog`) other than `Canvas` in the current profile.

3.3.5 Data Entry

The mobile phone's keypad is the user's only means of entering data. Numeric data is therefore easier to enter compared to alphanumeric input. This is because in the case of text data, it takes multiple keystrokes to enter a single character. Text entry should therefore be kept to a minimum.

3.3.6 Password Entry

Perform password entry in either of the following ways:

- Allow hidden passwords. In the `TextBox` constructor, pass `TextBox.DISPLAY_PASSWORD` to the display mode parameter. All input characters will be shown as the '*' character.
- Allow visible passwords. In the `TextBox` constructor, pass `TextBox.DISPLAY_ANY` to the display mode parameter. All input characters are displayed on the screen.

If using the first of these methods, it is standard only for numbers to be used. In contrast to the case with PCs, if characters other than these are included in passwords, it will result in frustrating operations for the user. If using only numbers, the `TextBox.setInputMode()` method can be used to setup number-input mode as the initial input mode, thus preventing incorrect operation by the user.

When the use of non-characters (i.e., text characters) is to be permitted for passwords, the second method should be used.

3.3.7 Use of Threads

Threads are very useful for performing long, blocking operations such as network I/O or UI interaction. However, in any multi-threaded system, there is overhead associated with the context switching between active threads. Since the phone devices are limited in processing power, keep the number of threads to a minimum.

Furthermore, even if multi-thread programming has not been carried out for the application program side, the system may generate threads for event processing or the like. Regardless of the model on which they will operate, thread-safe program design must be implemented when creating i-αpplis.

Notes:

In the i-αppli runtime environment, dialogs or pop-ups will be displayed under the following circumstances:

- The application itself uses the `Dialog` class to display a dialog
  - In HTTP(S) communications, the Web server requests authentication (BASIC authentication)
  - In HTTP(S) communications, when displaying a warning to the user because there is a problem with the server's certificate
  - When using the infra-red port for OBEX external-connection functionality
  - When confirming the launch of another application via application linking functionality with the user.

  These dialogs and pop-ups can only be displayed exclusively, and cannot be displayed simultaneously from multiple threads. If an attempt is made to display these from multiple threads simultaneously, the resulting behavior varies depending on the phone's manufacturer. It is necessary for developers to design application programs so that the above conditions do not occur simultaneously in multiple threads.

3.4 Security

When designing i-αpplis with sensitive information passed between client and server, consider using the HTTPS protocol. Note that when this mechanism is used on the client side, the burden on the CPU increases. This is because at the beginning of every SSL session, on the client side (in this case, the i-αppli) in addition to having to create master keys, data encryption and decryption based on these keys must be performed whenever data is processed and sent to or received from the server.

See Section 5.3 for information about communications using the HTTPS protocol.

Note that when an i-αppli communicates with a server, the server must have the same protocol, server address, and port number as the location from which the i-αppli was downloaded. Consequently, i-αpplis which are to use the HTTPS protocol must be downloaded using this protocol.

3.5 Operating Over a Wireless Network

Be aware that the i-αpplis you develop will be used in an environment in which network connection interruptions are commonplace (users frequently leave and re-enter areas with mobile phone signals available). The i-mode service employs a packet billing system, so i-αpplis should transmit only the necessary amount of data when communicating over the network.

In designing client/server i-αppli, it is necessary to strike a balance between sending/receiving a number of small data packet transactions (minimizing data loss due to interruptions) or sending/receiving one large data packet (minimizing communications overhead).

For detailed descriptions of problems regarding establishment, test, and end of connection, and those regarding management of transactions and sessions, see Chapter 5. See also the “Process Suspension and Resumption” section in this chapter.

Notes:

- With i-ápplis that require communication, an i-ápplı developer may implement a function to retry communication processing to recover from a communication error or an error in the data obtained by communication.

  In the perspective of comfortably using the i-αppli on a mobile phone for which the communication quality (signal quality) frequently changes, it is useful for an i-αppli to automatically retry by detecting an error related to communication. However, in order to prevent the i-αppli from retrying communication indefinitely, make sure that an upper limit is set in the number to be repeated when implementing the process.

  The i-mode service adopts the packet accounting system, and the telecommunication fee is charged according to the actual amount of communication. Even if the size of each communication data is small, retrying communication indefinitely might add significant charges over a short period of time.

  The number of retries in communication processing or an application logic dealing with communication processing should be set as several times at most. When the purposed process cannot be completed within that number of tries, it is strongly recommended that errors be reported to a user by displaying a dialog.

  (For stand-by applications, it is not recommended to exit the i-αppli automatically when an error is detected. This is because there is a possibility that stand-by applications are automatically re-launched by systems and same process may be repeated as a result.)

3.6 Process Suspension and Resumption

The JAM automatically suspends a running i-αppli when the phone receives an incoming phone call during i-αppli runtime. The JAM then sends a Resume message to the i-αppli and restarts program execution when the user terminates the phone call. In this way, in cases where it is necessary for a different function with a higher priority to operate on a mobile phone, the suspension and resumption of i-αpplis may be carried out.

In addition to the receipt of an incoming phone call, certain other conditions exist for which i-αpplis will be suspended. The following presents cases where i-αpplis are suspended and resumed.

- When there is an incoming phone call during i-αppli execution (When i-αppli communication is being executed, the process may not be suspended depending on the i-mode incoming setup on the mobile phone). And when there is an incoming mail message while a stand-by application is being executed in inactive mode or sleep mode. Note that when the stand-by application is operating in active mode, response will be the same as for execution of normal i-αpplis — in other words, the i-αppli will continue to execute regardless of whether or not mail or messages are received.
- Another application launches during i-αppli runtime. This includes both situations where a native application such as a time alarm is launched automatically during i-αppli runtime and situations where the application link function is used to launch a native application from another i-αppli. Note that in cases where the launched application is of the type which carries out packet communication (i.e., a browser or mailer i-αppli), the calling i-αppli will be shut-down rather than suspended.
- When a native function which entails user operation is called from i-αppli. This occurs mostly in application linking functionality, and includes many features which require user confirmation.
- The user performs a forced shut-down of an i-αppli, and upon display of the confirmation dialog, the forced shut-down is canceled.
- The mobile phone includes other mechanisms for the suspension and resumption of i-αpplis (i.e., suspension / resume keys, etc.), and these are operated.

[DoJa-2.0]

Suspension and resume conditions for i-αpplis were clarified in DoJa-2.0 Profile. Please be aware that operations other than those described above may take place with DoJa-1.0.

[DoJa-3.0]

Interruption/resumption of i-αppli has been added to basic functionality in DoJa-3.0 Profile. Interruption/resumption of i-αppli is not supported in some cellular-phones which support profiles older than DoJa-2.0 Profile. In these cellular-phones, when a condition arises requiring i-αppli to be interrupted, it will be terminated.

[DoJa-3.5]

Since FOMA mobile phones supporting the DoJa-3.5 Profile and later differ from PDC mobile phones, when an i-αppli is normally running and a mail message is received in standby active mode, it is obtained in the background without suspending the i-αppli. In addition, even when dealing with FOMA mobile phones supporting the DoJa-3.5 Profile and later, if the i-αppli is in standby inactive mode or in dormant mode, the mail message is obtained as usual by suspending the i-αppli.

When i-αppli execution resumes, the i-áppli main class (class inheriting `com.nttdocomo.ui.IApplication`) `resume()` method is called. The developer can implement i-αppli resume processing by overriding the `resume()` method.

In some i-αpplis — for example, a standalone calculator application — you, as the developer, need not take any special steps to handle suspend/resume. This is because nothing inside will have changed since the i-αppli was suspended and the progress of the application is entirely driven by user-generated events — namely, keystrokes. In particular types of i-αpplis, however, you must take special steps upon resuming execution. In general, there are a number of specific situations that require you to take some action upon resuming i-αppli execution:

- If yours is a client-server i-αppli, the communication result should be checked after the i-αppli is resumed, and communication should be retried whenever so required. This action is necessary since a communication failure may occur upon the suspension or resume of the i-αppli's communication processing.
- When an i-αppli is suspended and resumed, the timers (i.e., Timer or ShortTimer) which had been operating before suspension will be stopped. It will be necessary for this type of timer to be explicitly restarted upon the resume of an i-αppli.

  Note that with DoJa-1.0 Profile, if the execution of an i-αppli is suspended while a timer is being used, it is necessary after the i-αppli resumes to discard the timer instance and to create it anew.
- When the i-áppli is suspended and `AudioPresenter` was playing sound, `AudioPresenter` stops operating. It will not start playing automatically even after the i-αppli is resumed. In this case, an explicit play command (i.e., `play()`) can be used from the i-αppli to resume playback of the sound data from the start. Note that if `VisualPresenter` is carrying out playback of a movie upon the suspension of an i-αppli, this playback will be temporarily halted; however, this will automatically resume from the current frame when the i-αppli is resumed.
- With DoJa-2.0 Profile, when a `Canvas` is redisplayed as a result of the suspension or resume of an i-αppli, it is guaranteed that the `paint()` method for the corresponding `Canvas` will be called once from the i-αppli Runtime Environment. However, in cases where calling of this `paint()` method does not result in complete restoration of the entire screen (i.e., where the application program carries out only partial rendering in accordance with events), the degree to which the system can restore the screen will depend on the model in question.

  In DoJa-3.0 or later profile, when i-αppli resumes, if the system can completely restore the rendered `Canvas` contents, the `paint()` method will not be called.
- In certain cases where an application is suspended and resumed, it will be desirable for a suitable prompt message to be displayed upon resume and for this resume to be temporarily halted. For example, in the case of action games and other applications which require immediate user interaction, the application's ease-of-use upon resuming may be improved by including a short period of time for the user to make the necessary preparations for resume.

[DoJa-2.0]

The effect of suspension and resume on i-αpplis has been clearly defined for DoJa-2.0 Profile. Please be aware that operations other than those described above may take place with DoJa-1.0.

[DoJa-3.0]

In DoJa-3.0 Profile, the IApplication.getSuspendInfo() method has been added in order to retrieve information on the reason for interruption and events which fired during interruption when i-αppli interruption/resumption occurs.

Notes:

- Whether or not an i-αppli performing data communication will be suspended upon the receipt of an incoming phone call will depend on certain i-mode incoming settings. If these settings dictate that incoming phone calls are not to be received during i-mode communication, then i-αppli runtime will not be suspended even if an incoming phone call is received while it is communicating data.
- In situations where unprocessed events are queued in the event queue upon suspension of an i-αppli, all of these events will be discarded when the i-αppli is resumed.
- The details of the thread operation flow and thread switch control when an i-appli is running is left to the Java Virtual Machine's thread control. Therefore, the `resume()` method may not necessarily always be called first when resuming execution of an i-appli.

3.7 Accessing Hardware

The i-αppli Runtime Environment provides a `PhoneSystem` class as a means for accessing the mobile phone's hardware (i.e., its platform resources). Using this class, it is possible to control items such as the display's back-light or the vibrator. Note that frequent access of platform resource an i-αppli can consume a large amount of a mobile phone's battery power. Furthermore, even if the i-αppli should change these settings, they will automatically be returned to their initial settings when the i-αppli terminates.

[DoJa-3.0]

In DoJa-2.0 Profile or earlier, platform resource access functionality was contained almost totally in the i-αppli Optional API, and was only partially implemented depending on the phone's manufacturer. In DoJa-3.0 Profile, most of this was added to the i-αppli standard API.

3.8 Error Processing

If the CLDC API or the i-mode Java extension API detects an error during processing, they throw errors and exceptions just like ordinary Java APIs. You will need to catch exceptions that the application program generates, and take appropriate action as necessary (for instance, submit an exception message to the user).

If runtime exceptions (`java.lang.RuntimeException` and its subclasses) and errors (`java.lang.Error` and its subclasses) are not caught by the application program, the i-αppli will hang.

3.9 Stand-by Applications

DoJa-2.0 Profile or later provides support for stand-by applications as a means of implementing convenient resident-type i-αpplis such as clocks and the like.

Stand-by applications are resident-type i-αpplis which display a stand-by screen, possibly containing a stand-by image, while they are executing. Naturally, if the application program changes the image content, this will be immediately implemented in the stand-by screen.

In DoJa-1.0 Profile too, it was possible to show the user that an i-αppli was a resident on the stand-by screen by constantly preserving the i-αppli's executing state. However, this implementation method was characterized by the following problems.

- If the user wanted to make a phone call or to browse the web, the user was required to explicitly shut down the i-αppli. Furthermore, the user also had to explicitly restart the i-αppli when these operations had been completed.
- It was not possible to receive mail while this type of i-αppli was executing. For this reason, it was common for users to be unaware that mail had been received.
- Battery power continued to be consumed as the i-αppli executed, and it was not possible to continue execution for extended periods of time.

Eliminating these types of problems, stand-by applications provide a means for supplying users with convenient resident-type applications. If you are developing an i-αppli for which constant, continuous operation is a pre-condition, consider the possibility of implementing this i-αppli as a stand-by application.

Note that more details regarding stand-by applications are available in Chapter 9.

3.10 Identification of Mobile Phone Models

The following two methods have been provided to enable an i-αppli to identify the type of model on which it is executing.

- `microedition.platform`, a system property for i-αppli Runtime Environment

  An application program can use the method `java.lang.System.getProperty()` to reference system properties (i.e., the property set for acquisition of system information).Although the `microedition.platform` property is defined as a system property for indicating the host property in CLDC, the mobile phone's model name is stored as a property value with i-αppli Runtime Environment. Accordingly, i-αpplis can check this number to identify the model on which it is currently executing.

  The model name stored in this property will be identical to the name included in the user agent.

- `User-Agent` request data as transmitted to the server during HTTP communication

  Similar to situations where browsers are being used for web access, a user agent (i.e., the `User-Agent` header) will be included in the request header when HTTP communication is being carried out from an i-αppli. As a result, i-αpplis and those on the communicating server can check the content of the `User-Agent` header to identify the model of the currently-interacting mobile phone.

[DoJa-2.0]

With DoJa-1.0 Profile, the type of value to be set for `microedition.platform` is left to the discretion of individual manufacturers, and therefore, it will not be possible in all cases to obtain model names.

3.11 Simultaneous Execution of Multiple Applications on FOMA Mobile Phones

Among FOMA mobile phones, there are some models which can simultaneously execute multiple terminal applications (Ex. a browser and a conversation function) to perform site browsing using i-mode and conversation at the same time. Here, the effect of this on the life cycle of an i-áppli on those kinds of mobile phones is described.

(1) Switch to another application by a user operation while executing an i-αppli

When a user indicates to switch to another application by a user operation (such as a task key operation) while executing an i-αppli, as a rule, the execution of the i-αppli is suspended. When a switch to the i-αppli by a user operation is performed again, the execution of the i-αppli is resumed. The types of applications which can be simultaneously executed with an i-áppli depend on the manufacturer. If a user indicates the execution of an application which cannot be simultaneously executed with an i-αppli on the model at the same time as the i-αppli, the i-αppli is terminated, and the application is executed.

An application program can obtain the reason of the suspension of an i-αppli execution and an event which occurred while suspending via the `IApplication.getSuspendInfo()` method, but an event which generates while suspending, `IApplication.SUSPEND_MULTITASK_APPLICATION`, was added in DoJa-3.5 Profile and later to recognize suspending by switching applications as described in this section.

(2) Mail receiving operations while executing an i-αppli

On PDC mobile phones, if a mail is received while an i-αppli is running (while normally executing an i-αppli and being active in an i-αppli standby activated), the specification is programmed to notify by an icon that an undelivered mail is kept at the i-mode center instead of retrieving the mail. On the other hand, FOMA mobile phones retrieve a mail in the background without suspending an i-αppli when they receive a mail in that kind of state. If an i-αppli performs a function which cannot be simultaneously executed with the packet communication (such as an infrared remote control) in a state in which a mail is being retrieved in background, an exception occurs.

Moreover, even for FOMA mobile phones, if it is in an inactive or dormant mode for i-αppli standby execution, as with PDC mobile phones, the i-αppli is suspended and mail reception is performed.

Chapter 4
Designing the User Interface

User Interface (UI) functionality has the following basic requirements:

- Creation of high-level User Interface componenilities.ts such as buttons and labels.
- Event processing.
- Layout processing.
- Image file display and sound file playback capab

The i-αppli Runtime Environment User Interface API is made up of both low-level and high-level APIs. The following sections describe the basic areas of functionality and show code examples.

The high-level APIs provide various components. Developers combines those components to create an i-αppli user interface. Applications using the high-level API have less concern for hardware characteristics, which allows them to easily run on different manufacturers' mobile phones. The developer has a limited degree of freedom, but on the other hand, can write high-function i-αpplis with fewer lines of code. Items that are provided as user interface parts for the high-level API are referred to as "components." The developer assembles these components together to create an i-αppli. Screens using the high-level API generally have about the same level of expressiveness as i-mode HTML.

The low-level API allows developers to create i-αpplis whilst taking into consideration such hardware characteristics as the size of the mobile phone's screen or the keypad properties. The low-level API can be used to create i-αpplis that make the most of the features of each mobile phone. The developer has a high degree of freedom, but must manage more detail, such as graphics rendering, screen size, and logical coordinates.

A single screen is displayed as a `Panel` or `Canvas` instance.Panels are used by high-level API screens (screens make up of components), and Canvases are used by low-level API screens (screens that are drawn "by hand"). The display can only display one screen, either Panel or `Canvas`. Components cannot be used on the screen that is configured by `Canvas`. In screens created using a `Panel`, however, components for handling images can be used to incorporate low-level rendering.

One i-áppli can have multiple `Panel` and `Canvas`. An application program can use the method `Display.setCurrent()` to switch the display between `Panel` and `Canvas`.

Events such as a button press or a timer going off are handled differently by the high-level and low-level API. In the case of low-level API, all events can be received by the i-αppli, including key press/release events. With high-level API, however, depending on the component, some low-level events are handled internally, or converted into higher-level events and sent to the i-αppli.

In the case of low-level API, the low-level events which are generated as a result of actions such as key pressing or releasing, etc. are reported to the `Canvas` object. With low-level API, developers create and use sub-classes derived from the `Canvas` class, and by over-writing the `processEvent()` method from the `Canvas` class at this time, it is possible to incorporate low-level events. This method will be called when events occur, and therefore, processing for the event in question can be carried out internally.The type for the occurring event and parameters specific to that event are passed to the `processEvent()` method.

In the case of high-level API, high-level events resulting from button pressing and the like are reported to the `Panel` object. When processing for this type of high-level event is to be carried out within the i-αppli, it will be necessary to register a Listener object in the `Panel` object.When events occur, the `Panel` object will call the Listener object registered within it, and it will report the events' occurrence. Different Listener interfaces for high-level events have been made available for different types of event-occurrence source. Furthermore, the information reported to the Listener event upon the occurrence of an event will vary with respect to the type of Listener.

Refer to Section 4.3 for more details on event listeners.

The figure below illustrates the classes making up the User Interface. In addition, classes and interfaces on the 3D graphics rendering and the 3D sound control that are separately described in Chapter 13 are omitted in this class block diagram.

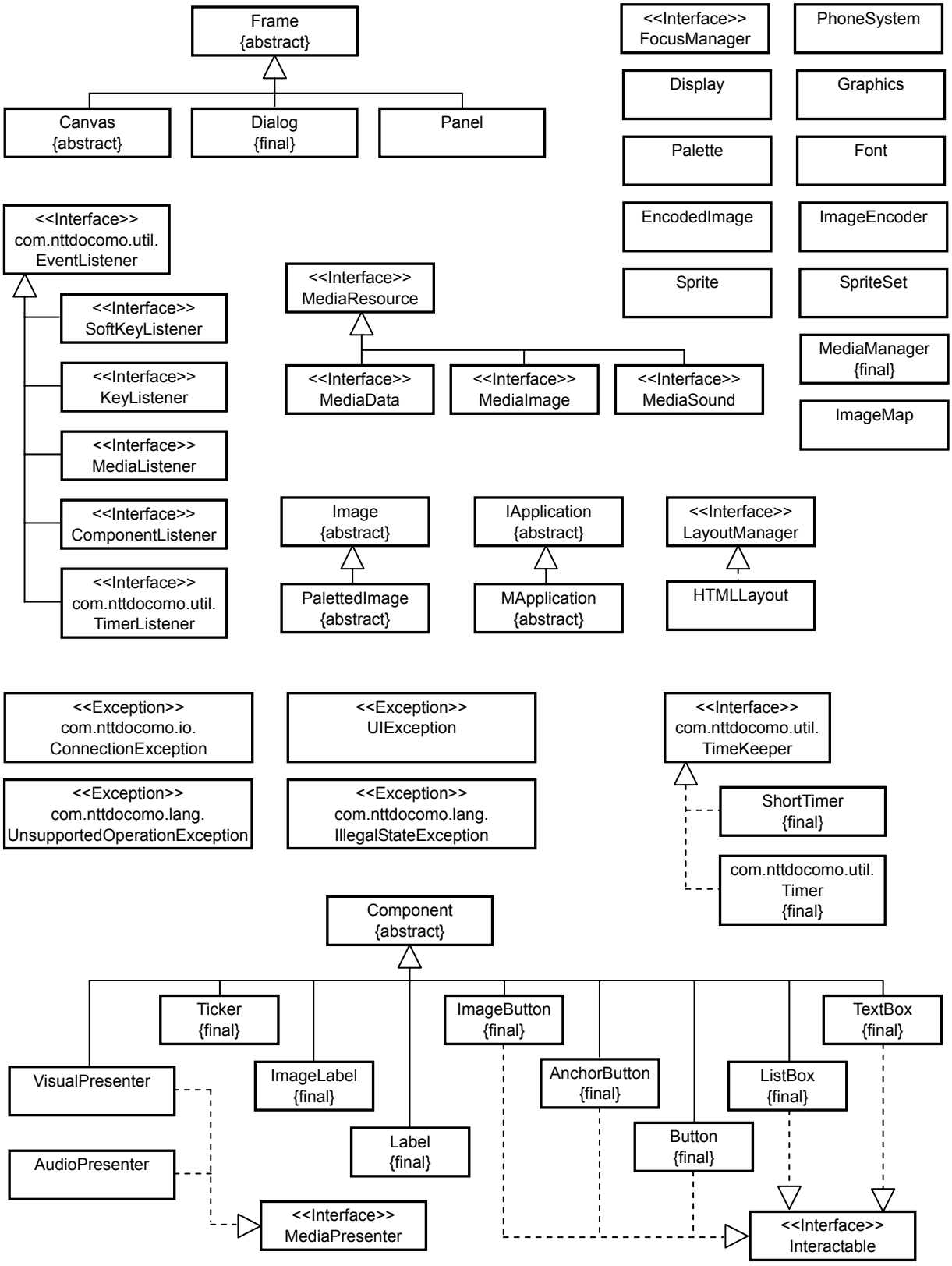


Figure 7: UI Class Hierarchy

4.1 4.1 Programming with High-level UI Components

The i-αppli Runtime Environment provides content developers with nine basic UI components:

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

The other basic UI component is the `Dialog`. The behavior of these elements will be nearly the same across different manufacturers, but the look and feel may be different. This is parallel to the characteristics of standard GUI-applications developed for the Windows, Macintosh, and UNIX computer platforms. The applications behave the same across the different platforms, but the look and feel is different. For example, button shapes on different platforms might appear rounded or square, and might have a thick or thin border. This is same for mobile phones. Low-level praphic routines may generate different appearances depending on the manufacturer. The default layout, scrolling, and focus management strategies will be roughly the same for all manufacturers. However, each manufacturer may choose to extend layout, scrolling, or focus management for their specific mobile phone. The following sections describe general descriptions of the elements and layout manager, and the detailed operations of various models of mobile phones may be different depending on each model.

The i-αpplis Runtime Environment contains a `Component` class which defines the common characteristics of the UI components. However, this class is prepared for the mobile phone manufacturers to provide the above basic UI components. Developers cannot define or use their own components inheriting the `Component` class.

[DoJa-2.0]

The components `AnchorButton` and `ImageButton` are newly added with DoJa-2.0 Profile.

4.4.1 Using the Panel

The `Panel` class acts as a simple container for other UI elements. Only one panel can be displayed on the screen at any given time. This does not restrict an i-αppli to defining only one panel. Most i-αpplis will consist of multiple panels. The user will navigate between these panels during the normal operation of the i-αppli.

It is possible to specify a title for `Panels` which indicate the corresponding function or purpose.

4.1.2 Using the Dialog

The dialog is used to display a text message to alert or inform the user, and/or prompt the user to make a decision. Dialogs are displayed on the screen overlapping the `Panel` or `Canvas`. The `Dialog` class extends Frame. The `Dialog` consists of a title, a text message, and at least one button. Only one `Dialog` can be displayed at a time.

The following types of Dialogs are available:

- Information
- Warning
- Error
- Yes/No
- Yes/No/Cancel

Notes:

- An exception will be thrown when an attempt is made to display more than one Dialog at the same time.
- Dialogs are modal.
- Special care should be taken on the application program side so that any background threads do not interfere with the modal operation of a Dialog. For instance, the resulting behavior when a background thread attempts to change the background Panel or Canvas is device-dependent.
- Although in the same way as for `Panel` and `Canvas`, `Dialog` is a sub-class of the `Frame` class, it is not possible to display a dialog using the method `Display.setCurrent()` (i.e., the method used to change the frame being displayed on-screen). Rather, the method `Dialog.show()` is used to display the `Dialog` screen.

4.1.3 Using Components

The UI components available (i.e., the sub-classes of `com.nttdocomo.ui.Component`) are: `Label`, `ImageLabel`, `Button`, `AnchorButton`, `ImageButton`, `TextBox`, `ListBox`, `Ticker`, and `VisualPresenter`.With the exception of `AnchorButton`, it is not possible for the size of the component to exceed the size of the screen (or in cases where a title is set for a `Panel`, the size of the screen not including the title area).

A `Panel` can be added only once (`Panel.add()` method) for one component's object.

The following section will provide a description of each UI component.

[DoJa-2.0]

With DoJa-1.0 Profile, the operation resulting from specification of a component size in excess of the screen size is depended on the manufacturer in question.

Label

The `Label` class is a component for displaying a single line of text on the `Panel`. The `Label` component does not have a border and it displays a string of characters. The text inside the `Label` component can be aligned left, center, or right. The following code sample shows the `Label` component:

Example: Label component

```
package uidemo;

import com.nttdocomo.ui.*;

/**
 * Class that demonstrates features of a label widget and
 * the setSize method that is called for it.
 * This class creates one button and two label widgets.
 * To demonstrate the feature of visibility and invisibility of a label widget,
 * use a button widget.
 */
public class LabelDemo extends UIDemoPanel {
       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);   // sets label size.

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

Notes:

- A `Label` component cannot receive focus.

ImageLabel

The `ImageLabel` class is a component used to display images on the `Panel`. The `ImageLabel` component can only display a static image. `ImageLabels` for which no static images are set are filled in by the background color of the component. The standard `Swing` classes of the Java GUI can create icons by setting images to Label components, but in the i-αppli Runtime Environment, a separate component has been defined. The following code sample shows the `ImageLabel` component:

Example: `ImageLabel` component

```
import com.nttdocomo.ui.*;
import com.nttdocomo.io.*;

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

              // get the media image
              MediaImage mi =
                   MediaManager.getImage("resource:///UIDemo/img/cup.gif");
              try {
                  mi.use();
               } catch (ConnectionException ce) {
                  // Handle resource I/O exceptions here
```

```
                }
                Image im = mi.getImage();

                // create an imagelabel with an image object.
                l1 = new ImageLabel(im);
                // add the imagelabel to the panel
                add(l1);
        }
        …
}
```

Notes:

- An `ImageLabel` component cannot receive focus.
- In DoJa-2.0 Profile, a `Graphics` object extracted from `Image` can be used to render to an image. When using this functionality to render to the image of an `ImageLabel` already being displayed on the screen, the contents may not be immediately reflected by the display, depending on the phone's manufacturer. In such situations, the rendered content can be reflected to the screen by rerendering all components, such as using the `ImageLabel.setImage()` method to reset the image.

Button

The `Button` class is a component that represents buttons, and implements the `Interactable` interface for receiving user actions. The `Button` component can be set to display text. This component can be set to enabled or disabled. The following code sample shows the Button component:

Example: `Button` component

```
import com.nttdocomo.ui.*;

public class ButtonDemo extends UIDemoPanel implements ComponentListener {
        Button b1, b2, b3, b4;
        ButtonDemo() {
                    // create four buttons and add them to the panel
                    b1 = new Button("B4 enabled");
                    b2 = new Button("B4 disabled");
                    b3 = new Button("ChangeB4");
                    b4= new Button("B4");

                    this.add(b1);
                    this.add(b2);
                    this.add(b3);
                    this.add(b4);
                    …
           }

           …
}
```

AnchorButton

The `AnchorButton` class is a component which represents HTML anchor-type buttons (i.e., underlined text which can operate as a button), and it implements the `Interactable` interface required to receive user operations. With `AnchorButton` components, it is possible to set both text and an image to be used as a heading. If the specified text is longer than the width of the component, this text will automatically be wrapped onto several subsequent lines.

The vertical size of `AnchorButton` is not limited by the vertical size of the screen. However, its horizontal size is limited by the horizontal size of the screen.

This component can be set to enabled or disabled. If an `AnchorButton` has been rendered disabled, its text will not be displayed with an underline, and it will have the same look and feel as regular text. For this reason, it is also possible to use disabled `AnchorButton` components to present long text passages to the user within a Panel.

The following code sample shows the `AnchorButton` component.

Example: `AnchorButton` component

```
import com.nttdocomo.ui.*;

public class AnchorButtonDemo extends UIDemoPanel implements ComponentListener {
       AnchorButton anchorButton1, anchorButton2;
       AnchorButtonDemo() {
                  // Create an AnchorButton capable of operation.
                  anchorButton1 = new AnchorButton("Jump to the next page.");
                  this.add(anchorButton1);
                  // Create an AnchorButton incapable of operation, and use this to present
          text to the user.
                  anchorButton2 = new AnchorButton(
                           “An AnchorButton setup to be incapable of operation can be” +
                           “used to present long text passages to the user.”);
                  anchorButton2.setEnable(false);
                  this.add(anchorButton2);
                  …
          }

          …
}
```

Notes:

- In DoJa-2.0 Profile, a `Graphics` object extracted from `Image` can be used to render to an image. When using this functionality to render to the image of an `AnchorButton` already being displayed on the screen, the contents may not be immediately reflected by the display, depending on the phone's manufacturer. In such situations, the rendered content can be reflected to the screen by rerendering all components, using the `AnchorButton.setImage()` method to reset the image.

ImageButton

The `ImageButton` class is a component which can paste images, and it implements the `Interactable` interface required to receive user operations. An `ImageButton` for which no valid image has been set will be painted with the component's background color.

This component can be set to enabled or disabled.

The following code sample shows the `ImageButton` component.

Example: `ImageButton` component

```
import com.nttdocomo.ui.*;
import com.nttdocomo.io.*;

public class ImageButtonDemo extends UIDemoPanel implements ComponentListener {
       MediaImage mImage;
       Image image;
       ImageButton imageButton;
       ImageButtonDemo() {
                  // Create an ImageButton and add it to the panel.
                  try {
                         mImage = MediaManager.getImage("resource:///label.gif");
                         mImage.use();
                         image = mImage.getImage();
                  } catch (ConnectionException ce) {
                         …
                  }
                  imageButton = new ImageButton(image);
                  this.add(imageButton);
                  …
          }

          …
}
```

Notes:

- In DoJa-2.0 Profile, a `Graphics` object extracted from `Image` can be used to render to an image. When using this functionality to render to the image of an `ImageButton` already being displayed on the screen, the contents may not be immediately reflected by the display, depending on the phone's manufacturer. In such situations, the rendered content can be reflected to the screen by rerendering all components, such as using the `ImageButton.setImage()` method to reset the image.

ListBox

The `ListBox` class is a component that represents buttons, and implements the `Interactable` interface for receiving user actions. The `ListBox` component has a rectangular border, and is internally scrollable. If all of the items to be displayed in the do not fit on the screen, an indicator appears to represent vertical scrolling. The following types of `ListBox` are available:

- Single-selection list
- Multiple-selection list
- Radio button
- Check box
- Numbered list
- Option menu (pop up)

Depending on the type of `ListBox` that you see, items contained inside the frame have different appearances and selection methods, as follows:

Type	Selection	
	Single selection	Multiple selection
List	Single selection list (SINGLE_SELECT)	Multiple selection list (MULTIPLE_SELECT)
Numbered list	Numbered list (NUMBERED_LIST)	None
Check box	None	Check box list (CHECK_BOX)
Radio button	Radio button list (RADIO_BUTTON)	None
Pop-up	Option menu (CHOICE)	None

This component can be set to enabled or disabled. The following code sample shows the ListBox component:

Example: `ListBox` component

```
package uidemo;
import com.nttdocomo.ui.*;

/**
 * Class that demonstrates ListBox widget features.
 * The list types that are demonstrated are radio_button, numbered_list,
 * checkbox, choice, single_select, and multiple select.
*/
public class ListBoxDemo extends UIDemoPanel 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");
              // create a listbox of type RADIO_BUTTON and add some items to it.
              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");
              // Create a listbox of type CHECK_BOX, and add some items to it.
              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");
              // Create a listbox of type NUMBERED_LIST, and add some items to it.
              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");
                // Create a listbox of type SINGLE_SELECT, and add some items to it.
                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");
                // Create a listbox of type MULTIPLE_SELECT, and add some items to it.
                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");
                // Create a listbox of type CHOICE, and add some items to it.
                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);

                //set the softkey listener on this panel
                this.setSoftLabel(Frame.SOFT_KEY_2, "List");
                setSoftKeyListener(this);
        }
        …
}
```

Notes:

- The numbered list is a special format of single-selection list. Users can select items from the top to the 9th of the numbered list by using the numeric keys on the mobile phone. To select the first item, use the number 1 key; to select the second item, use the number 2 key; and so on. The user selects the tenth and subsequent items using the same operations as in the single-selection list.
- Option menus (pop ups) are also a special type of single-selection list. When the mobile phone's "select" button is pressed while the option menu has focus, a pop-up appears. The user selects the desired item from this pop up. Unlike with other types of `ListBox`, the first item on the option menu starts off selected by default.
- In a ListBox that has been added to the current frame, when item setting operations (`setItems()`, `append()` etc.) from the application program and user operations with regards to that `ListBox` are performed simultaneously, the

behavior varies depending on the manufacturer. When performing item setting operations from the application program on a ListBox in the current frame, use the setEnabled() method to disable user operations.

TextBox

The `TextBox` class is a component that represents an input field, and implements the `Interactable` interface for receiving user actions. The `TextBox` component displays single or multiple lines of text. This is the only component that accepts text input. When the edit mode of `TextBox` is enabled by the application program, users can launch the IME (Input Method Editor) to enter or change text. When the edit mode is disabled, the `TextBox` becomes read-only.

The `TextBox` constructor specifies either `DISPLAY_PASSWORD` or `DISPLAY_ANY` as its display mode parameter. An i-αppli requiring text input to be hidden would use a `DISPLAY_PASSWORD` mode, which displays all characters as the '*' character.If the display mode parameter is set to `DISPLAY_ANY`, text entered in the `TextBox` component is displayed as-is.

The initial input mode (e.g. kana/kanji input, alphabetical input etc.) can be changed by calling the `setInputMode()` method with the corresponding initial input mode parameter of `KANA` (kana/kanji input mode), `ALPHA` (alphabetical input mode), or `NUMBER` (numeric input mode). In any `TextBox` component for which the display mode is `DISPLAY_ANY`, the initial input mode is `KANA` in a state in which the method `setInputMode()` is not called.

This component can be enabled or disabled. The following code sample shows the `TextBox` component:

Example: `TextBox` component

```
import com.nttdocomo.ui.*;

public class TextBoxDemo extends UIDemoPanel implements ComponentListener {
       TextBox t;               // single line textbox (textfield)
       TextBox t2;              // multiple line textbox (textarea)
       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);
                  …
       }
       …
}
```

Notes:

- In situations where the user is editing text using the TextBox (i.e., when the IME is active), care must be taken with the application program to ensure that the background thread does not interfere with the IME's model operations. For example, the operations occurring when the IME is launched and when the background thread tries to change the back `Panel` to a different frame will differ from model to model; accordingly, the appropriate care should be taken.

[DoJa-4.1]

In DoJa-4.1 Profile and later, the `setInputSize()` method was added to limit the text length (the length of text that is set for the `setText()` method, etc. or entered at IME), which is set for TextBox.Note that the `setInputSize()` method is optional and may not be supported depending on the manufacturer.

Ticker

The `Ticker` class is a component for displaying streaming character strings on the `Panel`. The `Ticker` component displays text in a ticker tape format. The string scrolls across the screen from right to left.

The following code sample shows the Ticker component.

Example: `Ticker` component

```
package uidemo;
import com.nttdocomo.ui.*;

// Class that demonstrates the ticker widget feature.
// When the panel is displayed for the first time, the default character string of a ticker
is moving.
// A user can change the text by pressing the text button.
// A ticker can be set to be either visible or invisible.

public class TickerDemo extends UIDemoPanel implements ComponentListener {
         Ticker ticker;          // Ticker widget
         Button textButton;      // Change of the text

         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);
         }



         // Showing that an event occurred at Component c.
         //
         // @param c Component. The component where this event occurred.
         // @param type Event type
         // @param param Event parameter
         public void componentAction(Component c, int type, int param) {

                 // A component action has occurred. When textButton is clicked,
                 // the character string of the ticker is changed to something else.
                 if (c == textButton) {
                         if (whichText==1) {
```

```
                                whichText=2; ticker.setText(textString2);
                            } else if (whichText==2) {
                                whichText=1; ticker.setText(textString1);
                            }
                        }
                    }
}
```

Notes:

- There is no way to control the scrolling behavior of the text inside the component from the application program.
- A `Ticker` component cannot receive focus.
- The speed of the scrolling text is manufacturer dependent.

VisualPresenter

The `VisualPresenter` class is a component for presenting dynamic images, and implements `MediaPresenter`. The `VisualPresenter` component is a high-level API component that plays back media data (e.g. GIF animations or FOMA iMotion format data) on the screen. This component can use `MediaListener` to process events related to media data playback. For more details on event processing, see Section 4.3. For media playback code examples, see Section 4.4.

Notes:

- The VisualPresenter is a component, and can only be used from the high-level API.
- A VisualPresenter component cannot receive focus.

4.1.4 Placing Components

Developers can either leave the placement of components to the layout manager, or disable the layout manager and specify the placement position of each component using the `setLocation()` method of the `Component` class.

The layout manger is used by setting in the `Panel`.To set the layout manager in the `Panel`, use the method `Panel.setLayoutManager()`.

Two different layout managers are provided in the i-αppli Runtime Environment — namely, the default layout manager as setup in the initial condition for `Panel` generation, and the HTML layout manager for creating layouts with a HTML-type specification method.

The default layout manager is used principally for simple, flow-type layouts. Components are added to the panel horizontally in sequence from left to right, and when no screen width remains to fit the next component, it is placed in the next available position (i.e., in the next row).

Furthermore, with the HTML layout manager, specifications such as line-break (i.e., the BR tag), new paragraph (i.e., the P tag), and alignment (i.e., the CENTER or DIV tag) can be implemented during the addition of components to Panel, thus enabling the developer to control the layout in a more explicit manner.

Developers should keep their component layout simple, in order to create general-purpose i-αpplis that are implementable on any manufacturer’s mobile phone. Note that the number of components that can be placed will be affected by the memory utilization and the specific limitations of each device.

Notes:

- The `LayoutManager` interface is provided by a manufacturer to enable layout-manager implementation, and a developer cannot use this interface to create his or her own layout manager.The Panel object's set`LayoutManager()` method can be called to switch to the layout manager used by that Panel.
- To disable the layout manager, call the `setLayoutManager()` method with null passed as the parameter on the `Panel` object. If the layout manager is disabled, the application program uses the `setLocation()` method of each component to specify the location of each component one at a time. Note that if the layout manager is disabled, scrolling will also be disabled on some devices.
- A single HTML layout manager object cannot make settings for two or more `Panels` at the same time. If one i-αppli contains a number of Panels for which it is desired to apply HTML layouts, please ensure that individual HTML layout managers are allocated to each of these Panels.
- Whenever an HTML layout manager is set to a Panel using the `setLayoutManager()` method, the initial condition will be adopted (i.e., left alignment with no line-break or paragraph settings).

The following code sample shows the positioning of components using the HTML layout manager.

Example: Component placement with the HTML layout manager

```
import com.nttdocomo.ui.*;

public class HTMLLayoutDemo extends UIDemoPanel {
         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");

                 // Generate the HTML layout manager and set it to Panel.
                 lm = new HTMLLayout();
                 setLayoutManager(lm);

                 // While specifying the layout, add Label to Panel.
                 add(label1);
                 // Place the subsequent components to the following line
                 // (equivalent to the BR tag).
                 lm.br();

                 add(label2);
                 // Place the subsequent components to the following line
                 // (equivalent to the BR tag).
                 lm.br();

                 // Display the components that are added between begin() and the
                 // corresponding end() centered.
                 lm.begin(HTMLLayout.CENTER);
                 add(label3);
                 lm.end();

                 // Display the components that are added between begin() and the
                 // corresponding end() flush-right.
                 lm.begin(HTMLLayout.RIGHT);
                 add(label4);
                 lm.end();
         }
}
```

The following code sample shows the layout manager being disabled, and the application program independently placing components.

Example: component placement with `setLocation()`

```
import com.nttdocomo.ui.*;

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

         NullLayoutDemo() {
                 // Use all components at this point.
                 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);

                 /*
                  * Create this button as an example of a component placed
                  * outside of the screen.
                  */
                 offScreen = new Button("This button is never seen");

                 /*
                  * At this point, set null to the layout manager. Therefore, it is
                  * necessary to set the location of each component. Note: The
                  * components that are placed outside of the coordinates
                  * in the physical screen are not displayed at all. Because of the
                  * null layout, the scrolling may be disabled.
                  */
                 setLayoutManager(null);

                 /* Note: Even in this case, the order of adding a component has its meanings.
                  * A component is given focus according to this order.
                  */
                 add(title);
                 add(button1Label);
                 add(button1);
                 add(button2Label);
                 add(button2);
                 add(text);
                 //Add this component too.
                 add(offScreen);

                 /*
                  * It is necessary to set the location of each component at this point.
                  * In other words, specify the layout of the panel uniquely. For the
                  * size of a component, it is possible to set automatically, or
                  * to directly call setSize and specify the size.
                  */

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

```
                button2.setLocation(45, 40);
                text.setLocation(2, 60);

                /*
                 * Note: This button is placed outside of the screen. Therefore, it is
                 * not displayed and it does not receive focus.
                 */
                offScreen.setLocation(1000, 1000);
        }
}
```

[DoJa-2.0]

The HTML layout manager (i.e., `HTMLLayout` class) is newly added with DoJa-2.0 Profile.

4.1.5 Focus and Scrolling

The actual details of how focus and scrolling management is handled is dependent upon the manufacturer. i-appli compatible mobile phones all have a simple default focus manager installed. With the typical default focus manager, the order in which components are added to the `Panel` will be the order in which the components receive focus. When an i-αppli is launched and the screen is displayed, the first interactable and enabled component will have focus. However, when a component with focus is positioned outside the display range, it will not be possible to operate that component.

The default scroll operations are tightly knit with the focus manager. This is due to the fact that scrolling the screen happens together along with the movement of focus.

Notes:

- The `FocusManager` interface is provided by a manufacturer to enable focus-manager implementation, and a developer cannot use this interface to create his or her own layout manager.The `Panel` object's `setFocusManager()` method can be called to switch to the focus manager used by that `Panel`. However, no focus manager other than the default focus manager has been defined for previous profiles.

  Note that unlike the layout manager, the focus manager cannot be disabled.

4.1.6 High Level Event Processing

On a screen consisting of a `Panel`, events created by the components on that panel are passed to the `componentAction()` method of an object implementing `ComponentListener`. Similarly, softkey events are passed to the `softKeyPressed()` and `softKeyReleased()` methods of an object implementing `SoftKeyListener`. In this way, a listener model is employed for high level event processing.

Both the `ComponentListener` and `SoftKeyListener` are registered to the `Panel` before use. See Section 4.3 for a detailed code sample of the event listener for high-level event processing.

4.1.7 Font Support in Components

In DoJa-3.0 Profile, the `setFont()` method has been added to the `Component` and `Dialog` classes, allowing the font of the characters displayed in these to be specified from the application program. By creating a `Font` object for the font to be used and specifying it in the arguments of the `setFont()` method, fonts other than the default font can be displayed.

When a font is set to a component, this method will use it to display all character strings on that single component. Character strings composed of multiple, different fonts cannot be displayed on the same component. If this method is called on a component that has no functionality for displaying characters, nothing happens.

When a font is set to a dialog, this method will use it to display only the text body. The font for the dialog's title and buttons cannot be changed.

Notes:

- The display character fonts for the panel titles and softkey labels cannot be changed.

4.2 Programming with Low-level UI API

Applications like games sometimes have a wider proliferation of graphic rendering features than interfaces using only components. This section discusses how to use the low-level API and handle events, mainly for graphic rendering.

4.2.1 Using the Canvas

The `Canvas` is a screen area used for drawing graphics. Developers can subclass the `Canvas` class and override the `paint()` method to draw inside the `Canvas`.Whenever the system determines that `Canvas` rendering is necessary or when an application program specifically requests re-rendering (i.e., when the `repaint()` method is called), the method `paint()` will be called back by the system.

As with the `Panel` object, a single `Canvas` object is the equivalent of one screen. Therefore, you cannot display multiple `Canvas`es on a single screen, nor can you display both a `Canvas` and a `Panel` at the same time on the same screen. It is possible, however, to alternate between multiple `Canvas`es and `Panel`s on a single screen.

Following is a simple example of drawing on the `Canvas`.

Example: Drawing on a `Canvas`

```
import com.nttdocomo.ui.*;
public class CanvasDemo extends Canvas {
         int x = 20;
         int y = 10;

         CanvasDemo() {
                 // Create softkey label “back”.
                 setSoftLabel(Frame.SOFT_KEY_1, "Back");
         }

         /**
          * Draw a string and a rectangle in the canvas area.
          */
         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);
         }

         …
}
```

Notes:

- The `repaint()` method is used to specify the need for rendering to the system, and it is not possible to guarantee that a single calling of the `repaint()` method will correspond to a single calling of the `paint()` method. When considered from the application program, therefore, there will be cases where several consecutive callings of the `repaint()` method are grouped together in a single calling of the `paint()` method. Even in situations where a developer can group `paint()` methods into a single calling, it will be necessary to describe the `paint()` methods so that screen rendering may be carried out correctly.
- In DoJa-3.0 Profile, calls to the `repaint()` method on a `Canvas` that isn’t set to the current frame are ignored. In profiles older than DoJa-2.x Profile, behavior depends on the manufacturer when the `repaint()` method is called on a `Canvas` that isn’t set to the current frame.

[DoJa-5.0]

For DoJa-5.0 Profile and later, depending on the manufacturer, by using the `DrawArea` key of ADF, a display area may be used wider than the default display area (i.e., when not using the `DrawArea` key). For such models, by using the method `Canvas.setSoftLabelVisible()`, the softkey label display may be switched on and off when displaying the `Canvas`. Note that although the method `setSoftLaelVisible()` is defined in the class `Frame`, this method cannot be called for frame objects (`Panel` and `Dialog`) other than `Canvas` in the current profile.

4.2.2. Drawing Graphics

A graphics object (of the `Graphics` class) is used to draw lines, shapes, and text. Because these objects also include information about the drawable area, font, color, etc., they are also called "graphics contexts". In the previous section's example we used a graphics context "g", and called the `drawString()` method after clearing the screen with the `clearRect()` method to draw some text. Then we called the `drawRect()` to draw a rectangle to the screen.

The `lock()` and `unlock()` methods shown in the example above are supplied by the `Graphics` class to control the double-buffering behavior of graphics rendering. Double buffering eliminates flickering by painting into an offscreen buffer that is subsequently copied onscreen. Therefore, when double buffering is used, no flicker occurs.

Some manufacturers do not support double buffering. On devices that do not support double buffering, these two methods have no effect. On devices that support double buffering, after the `lock()` method is called (for one or more times), painting to the screen will be temporarily disabled until the `unlock()` method is called once with a `true` argument, or for a matching number of times with a false argument.

Notes:

- The actual double buffering implementation used differs based on the mobile phone model. In order to ensure the correct operation of your application programs, always adhere to the following guidelines when using double buffering.
    - Do not intermix calls to the `lock()` and `unlock()` methods of different `Graphics` objects.
    - You must unlock a `Graphics` object before your `paint()` method returns.

  The following code snippet shows correct usage of the lock/unlock methods.

  Example: Correct implementation (in accordance with the guidelines above)

```
import com.nttdocomo.ui.*;
public void paint(Graphics g) {
     Graphics g2 = g.copy();
     g.lock();
     // draw some stuff...
     g.unlock(false);
     g2.lock();
     // draw more stuff...
     g2.unlock(false);
}
```

  The following code snippet shows incorrect usage of the lock/unlock methods.

  Example: Incorrect implementation

```
import com.nttdocomo.ui.*;
public void paint(Graphics g) {

     Graphics g2 = g.copy();
     g.lock();
     // draw some stuff...
     g2.lock();              // intermixing g's lock with g2's
     // draw more stuff...
     g.unlock(false);        // g2 is not unlocked
}
```

    - Do not issue the `Graphics.dispose()` method for graphics objects passed as the arguments of the `paint()` method called back from the system.Where necessary, dispose of graphics objects is performed by the i-αppli Runtime Environment that is a caller of the method `paint()`. In addition, when an application program obtains a graphics object from a `Canvas` object or an `Image` object by using the

`getGraphics()` method, make sure to call the `dispose()` method when the graphics object is no longer necessary.

- With the i-αppli Runtime Environment, it is possible to set a rectangular area using the parameters of the `repaint()` method to facilitate re-rendering only within the specified rectangular area. Although it may be assumed for DoJa-1.0 Profile that certain mobile phones may not provide support for this function with the rendering device's clipping rendering capability, all mobile phones compatible with DoJa-2.0 Profile will support this function.
- The i-αppli Runtime Environment supports transparent GIFs, and if an image object generated from a transparent GIF media image is drawn, the transparent part will be out of target to draw. However, the effect is only when an image object generated from a transparent GIF is drawn, and the transparent information will be lost from the draw results.
- With DoJa-2.0 Profile, when a `Canvas` is redisplayed as a result of the suspension or resume of an i-αppli, it is guaranteed that the `paint()` method for the corresponding `Canvas` will be called once from the i-αppli Runtime Environment. However, in cases where calling of this `paint()` method does not result in complete restoration of the entire screen (i.e., where the application program carries out only partial rendering in accordance with events), the degree to which the system can restore the screen will depend on the model in question.

  In DoJa-3.0 or later profile, when i-αppli resumes, if the system can completely restore the rendered Canvas contents, the paint() method will not be called.
- In DoJa-2.0 Profile, the default font used for low-level string rendering is 12 dot font size. However, in DoJa-1.0 Profile the default font size is manufacturer dependent. Note that regardless of the profile supported, support for fonts other than the default font will vary with respect to manufacturer. In general, because it is necessary to store font infromation in the cellular-phone's limited memory, the actual number of fonts installed is limited to a few varieties. DoJa-1.0 and DoJa-2.0 Profiles support Font.SIZE_MEDIUM, the default font size.

  In DoJa-3.0 or later profile too, in consideration of keeping application program compatibility between profiles, the default font is a 12 dot font. However, as cellular phones are now equipped with high-precision LCD devices, the default font size has been changed to support `Font.SIZE_TINY`, which has been newly added in DoJa-3.0 Profile. In DoJa-3.0 or later profile, the medium font is a 24 dot font (sizes for fonts other than tiny and medium may vary depending on the manufacturer).

  Due to the advance of high-precision LCD devices, the size of a font with the same number of dots appears smaller, so in designing screen-layouts, use of a larger font should be considered depending on the situation.

  [DoJa-5.0]

  In DoJa-5.0 Profile, some models may appear which support font sizes other than the pre-existing sizes (`SIZE_TINY`, `SIZE_SMALL`, `SIZE_MEDIUM`, and `SIZE_LARGE`). On these models, you can use the Font.getSupportedFontSizes() method to retrieve a list of supported font sizes for that model. (On models which only support the pre-existing font sizes, this method cannot be used to retrieve any such information.)

  To use font sizes other than the pre-existing ones, you can use the `Font.getFont(int, int)` method which was newly added in DoJa-5.0 Profile. When you directly specify a numerical font size for this method it will return a Font object which corresponds to that size.
- Regarding graphics rendering in DoJa-2.0 Profile and later, functions belonging to i-αppli Options and i-αppli Extensions (sprite and 3D graphics) were newly defined. For information on these, refer to the "i-αppli Content Developer's Guide: i-αppli Options and Extensions", and the "i-αppli Content Developer's Guide: Option and Extension API Reference". Among these are items that were adopted into the i-αppli standard API in the procedure of profile upgrades.
- In DoJa-3.0 or later profile, rendering to the graphics object associated with a `Canvas` only occurs if that Canvas is set to the current frame. If the associated Canvas is not set to the current frame, the call to the rendering method for that graphics object will be ignored.

4.2.3 Rendering to an Image Object

Support for rendering to Image objects is provided with DoJa-2.0 Profile or later.

The acquisition of `Graphics` objects from `Image` objects is carried out by calling the `Image.getGraphics()` method. Rendering specification for a `Graphics` object acquired from an `Image` object is reflected in the display content of the original `Image` object. In this way, the rendered `Image` object can be set as an `ImageLabel` or as an `ImageButton` to allow the results of rendering processing by the application program to also be displayed when using `Panels`.

To retrieve a `Graphics` object from an Image object using the `Image.getGraphics()` method, the Image object must be a new Image object created via the `Image.createImage()` method. If the `getGraphics()` method is called with respect to an Image object acquired using the `MediaImage.getImage()` method, an `UnsupportedOperationException` will be thrown.

Images created using `Image.createImage()` will be filled with the default background color of the `Canvas` for that particular model. However, the `Image` itself has no concept of a background color. In order to specify the clearing of rendering content for `Graphics` acquired from an Image, rather than using the `Graphics.clearRect()` method, the `Graphics.fillRect()` method is used after first setting the appropriate color.

The following code sample shows the usage of an Image object rendered in accordance with application program specifications as an `ImageLabel`.

Example: Rendering to an `Image` object

```
import com.nttdocomo.ui.*;

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

          ImageDrawDemo() {
                  // Create new Image object and acquire the Graphics object.
                  int width  = 100;
                  int height = 50;
                  img = Image.createImage(width, height);
                  g = img.getGraphics();

                  // Render in the acquired Graphics object.
                  g.lock();
                  g.drawRect(0, 0, width-1, height-1);
                  g.drawString("Rendering sample", 5, 16);
                  g.drawString("to Image", 5, 32);
                  g.drawString("via Graphics", 5, 48);
                  g.unlock(true);

                  // Set to the Image object ImageLabel and display on Panel.
                  iLabel = new ImageLabel(img);
                  add(iLabel);
          }
}
```

Notes:

- It is guaranteed in all models that the `Image.createImage()` method can generate Image, for at least the size of a square in which each side is as long as the length of the native draw area for that model. Please be aware that the

ability to render `Image` which are larger than the mobile phone's rendering area will depend on the manufacturer's specifications.

[DoJa-2.0]/[DoJa-3.0]

It is guaranteed in all models that the `Image.createImage()` method can generate Image at least in size of the draw area for the i-αppli in a model in the DoJa-2.x and the DoJa-3.x Profiles. Whether or not a bigger `Image` can be generated depends on the manufacturer.

4.2.4 Color Support

A Graphics object contains information about the color of objects.The class Graphics contains the following methods, and each of them returns an integer representing the color.

`getColorOfName(int name)`	:returns an integer value corresponding with a color name (Graphics constant).
`getColorOfRGB(int r,int g,int b)`	:returns an integer value corresponding with a RGB value.
`getColorOfRGB(int r,int g,int b,int a)`	:returns an integer value corresponding with a RGB value and an alpha value.

Half transparent rendering (blend of a pixel-color for the rendering source and a pixel-color for the rendering location) is enabled by using the color specification including an alpha value. A value from 0 to 255 is specified to alpha value "a", and 255 means it is completely opaque. However, the only rendering methods for which an alpha value element of color is applicable are as follows.

DoJa-4.x Profile: `fillRect()` only

DoJa-5.x Profile: `fillRect()`, `fillArc()`, and `fillPolygon()` only

The setting content of the alpha value element becomes unavailable in other rendering methods or color specification methods for high level UI, and it is assumed that 255 is specified in the alpha value.

The method of managing colors is manufacturer dependent, and the integer values (and return values) assigned to colors in the i-αppli Runtime Environment may differ from one manufacturer to the next. The developer should not assume that these integers are always the same. For example, if black has a value of 1 on one manufacturer's black and white screen, the value of black on another manufacturer's color screen might be 255. Application programs must use the methods described above to convert common color definitions across all models (i.e., by name or RGB value) into model-specific color definitions.

The following shows an example of incorrect code.

Example: Graphics Canvas

```
import com.nttdocomo.ui.*;
public class MyCanvas extends Canvas {
       …
       public void paint(Graphics g) {
              …
              if(g.getColorOfName(Graphics.BLACK) == 1) {// Perform some operation.
                  // Even If the result is true for a model of a manufacturer and the
                  // code here is executed, the result may be false for a model of another
                  // manufacturer, and the code may be skipped.
              }

              …
      }
}
```

[DoJa-2.0]

i-αpplis can use color-supporting pictographic chatacters (i.e., from the external character set as defined by NTT DOCOMO for i-mode services). For color-supporting pictographic characters, a default display color is allocated to each individual pictographic chatacter.

The `Graphics` class from DoJa-2.0 Profile or later features an additional `setPictoColorEnabled()` method, and this is used when explicitly indicating the rendering colors of `Graphics` objects to select whether to render with the pictographic character's default display color or whether to render using the rendering color as specified for the `Graphics` object.

When rendering pictographic characters in DoJa-1.0 Profile with the rendering color for `Graphics` objects being explicitly indicated, individual manufacturer specifications will determine whether rendering of the pictographic characters will be carried out with the default display color or whether the rendering color as specified for the `Graphics` object will be used.

[DoJa-4.0]

Color specification using an alpha value (the `getColorOfRGB(int,int,int,int)` method) is new to the DoJa-4.0 Profile.

4.2.5 Low Level Event Processing

The method for low-level event processing as suitable to `Canvas` usage differs from that for high-level event processing for `Panel` usage.The `Canvas` object only handles low-level events. When a developer inherits the `Canvas` class, it is possible to over-write the `processEvent()` method. This method will be invoked with the low-level event and parameters that go along with it. The following table shows valid event types to send to `Canvas`. These event types are defined by the `Display` class.

Event type	Event type value	Parameters
`KEY_PRESSED_EVENT`	0	Value of pressed key
`KEY_RELEASED_EVENT`	1	Value of released key
`RESUME_VM_EVENT`	4	Do not use
`UPDATE_VM_EVENT`	6	Do not use
`TIMER_EXPIRED_EVENT`	7	Timer ID of ShortTimer generated by timer event.

The following code example shows the implementation of the `processEvent()` method for handling low-level events.

Example: Event handling implementation

package spaceinv;

```
import com.nttdocomo.ui.*;
import java.util.Random;

/**
 * Use this class when displaying successive admitted high score, which is set
 * on a specific high score server. The score and its name are displayed in an
 * animated manner. This class moves to the next screen when it receives an input
 * from a user or the idle timer maintained by this class reaches the set time.
 */
public class HighScoreScreen extends Canvas implements Runnable {
```

```
        private boolean done;
        private ShortTimer timer;

        /** Create a new instance of this class. */
        public HighScoreScreen() {
                …
        }
        /** reset this instance to the initial state. */
        public void reset(String[] list) {
                …
        }
        /** required for the Runnable interface.*/
        public void run() {
                …
        }
        /** display this screen.  */
        public void paint(Graphics g) {
                …
        }

        /** override processEvent() of Canvas to perform even processing. */
        public void processEvent(int type, int param) {
                if (type == Display.KEY_PRESSED_EVENT ||
                        type == Display.TIMER_EXPIRED_EVENT) {
                        done = true;
                        timer.stop();
                        timer.dispose();
                        SpaceInvaders.WELCOME.reset();
                        Display.setCurrent(SpaceInvaders.WELCOME);
                }
        }
}
```

[DoJa-2.0]

With DoJa-2.0 Profile or later, the sequence for the occurrence of low-level events upon the resume of an i-αppli is as defined below.

When resuming an i-αppli, the `resume()` method of the i-αppli's main class is called after the `RESUME_VM_EVENT` notification, which indicates the i-αppli has been resumed.

If an `UPDATE_VM_EVENT` indicating the destruction of rendering data occurs upon the resuming of an i-αppli, this will occur after `RESUME_VM_EVENT`.

[DoJa-5.0]

In DoJa-5.0 Profile and later, the class `com.nttdocomo.opt.ui.Display2` was added to define optional events related to a manufacturer's own unique hardware configuration, in addition to the basic low-level events explained here in this section.

For details on the events available in this package, see the "`Display2`" section in the "i-appli Content Developer's Guide: Optional and Extension API Reference".

Notes:

- The `processEvent()` method is called in response to a user action, or to the timer or some other kind of asynchronous event.When calling the `processEvent()` method, be aware that thread allocation is up to the manufacturer. For instance, some manufacturers' devices may process multiple events generated simultaneously with a single thread. Make an effort to wrap up processing inside the `processEvent()` method extremely quickly.

4.2.6 Using the IME in a Canvas

In screens using a `Panel`, the IME can be launched by user `TextBox` operations and character strings entered by the user can be obtained by the i-αppli. In contrast to this, in screens using a `Canvas`, the IME can be launched from the application program using the APIs newly established in the DoJa-3.0 profile. Using these APIs, when using `Canvas` it is possible for the i-αppli to obtain character strings input by the user in the same way as when using Panel.

When the `Canvas.imeOn()` method is invoked from the application program, the IME is launched. This method is non-blocking and returns to the application program when the IME has completed start-up.

When the user finishes the IME operation (by committing or cancelling), the application program receives an IME event notification. The developer can retrieve the IME processing results by overriding the `Canvas.processIMEEvent()` method.

The following code gives an example of a Canvas that uses the IME.

Example: Using the IME from a Canvas

```
import com.nttdocomo.ui.*;

class MainCanvas extends Canvas {
      String inputString = "";

      public void processEvent(int type, int param) {
            // Start IME with a release of the softkey 1.
            if(type==Display.KEY_RELEASED_EVENT&&param==Display.KEY_SOFT1)
                  imeOn("initial string",TextBox.DISPLAY_ANY, TextBox.KANA);
            // When the startup of IME is completed, the control comes back to the application
            program.
      }

      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) {
            // When the user operation for IME is completed, this method is called.
            // For the parameters of an IME event, there are an event type (confirmation
            // or cancel) and an input character string.
            if (type==Canvas.IME_CANCELED) inputString = "[CANCELED]";
            else if (type==Canvas.IME_COMMITTED) inputString = text;
            repaint();
      }
}
```

Notes:

- Caution is needed on the application program side to prevent background threads from interfering with the IME's modal operations when the IME is in the starting-up state. For example, the operations occurring when the IME is launched and when the background thread tries to change the back `Canvas` to a different frame will differ from model to model; accordingly, the appropriate care should be taken.
- Calls to the `imeOn()` method will be ignored when the `Canvas` is not set to the current frame or a Dialog is being displayed.
- The `processIMEEvent()` method call is triggered by the occasion of asynchronous events which occur when the user finishes the IME operation.When calling the `processIMEEvent()` method, be aware that thread allocation is up to the manufacturer. For example, in one manufacturer's model, processing may occur in one thread with respect to

the IME event and the other events. Make an effort to wrap up processing inside the `processIMEEvent()` method extremely quickly.

- The `imeOn()` method cannot be called from an inactive state stand-by application.

[DoJa-4.1]

In DoJa-4.1 Profile and later, a method with an argument to limit the length of text that can be input via the IME was added as a variation of the `imeOn()` method. However, support for this method is optional, and in some cases may not be provided depending on the manufacturer.

4.3 Event Listeners

An event listener is an object registered with an event-notifying object (e.g. `Panel` or `MediaPresenter`) to listen for particular types of events. When events occur, the object constructs the proper event type and notifies the corresponding event listener of this object. There are five types of event listeners:

- `ComponentListener`
- `SoftKeyListener`
- `KeyListener`
- `MediaListener`
- `TimerListener`

There are no event listeners for Dialogs. Use the `Dialog.show()` method to display a dialog on the screen.The `Dialog.show()` method returns control to the application program after the user finishes interacting with the dialog.The return value of the `Dialog.show()` method indicates which button the user pressed in the dialog.

Notes:

- A listener method is called in response to a user action, the timer, or some other kind of asynchronous event. When calling a listener method, be aware that thread allocation is up to the manufacturer. For instance, some manufacturers' devices may process multiple events generated simultaneously with a single thread. Make an effort to wrap up processing inside the listener method extremely quickly.

4.3.1 ComponentListener

The `ComponentListener` interface is used upon stand-by for the launching of events for UI components on the displayed Panel. The interface has only one method defined: `componentAction()`. To define and register the component listener:

- The class used as a listener must implement the `ComponentListener` interface and define the `componentAction`() method. When a component event takes place, the `componentAction()` method is invoked, passing to the component listener the type of event, the component in which the event took place, and any parameters needed.
- The `setComponentListener()` method must be called on the Panel object, and register an instance of the listener class implemented above.

The three types of events defined for the ComponentListener are:

- `BUTTON_PRESSED`:
  Event that is issued when a `Button` is pressed. Does not use event parameters.

- `SELECTION_CHANGED`:
  Event that is issued when the selected item in the `ListBox` has changed. Passes the index of the item that was selected or whose status has changed as the event parameter (If multiple items have been selected, the index of the smallest item is passed).
- `TEXT_CHANGED`:
  Event that is issued when the character input of the `TextBox` is terminated. Does not use event parameters.

Next is an example of defining and using the `ComponentListener` as an inner class:

Example: Inner class implementing ComponentListener

```
package uidemo;
import com.nttdocomo.ui.*;

/**
 * Class that demonstrates features of the label widget and the setSize method that
 * is called for it. This class creates one button and two label widgets.
 * To demonstrate the feature of visibility and invisibility of a label widget,
 * use a button widget.
 */
public class LabelDemo extends UIDemoPanel {
       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);   // sets label size.

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

       /**
         * Internal class that processes button events.
         */
       class ListenerClass implements ComponentListener {
       /**
         * Showing that an event occurred in Component c.
         *
         * @param c  Component where this event occurred.
         * @param type  Event type
         * @param param  Event parameter
         */
              public void componentAction(Component c, int type, int param) {
              /*
              * A component action is executed. If a component is a button, and the
              * count is 0, set the first label of this panel to be invisible, and if
              * the count is 1, set the same label to be visible respectively.
              */
                     if (c == btn1) {
                            if (count == 0) {
                                   lbl1.setVisible(false);
                                   count++;
                            }
                            else {
                                   lbl1.setVisible(true);
                                   count=0;
```

```
                                    }
                                }
                            }
                        }
}
```

Notes:

With `TextBox` and `ListBox`, the methods provided by the corresponding classes can be used to change the text set for the `TextBox` or the `ListBox` selection condition. In this way, the notification of component events upon condition changing is possible using methods and not user operations.

4.3.2 SoftKeyListener

The `SoftKeyListener` interface is used to listen for the two softkeys to be invoked. The interface has two methods defined: `softKeyPressed()` and `softKeyReleased()`. To define and register the softkey listener:

- The class used as a listener must implement the `SoftKeyListener` interface and define the `softKeyPressed()` and `softKeyReleased()` methods. When a softkey is pressed, the `softKeyPressed()` method is invoked, and when one is released, the `softKeyReleased()` method is invoked.
- The `setSoftKeyListener()` method must be called on the Panel object, and register an instance of the listener class implemented above.

Notes:

- Softkeys can also be used from a `Canvas`. In accordance with low-level processing methods, use `Canvas.processEvent()` instead of a softkey listener (as with ordinary keystrokes, `KEY_PRESSED_EVENT` or `KEY_RELEASED_EVENT` is notified).

4.3.3 KeyListener

The `KeyListener` interface is used to listen for all keys besides the softkeys to be invoked. The interface has two methods defined: `KeyPressed()` and `KeyReleased()`. To define and register a key listener:

- The class used as a listener must implement the `KeyListener` interface and define the `keyPressed()` and `keyReleased()` methods. When a key is pressed, the `keyPressed()` method is invoked, and when one is released, the `keyReleased()` method is invoked.
- The `setKeyListener()` method must be called on the `Panel` object, and register an instance of the listener class implemented above.

[DoJa-2.0]

In a `Panel` object, the direction and selection keys are used for tasks such as screen scrolling or component operation. With DoJa-2.0 Profile or later, the following definitions are made for this type of key event when using high-level API.

- Direction keys (i.e., up, down, left, or right) do not generate events. These keys are generally subjected to `Panel` processing in order to facilitate scroll and focus control.
- The selection (or confirmation) key will not generate key events only if the component in focus is being presented on the display. In situations where the component in focus is not being displayed, the selection key will be subjected to `Panel` processing for the component in question.

Note that with DoJa-1.0 Profile, the decision on which key events were to be subjected to `Panel` processing under which conditions was left to the discretion of the individual manufacturers.

4.3.4 MediaListener

The `MediaListener` interface is used when media data is played, to listen for media events (`AudioPresenter` or `VisualPresenter` events). The only method that the interface defines is `mediaAction()`. To define and register the media listener:

- The class used as a listener must implement the `MediaListener` interface and define the `mediaAction()` method. When a media event occurs, this method is invoked.
- The `setMediaListener()` method must be called on the `MediaPresenter` object, and register an instance of the listener class implemented above.

The three types of events defined for the AudioPresenter are:

- `AUDIO_COMPLETE` : Sound media playing complete event
- `AUDIO_PLAYING` : Sound media playing started event
- `AUDIO_STOPPED` : Sound media playing stopped event
- `AUDIO_PAUSED` : Sound media data's pause event
- `AUDIO_RESTARTED` : Sound media data's restart play event
- `AUDIO_SYNC` : Sound media data's synchronization event

The three types of events defined for the `VisualPresenter` are:

- `VISUAL_COMPLETE` : Visual media playing complete event
- `VISUAL_PLAYING` : Visual media playing started event
- `VISUAL_STOPPED` : Visual media playing stopped event

For more details on media data, refer to Section 4.4.

[DoJa-2.0] [DoJa-3.0]

- The only media event guaranteed to occur in DoJa-1.0 Profile is `AUDIO_COMPLETE`. In DoJa-1.0 Profile, whether other media events are supported depends on the manufacturer. Also, media events other than the start play/pause/complete events are not supported in DoJa-1.0 Profile.
- In DoJa-2.0 Profile, the start play/pause/complete events are supported by all models in both `AudioPresenter` and `VisualPresenter`. In DoJa-2.0 Profile, media events other than these are included in the i-áppli Optional API .
- In DoJa-3.0 Profile, all media events mentioned in this section have been adopted into the i-αppli standard API.

4.3.5 TimerListener

The `TimerListener` interface is used to listen for Timer expired events. The only method that the interface defines is `timerExpired()`. To define and register the timer listener:

The class used as a listener must implement the `TimerListener` interface and define the `timerExpired()` method. When a timer expires, this method is invoked.

The `setListener()` method must be called on the `Timer` object, and register an instance of the listener class implemented above.

Notes:

- ShortTimer has no interface which supports this.When using a ShortTimer, handle TIMER_EXPIRED_EVENT events through the Canvas's processEvent() method.

4.4 Using Multimedia Data

4.4.1 Types of Multimedia Data Usable by i-applis

The i-appli Runtime Environment comes with the ability to handle images (static images and video) and sound as multimedia data. Below are the multimedia data formats that the standard i-appli Runtime Environment can handle.

- GIF format image data: DoJa-1.0 or later

  Supports GIF87a and GIF89a. In addition to standard images, interlaced GIF, transparent GIF, and animated GIF is also supported. Animated GIFs can be played as movies using the `VisualPresenter` class. (See Section 4.4.2)

- JPEG format image data: DoJa-2.0 or later (FOMA mobile phones supported this earlier from DoJa-1.0)

  The JPEG format which can be used on all models is JFIF Baseline encoding. Depending on the manufacturer, EXIF is also supported. If data in an unsupported encoding such as Extended Sequential DCT, etc. is used, an exception may occur.

- BMP format image data: DoJa-5.0 or later

  BMP format data that meets the following conditions may be used:

  Must be Windows BMP format, bottom-up DIB

  Must be 1, 4, 8, 16, or 24 bits-per-pixel

  Must be uncompressed (the compression format must be `BI_RGB`; however, if the data is 16 bits per pixel `BI_BITFIELDS` (RGB 5-5-5 or RGB 5-6-5) is also supported)

  BMP format data may only be used within an i-appli; such data cannot be saved to the native image storage area (My Pictures).

- MFi format sound data: DoJa-1.0 or later

  i-mode's standard sound data format. As such, MFi (Melody For i-mode) format sound data is supported. For the basics of playing sounds, see Section 4.4.2. With MFi format sound data you can also use a variety of extended functions such as ADPCM and 3D sound.

- SMF format sound data: DoJa-1.0 or later (FOMA mobile phones only)

  FOMA mobile phones can also use SMF (Standard MIDI File) format sound data in addition to the MFi format. For the basics of playing sounds, see Section 4.4.2.

- iMotion format video data: DoJa-1.0 or later (FOMA mobile phones only)

  FOMA mobile phones support the iMotion format for video content. On such FOMA mobile phones, i-applis are also able to handle iMotion format video data. (See Sections 4.4.2 and 4.4.5) Please note that there are multiple versions of iMotion format video data, and the proper version must be used for the target mobile phone model. A list of which models support which versions of this format will be provided separately by NTT DOCOMO.

4.4.2 Using Presenters

The i-αppli Runtime Environment provides functionality for displaying images including animation and playing sound files. The classes used for playing back visuals and sounds are VisualPresenter and AudioPresenter repectively.

Notes:

- The `VisualPresenter` is a component and, when using the high level API, must be added onto a `Panel` object in order to be used.In `VisualPresenter`, GIF animation is supported as a standard visual that can be played back. In addition, FOMA mobile phones support iMotion format data as movie data (see Section 4.4.5).
- `AudioPresenter` is not a component.`AudioPresenter` can be used when either a Panel or a Canvas is being displayed.
- In the i-αppli development environment for DoJa-2.0 or later profile provided by NTT DOCOMO, `AudioPresenter` supports MFi format and MIDI format data. For more details regarding actual development procedures for use with the i-αppli Runtime Environment, refer to Chapter 15.

  [DoJa-2.0]

  With the i-αppli Runtime Environment supplied by NTT DOCOMO which is compatible with DoJa-1.0 Profile, the use of MFi data is not possible for sound playback. Note that only the MIDI format may be used for sound files in this environment.

- On FOMA mobile phones, certain features can be launched and used while switching tasks between an i-appli, the Browser, a phone call, etc.On these mobile phones, if an i-appli uses any function which handles sound or voice (i.e., playing a sound via `AudioPresenter` or playing an iMotion via `VisualPresenter`) while another task is also using some function which handles sound or voice, an exception (`UIException`) will occur.

  Functions which cannot be used simultaneous with an i-appli's sound/voice related functions are the phone call and music player functions, among others. (On some models, if a sound is played in an i-appli while the music player is running an exception will not occur; instead, a feature exists to switch the sound control between the music player and the i-appli's sound.)

The following example shows how to display an image file using the `VisualPresenter` class, and how to play a sound file using the `AudioPresenter` class.

In this example, resources bundled in the i-αppli's JAR file are used as the image and sound files (using a URL starting with "`resource:///`"). Refer to Chapter 7 for more details on resources. In this URL, not just resources, but also the HTTP URL indicating the file on the Web server and the URL indicating the position on the ScratchPad can be specified.

Example: Display an image file

```
package uidemo;
import java.util.*;
import com.nttdocomo.ui.*;
import com.nttdocomo.io.ConnectionException;

public class VisualPresenterDemo extends UIDemoPanel {
    int           px, py;         // The display position of the current image
                                  // (in VisualPresenter)
    VisualPresenter theVP;        // Presenter
    Button         startButton;   // Playback button for GIF animation
    Button         stopButton;    // Stop button for GIF animation
    ListBox        repeatChoice;  // Checkbox for instructing repeat playback
    Ticker         ticker;        // Ticker tape for describing the key map.

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

```
/**
 * VisualPresenterDemo constructor
 * Acquire necessary image data and pass it to VisualPresenter.
 */
VisualPresenterDemo() {

    // Acquire image data.
    MediaImage mi =
        MediaManager.getImage("resource:///uidemo/img/singleloop.gif");

    // Image data cannot be used until the use() method is called.
    try {
        mi.use();
    } catch (ConnectionException ce) {
        // Process an input/output exception at this point.
    } catch (UIException uie) {
        // Process a UI exception at this point.
    }

    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);

    // Configure the media listener.
    theVP.setMediaListener(new MediaListenerClass());

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


class KeyListenerClass implements KeyListener {

    /**
     * Ignore key release events.
     */
    public void keyReleased(Panel p, int key) {}

    /**
     * Adjust the image display position by a key press event.
     *
     * @param p      Panel where an event occurred.
     * @param key    Value of the pressed key
     */
    public void keyPressed(Panel p, int param1) {

        if (VisualPresenterDemo.this == p) {
```

```
            if (param1 == Display.KEY_8) {            // Image position (bottom)
                py--;
            } else if (param1 == Display.KEY_2) {     // Image position (top)
                py++;
            } else if (param1 == Display.KEY_4) {     // Image position (left)
                px++;
            } else if (param1 == Display.KEY_6) {     // Image position (right)
                px--;
            }

            // Set the image display position by calling
            // the setAttribute() method of VisualPresenter.
            theVP.setAttribute(VisualPresenter.IMAGE_XPOS, px);
            theVP.setAttribute(VisualPresenter.IMAGE_YPOS, py);
        }
    }
}

class ListenerClass implements ComponentListener {

    /**
     * Control the start and stop of playback by a component event.
     *
              * @param c      Component where event occurred.
              * @param type Event type
              * @param param  Event parameter
     */
    public void componentAction(Component c, int type, int param) {
        if (c == startButton) {               // When the playback button is pressed,
                                              // playback starts.
            theVP.play();
        } else if (c == stopButton) {         // When the stop button is pressed,
                                              // playback stops.
            theVP.stop();
        }
    }
}

class MediaListenerClass implements MediaListener {

    /**
     * Media listener for receiving an event from VisualPresenter
     * and playing again as required. The loop playback capability is
     * realized at this point.
     */
    public void mediaAction(MediaPresenter source, int type, int param) {

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

            // An image playback complete event is processed at this point. In the
            // Panel title, display the completion of playback and repeat playback
            // as required. This processing is used only for non-loop playback GIF
            // animation.
            case VisualPresenter.VISUAL_COMPLETE :
                setTitle("Complete");

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

            // An image playback start event is processed at this point.
            // In the Panel title, display that the playback started.
```

```
                case VisualPresenter.VISUAL_PLAYING :
                    setTitle("Playing");
                    break;

                // An image playback stop event is processed at this point.
                // In the Panel title, display that the playback stopped.
                case VisualPresenter.VISUAL_STOPPED :
                    setTitle("Stopped");
                    break;
                }
            }
        }
    }

    /**
     * MediaImage should be discarded (unuse() and dispose()) if it is no longer necessary.
     * By discarding it, the resources are released. At this point, do not discard MediaImage
     * because it may be reused.
     */
    public void leave() {
    }
}
```

Example: Play back a sound file

```
package uidemo;

import com.nttdocomo.ui.*;
import com.nttdocomo.io.ConnectionException;

public class AudioPresenterDemo extends UIDemoPanel {
       ListBox           loopChoice;     // Loop playback flag
       Button            playButton;     // Sound playback button
       Button            stopButton;     // Sound stop button
       Label             statusLabel;    // Status display label
       AudioPresenter    theAP;          AudioPresenter
       MediaSound        theMediaSound;  // MediaSound

       /**
       * AudioPresenterDemo constructor
       * Create two buttons for playing/stopping a sound file and a checkbox for
       * selecting loop playback.
       */
       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");

              // Sound data cannot be used until the use() method is called.
              try {
                     theMediaSound.use();
              } catch (ConnectionException ce) {
                     // Process an input/output exception at this point.
              } catch (UIException uie) {
                     // Process a UI exception at this point.
              }

              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 {
        /**
        * Listener class for receiving an event of the button for controlling
        * playback/stop of sound data.
        */
        public void componentAction(Component c, int type, int param) {
                if (c == playButton) {
                        theAP.play();
                } else if (c == stopButton) {
                        theAP.stop();
                }
        }
}

class MediaListenerClass implements MediaListener {
        /**
        * Media listener for receiving an event from AudioPresenter
        * and repeat playback as required. The loop playback capability
        * is realized at this point.
        */
        public void mediaAction(MediaPresenter source,int type,int param) {
                if (source == theAP) {
                        switch(type) {
                        // A sound playback complete event is processed at this point.
                        // In the status display label, display the completion of
                        // playback and repeat playback as required.
                                case AudioPresenter.AUDIO_COMPLETE:
                                        statusLabel.setText("complete");
                                       if (loopChoice.isIndexSelected(0))
                                                ((AudioPresenter)source).play();
                                        break;

                        // A sound playback start event is processed at this point.
                        // In the status display label, display that the playback
                started.
                                 case AudioPresenter.AUDIO_PLAYING:
                                        statusLabel.setText("playing ");
                                        break;

                        // A sound playback stop event is processed at this point.
                        // In the status display label, display that the playback
                        // stopped.
                                case AudioPresenter.AUDIO_STOPPED:
                                        statusLabel.setText("stopped ");
                                        break;
                        }
                }
        }
}

/**
* When this screen is hidden, call the AudioPresenter.stop() method
* in order to stop the sound playback. If MediaSound is no longer used,
* the MediaSound. unuse() and MediaSound.dispose() methods should be called
* (various resources are released). At this point,
```

```
        * these methods are not called because MediaSound may be reused.
        */
        public void leave() {
               theAP.stop();
        }
}
```

[DoJa-2.0]

- With DoJa-1.0 Profile, the behavior which occurs when the `stop()` method is called with respect to an AudioPresenter without a media data setting will depend on the manufacturer in question. In contrast, if this should occur with DoJa-2.0 Profile or later, an exception (i.e., `UIException`) will be generated. Please use the method `MediaPresenter.getMediaResource()` to determine whether or not media data has been set for a media presenter.
- With DoJa-2.0 or later profile, the following specifications have been added for the playback of animated GIFs using `VisualPresenter`.
  - The minimum number of frames to allow playback is eight.
  - When replaying a looped animated GIF, a replay completion event is generated each time the loop is completed.
  - The maximum number of repetitions for a looped animated GIF is sixteen. Even in cases where looping with a larger number of repetitions is specified, replay will stop when the sixteenth loop has been completed.

[DoJa-3.0]

With DoJa-3.0 or later profile, data-file byte images can be specified for retrieving media data in addition to URLs. The byte image can be given as either a byte array or in `InputStream` format.

Also in DoJa-3.0 or later profile, attributes have been added to adjust the below items related to `AudioPresenter`.

- Tempo (`AudioPresenter.CHANGE_TEMPO`)
- Volume (`AudioPresenter.SET_VOLUME`)
- Key (`AudioPresenter.TRANSPOSE_KEY`)

Similar functionality is contained in DoJa-2.0 Profile i-αppli Optional API (`AudioPresenter2` class), but please note that, in terms of programming, the two are not compatible.

[DoJa-4.0]

The following functions were added to `MediaResource` which is a common interface of media data in the DoJa-4.0 Profile and later.

- `isRedistributable()`: Obtains if this media data is set to be redistributable. Whether it can be set to be redistributable and its setting method depends on the types of media data and mobile phone platforms (PDC or FOMA). Data stored in the native storage area as non-redistributable cannot be retrieved from the mobile phone by using mail attachment or an infrared function.
- `setRedistributable()`: Sets this media data as redistributable or non-redistributable.
- `getProperty()`: Obtains a property value of media data. Types of supported properties are determined by the type of media data.The following properties are supported in the DoJa 4.0 Profile:

  - `MediaSound.AUDIO_3D_RESOURCES`

    Obtains the number of 3D sound control resources required to use 3D sound control information (location information) when this media sound data is embedded with it.See Chapter 13 for information about the 3D sound control and the 3D sound control resources.

  - `MediaImage.MP4_AUDIOTRACK`

    Obtains the number of audio tracks contained in the data (a number string representing a number) when this media image data is iMotion format data.

  - `MediaImage.MP4_TEXTTRACK`

Obtains the number of text tracks contained in the data (a number string representing a number) when this media image data is iMotion format data.

- `MediaImage.MP4_VIDEOTRACK`

  Obtains the number of video tracks contained in the data (a number string representing a number) when this media image data is iMotion format data.

[DoJa-4.1]

In DoJa-4.1 Profile and later, some manufacturers support acquisition of the `MediaImage` object by specifying the data of Flash content to an argument of the `MediaManager.getImage()` method and calling it. Note that the `MediaImage` object that is generated from the Flash content cannot be played by a presenter.The `MediaImage` object generated from the Flash content can be used only for the purpose of saving the Flash content in the native image storage area via the `ImageStore.addEntry()` method.

[DoJa-5.0]

The static method `use()` was added to the `MediaManager` class in DoJa-5.0 Profile and later to turn multiple `MediaImage` or `MediaSound` objects into a usable state together in a single batch. In this use() method, you can specify whether or not you only want to use these media data objects once or not. For more information about this "one time only" usage specification for media data objects, see Section 4.4.7.

The following functions were added to the `AudioPresenter` class in DoJa-5.0 Profile. However, these functions are optional and may or may not be supported depending on the manufacturer.

- Looped play specification and looped play event (the `LOOP_COUNT` attribute and `AUDIO_LOOPED` event)
- Playback of iMotion audio tracks (the `getAudioTrackPresenter()` method)
- Retrieval of the playback time for an entire song (the `getTotalTime()` method)

4.4.3 Multiple Simultaneous Playback with AudioPresenter

From DoJa-2.0 Profile or later, an application program can retrieve multiple instances of `AudioPresenter` and instruct these to playback simultaneously. The behavior of this functionality differs slightly among the profiles of DoJa. Below, the behavior of each profile shall be explained in outline.

[DoJa-2.0]

In DoJa-2.0 Profile, the minimum number for `AudioPresenter` simultaneous playback guaranteed across all phone models is one. In other words, multiple `AudioPresenter` simultaneous playback cannot be performed depending on the manufacturer.

When the application program instructs simultaneous playback from more presenters than is possible, a decision is made based on the value of the priority ranking attribute assigned to the presenter (`AudioPresenter.PRIORITY`). In mobile phones which can only play back one sound at a time, if a high-priority presenter is directed to play while a low-priority presenter is playing, the high-priority presenter will play, cutting off the low-priority presenter. When the high priority presenter has finished playing, the low priority presenter playback will be resumed.

On mobile phones that do not support the priority attribute, presenters which begin playing later are handled as if they have higher priority than presenters which are in the process of playing. Further, behavior when playback instructions exceed the mobile phone's capabilities (number of simultaneous sounds, number of usable channels), differs depending on the manufacturer.

[DoJa-3.0]

In DoJa-3.0 Profile, the minimum number for `AudioPresenter` simultaneous playback guaranteed across all phone models is two.

When the application program instructs simultaneous playback from more presenters than is possible, a decision is made based on the value of the priority ranking attribute assigned to the presenter (`AudioPresenter.PRIORITY`). In mobile phones which can play back two sounds at a time, if a high-priority presenter is directed to play while a medium-priority and a low-priority presenter are playing, the low-priority presenter is stopped, and playback of the high-priority presenter begins (the medium-priority presenter will continue playback, uninterrupted).If multiple presenters exist with priorities such that any of them could be stopped, which presenter is stopped varies depending on the manufacturer.

In mobile phones that do not support the priority attribute, if simultaneous playback from more presenters than is possible is instructed, presenters which begin playing later are handled as if they have higher priority than presenters which are in the process of playing.

If instructions are within the limits of the mobile-phone's capabilities (number of simultaneous sounds, number of usable channels), all simultaneous sound playback will be heard, regardless of presenter priority. If instructions are performed that exceed the mobile-phone's capabilities, it is possible not all sounds will be heard, even if the instructions are within the limits of the number of presenters for which simultaneous playback is possible.

Further, with regards to `AudioPresenter` simultaneous playback, the concept of ports has been introduced in DoJa-3.0 Profile. Ports are virtual playback machines, enough of which are prepared for several simultaneous playback `AudioPresenters`. Which ports individual `AudioPresenters` are assigned can be declared explicitly with an optional argument in the `AudioPresenter.getAudioPresenter()` method. It is not possible to simultaneously play an `AudioPresenter` with a port number assignment along with an `AudioPresenter` without a port assignment.

[DoJa-4.0]

The number of simultaneous replayable `AudioPresenter` commonly guaranteed in all models is extended to at least four in the DoJa-4.0 Profile in the case where a port is specified for `AudioPresenter`. Others follow behaviors in the DoJa-3.0 Profile.

Notes:

- FOMA mobile phones support the MFi format and the SMF format as sound data, but it depends on manufacturers whether both of them can be simultaneously played back with mixing. In the case of simultaneously playing back multiple sounds, make sure to integrate the sound data formats to either one of them.

4.4.4 Audio-Synchronization Events

With DoJa-3.0 or later profile, `AudioPresenter` supports the triggering of synchronization events in sound playback. The term “synchronization event” is used to refer to media events which are notified to MediaListener in synchrony with the playback of a sound.The method `AudioPresenter.setAttribute()` is used to set the synchronization-event generation attribute (i.e., `AudioPresenter.SYNC_MOD`E) as either ON (i.e., `AudioPresenter.ATTR_SYNC_ON`) or Off (i.e., `AudioPresenter.ATTR_SYNC_OFF`), thus allowing the generation or non-generation of these events to be controlled.

When using the synchronization event function, any one of the channels comprising the sound data is specified as the event generation source. Subsequently, synchronization events will be notified to MediaListener linked to the specified channel and specified note messages.

Note that the channel specified as the event generation source will also be subject to sound generation. If this is not necessary, however, specify this channel as being muted during the creation of sound data.

[DoJa-2.0] / [DoJa-3.0]

Sound playback synchronization events were not supported in DoJa-1.0 Profile, and were relegated to the i-αppli Optional APIS in DoJa-2.0 Profile. This functionality has been adopted into the i-αppli standard API from DoJa-3.0 Profile or later.

4.4.5 Play back of iMotion with VisualPresenter on FOMA mobile phones

The only movie media that can be played back with `VisualPresenter` on PDC mobile phones is GIF. On the other hand, FOMA mobile phones can play back iMotion format data in addition to animated GIFs.The iMotion format data can be played back by generating a `MediaImage` object in the same manner as animated GIF and by using `VisualPresenter`.

You cannot use the `getImage()` method to retrieve an Image object from a `MediaImage` object created from iMotion format data, nor can you use the `ImageStore` class to save or re-acquire that `MediaImage` object. (To store or acquire iMotion format data, use the i-appli API's `MovieStore` class instead.)

The following extensions are done to handle the iMotion in the class VisualPresenter and the class MediaImage on the i-αppli Runtime Environment of the DoJa-3.5 Profile and later.

Specification of Player Modes

The following player modes are available to playback iMotion data: native player mode in which an i-αppli suspends by a playback indication (`VisualPresenter.play()`) and a native iMotion player is executed, and inline player mode in which a component `VisualPresenter` placed on a `Panel` plays back an iMotion without suspending an i-αppli. Also, in DoJa-5.0 Profile and later, a full-screen player mode was added so that the native player could utilize bigger display sizes for magnified playback. Full-screen player mode is a variation of native player mode, and depending on the playback instruction the i-appli be temporarily suspended. Furthermore, the player display (visual quality) when full-screen player mode is used from an i-appli is based on the player display when full-screen player mode is used by the native player.

The native player mode is the one that can be used on any model. Inline player mode and full-screen player mode are optional, and it depends on manufacturers whether these modes cans be used. Furthermore, it also depends on the manufacturer whether iMotion data which contains text tracks can be played in either inline player mode or full-screen player mode.

The player mode can be selected by setting the `PLAYER_MODE` attribute via the `setAttribute()` method in the `VisualPresenter` class.

Specification of Sound Playback Modes

iMotion is a type of media data that can include sound. For the playback of iMotion in the inline player mode, it can be selected whether to play back the sound data in iMotion. If it is set that sound data in iMotion is not to be played back, `VisualPresenter` does not use resources related to audio playback, and an i-αppli can simultaneously play back a media sound with `AudioPresenter` separately defined from `VisualPresenter`.

The sound playback mode can be selected by setting an attribute `AUDIO_MODE` by the method `setAttribute()` in the class `VisualPresenter`.

Handling Progressive Downloads

There are two kinds of download methods in iMotion: a normal download method which allows playback after completing the download of all data to a mobile phone, and a progressive download method which starts the playback after a certain amount of data is downloaded and performs download and playback of the data in a parallel manner afterwards. The `getStreamingImage()` method in the `MediaManager` class is added to the DoJa-3.5 Profile and later to handle progressive downloads at the API level. A `MediaImage` object generated from an iMotion content supporting the progressive download carries out, by using that method, download and playback of data in a parallel manner when the class `VisualPresenter` receives an indication to playback.

Note that, depending on the manufacturer, progressive downloading in i-applis may not be supported. Even if you use the `getStreamingImage()` method on such models the download will not be a progressive download, and the content will be played once it has been downloaded in its entirety.

The maximum size of iMotion data that can be played from an iAppli when a HTTP scheme iMotion URL is specified to the `MediaManager.getImage()` method is 150 KB. (This is due to the HTTP(S) communications size restriction as covered in Section 5.1 of this document.) For larger iMotion data that has been downloaded in parts and put back together in the ScratchPad or in runtime memory, the maximum playable size increases to 500 KB. (Depending on the manufacturer, even larger iMotion data may be supported.)

In addition, a standby application in an inactive mode cannot playback an iMotion in the native player mode.

[DoJa-5.0]

The regulations for the maximum size of iMotion data that can be played back from an i-appli (the common size restrictions imposed on all models) differs between different DoJa Profile versions. DoJa-5.0 Profile allows iMotion data of up to 500 KB to be played, but older versions are restricted to a maximum of 300 KB.

Notes:

- iMotion data which is configured as full music content cannot be played via the `VisualPresenter` class.

4.4.6 Output Control of Multimedia Data to External Memory

Multimedia data that can be manipulated by i-applis (objects which implement `MediaResource`-inherited interfaces) can be stored in the mobile phone's native data storage area (hereinafter referred to as internal memory) by using the application linking functionality (i.e., the `ImageStore` class or the `MovieStore` class) as explained in Chapter 11 of this document. However, on FOMA 902i or newer mobile phones, internal memory content can be transferred to external memory by utilizing various content protection technologies. Furthermore, that data can even be transferred to the internal memory of another mobile phone.

In DoJa-4.1 Profile (902iS) and later, a system for i-applis to control the protection level of content when multimedia data stored in inernal memory is output to external memory via user operation was added.

The following section will explain about how to deal with content protection levels for multimedia data stored in internal memory from an i-appli.

Content not set as non-redistributable (re-distribution is allowed)

Content not set as non-redistributable is not eligible for the external memory's copyright protection system. Therefore, if an i-appli saves multimedia data which is not set as non-redestributable and that data is transferred to external memory via user operations, that content is stored to the external memory in non-encrypted, plain text form. As such, data transferred to external memory in this way can be viewed with software on a PC or other such device.

Content set as non-redistributable (re-distribution is not allowed)

Content set as non-redistributable is eligible for the external memory's copyright protection system and therefore will be encrypted when output to external memory. Normally, if data which was registered as non-redistributable is moved to the external memory, that content cannot be decoded unless it is on the same device as the data was saved on and mobile phones where the same UIM is inserted as the data was saved. Remember, before saving data to the internal memory in an i-appli use the `MediaResource.setProperty()` method and specify:

The first argument (property name): `"X-Dcm-Move"`

The second argument (property value): `"0001"`

By doing this, that data will be able to be decoded by any mobile phone as long as it has the same UIM as the one used when that data was originally saved.

This property information is retained even in internal memory, and will still be retained as valid even when an i-appli uses the `ImageStore` class, etc. to load that data back from the internal memory. (However, when the `MediaManager` class is used to make objects from such multimedia data, these properties will revert back to unspecified.)

4.4.7 Memory Management When Using Media Data

When an i-appli uses media data such as images or sounds, a relatively large amount of application runtime memory (heap memory) is used. In most implementations when you are dealing with media data, you will use memory for storing original data in each format such as GIF or JPEG, and memory to store that data when it has been converted to an internal format.

in DoJa-5.0 Profile, the memory management conducted by the i-appli Runtime Environment when media data is used follows the models listed below, and the API was extended in order to provide some functionality to control that management from an i-appli.

Original Data Area:

This memory is used to store original data such as GIF or JPEG. As a rule, this memory is allocated when a media data object is created, and is maintained until the `dispose()` method is called on that object.

Internal Data Area:

This memory is used to store original data which has been converted into a format which can be handled by the system internally. For example, for an image this would be pixel data which has been extracted from a GIF or JPEG. As a rule, this memory is allocated when the `use()` method is called on a media data object, and is maintained until the `unuse()` method is called on that object.

For example, for media data which you are sure you will not call `use()` or `unuse()` on multiple times from the time the application is launched until it is closed (i.e., you will not revert between usable and unusable states for that data multiple times), you can destroy the original data when you make the media data usable which allows you to explicitly free up that much memory. (This is called a "one-time only" usage specification.)

Furthermore, if you are dealing with multiple pieces of media data which you only want to use for a short amount of time, you can specify to use a single internal data area for multiple media data objects to conserve space (i.e., re-using the same internal data area).

These functions are provided as extensions to the `use()` method defined for each media interface.

Notes:

Depending on the implementation of the i-appli Runtime Environment, the memory management models may not be applied exactly as described in this section. In such implementations, you may not notice much of an effect, if any at all, when controlling memory usage from an i-appli.

Depending on the type of media data you are handling, you may not be able to re-use internal memory data areas for multiple media data objects. Media data for which internal data areas can be re-used are: static images (not including animated GIFs) and sound (MFi and SMF).

If the use() method is called more than once on media data which has been specified to be used "one-time only" an exception will occur. Furthermore, note that media data objects for which this has been specified do not store any original data. As such, you cannot save that data to the native data area via `ImageStore` or any other method.

When re-using internal data areas, the memory that has been allocated for that area cannot be expanded upon when re-using it. Ensure that the amount of memory allocated when the internal data area is first used will be large enough to store any data you plan to store there later. You will need to think about the size of internal data areas based on the type of media data you plan on storing there. For more details, see the sections about the `use()` method for the various media interfaces in the API Reference.

`Image` objects retrieved from a `MediaImage` object (the `getImage()` method) must reference to the internal data area held internally by that `MediaImage` object. Therefore, when using an `Image` object retrieved in this way, you must ensure that the internal data area held by its corresponding `MediaImage` is not destroyed.

[DoJa-5.0] / [DoJa-5.1]

In DoJa-5.0 Profile, if a stream is specified when creating new media data (i.e., when "stream" is specified as an argument to the `MediaManager` class' media data retrieval method), no original data area is used. In this case, when the `use()` method is called on such a media data object a new block of data is read from the stream and the internal data is created.

Conversely, in DoJa-5.1 Profile this has been changed so that an original data area is used if a stream is specified when creating new media data.In DoJa-5.1 Profile, if the `use()` or `unuse()` methods are repeatedly called on a media data object for which "stream" has been specified, the internal data will always be created from the original data loaded on the first call to those methods.

4.5 Image Processing

Functions related to image processing (encoding or processing of images) were strengthened through upgrades of profiles in the i-αppli Runtime Environment. In this section, these features shall be explained in summary.

4.5.1 Image Encoder

The image encoder added in the DoJa-3.0 Profile provides functionality for capturing the content rendered in `Image` and `Canvas`, and encoding it to a byte image of a specified format. Using this feature, the contents of the image file can be processed on `Image` or `Canvas`, then reconverted into an image file and stored in ScratchPad or sent to a server.

The `ImageEncoder` class is used to encode images. By specifying a graphics format name as an argument when calling the static `getEncoder()` method, the encoder object corresponding to that graphics format can be retrieved. Handling of the actual encoding is performed by using the encoder object's `encode()` method.

When the `encode()` method is called to perform image encoding, an `EncodedImage` object is returned as the result. The `EncodedImage` is a class encapsulating the encoded image data, and from this object the image's byte image can be retrieved.

The example below shows image encoding handling.

Example: Using the image encoder

```
import com.nttdocomo.ui.*;
class MainCanvas extends Canvas {

      public void paint(Graphics g) {
            // Render something in this Canvas.
            ...
      }

      public void processEvent(int type, int param) {
            // Capture the rendered contents of Canvas with a softkey event.
            if (type==Display.KEY_RELEASED_EVENT && param==Display.KEY_SOFT1) {
                  // The image format should be JPEG.
                  ImageEncoder ie = ImageEncoder.getEncoder("JPEG");
                  // Capture an area with width of 50 dots and height of 50 dots from
                  // the coordinates (0,0) of this Canvas. The captured result is returned
                  // as the EncodedImage object.
                  EncodedImage ei = ie.encode(this, 0, 0, 50, 50);
                  // Acquire a stream for extracting an image byte image
                  // (in this case, JPEG).
                  InputStream is = ei.getInputStream();
                  ...
            }
      }

      ...
}
```

Notes:

- The commonly supported graphics format for every model of phone is JPEG (JFIF Baseline encoding).To retrieve the JPEG image encoder, specify "JPEG" as the argument for the `ImageEncoder.getEncoder()` method. Also, the size of image, both height and width, that can be encoded is dictated by the size of the mobile phone's rendering area. Note that whether or not an image exceeding this size can be encoded varies depending on the phone's manufacturer.
- When capturing the `Canvas`, that `Canvas` object must be set to the current frame. Also, if the IME is being started in that `Canvas` object, capturing cannot be performed.
- When capturing an `Image`, that `Image` object must be an object that can be called by the `getGraphics()` method. In other words, it must be an `Image` object that was created with the `Image.createImage()` method.

- In some cases, capture is not supported for the `Canvas` and `Image` inheritance classes that exist in the i-αppli Optional and Extension APIs.

4.5.2 Pixel Operations

Pixel operations are functions for providing direct access from the application program to each pixel (picture element) arranged in the graphics context. The application program can use the following features of the `Graphics` class as pixel operations.

- Retrieval of color information for specific pixel positions in the graphics context (`getPixel()` family methods)
- Rendering dots to specific pixel positions (`setPixel()` family methods)

Pixel operation functionality was defined as an i-αppli Optional API in DoJa-2.0 Profile. In the DoJa-3.0 Profile, those functions were partly stregthened and moved into the i-αppli standard API.

Notes:

- In methods for retrieving color information (`getPixel()` family methods), if out of range coordinates, or a `Graphics` object in which the `Canvas` is not set to the current frame is used, color information indicating black will be returned. This behavior has changed from the pixel operation functionality in DoJa-2.0 Profile's i-αppli optional API.
- The color retrieval methods retrieve information from the back buffer and return it to the application program when double buffering is enabled. For this reason, the colors displayed on the screen may not agree with the colors retrieved by this method.

4.5.3 Image Rotation/Flipping/Enlargement/Reduction

The `Graphics.setFlipMode()` method is called to rotate or flip images. If this method is called, specifying a rotation or flip method with respect to the `Graphics` object, a rotated/flipped image will be rendered with subsequent calls to the rendering method (`drawImage()` method or `drawScaledImage()` method).

The rotation/flip methods that can be specified are as follows.

- `Graphics.FLIP_HORIZONTAL` : Image will be horizontally flipped.
- `Graphics.FLIP_NONE` : No flipping.
- `Graphics.FLIP_ROTATE` : Image will be flipped lengthwise and breadthwise (180 degree rotation).
- `Graphics.FLIP_ROTATE_LEFT` : Image will be rotated 90 degrees left.
- `Graphics.FLIP_ROTATE_RIGHT`: Image will be rotated 90 degrees right.
- `Graphics.FLIP_VERTICAL` : Image will be flipped vertically.
- `Graphics.FLIP_ROTATE_RIGHT_HORIZONTAL`: Image will be rotated 90 degrees right and flipped horizontally.
- `Graphics.FLIP_ROTATE_RIGHT_VERTICAL`: Image will be rotated 90 degrees left and flipped vertically.

[DoJa-4.1]

`FLIP_ROTATE_RIGHT_HORIZONTAL` and `FLIP_ROTATE_RIGHT_VERTICAL` were added with DoJa-4.1 Profile.

The `Graphics.drawScaledImage()` method is called to enlarge or shrink an image. The rendering positions for the image and graphics context are set in this method's arguments, in addition to the below:

- Width and height of the rectangular area of the graphics context to which the image will be rendered.
- Upper left coordinate, and width and height of the rectangular area that is to be rendered from the rendering source image

The contents of the rendering source's rectangle will be enlarged or shrunken to exactly fit into the rendering target rectangle.

The `drawImage()` method which can specify with representing rotation or scaling of images in any affine transformations is added to a class `Graphics` in the DoJa-3.5 Profile and later. The 3 x 3 affine transformation matrix shown below is applied to each of the image coordinates, and the image is drawn to the calculated coordinate group. In the equation below, x and y represent the coordinates on the original image, x' and y' represent the coordinates after the conversion (coordinates on the target image), and `m00` to `m12` represent the affine transformation matrix that can be specified by the application program.

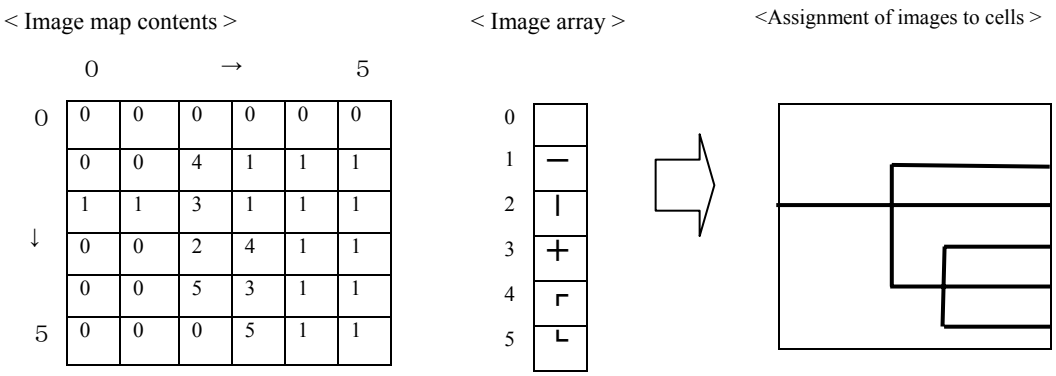
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \frac{1}{4096} \begin{pmatrix} m00 & m01 & m02 \\ m10 & m11 & m12 \\ 0 & 0 & 4096 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

The method `Graphics.setFlipMode()` and the method `Graphics.drawScaledImage()` are defined in the i-αppli option API as of the DoJa-2.0 Profile. In the DoJa-3.0 Profile, those functions were partially stregthened and moved into the i-αppli standard API.

4.5.4 Image Maps

Image maps (`com.nttdocomo.opt.ui.ImageMap` class) provide the functionality to arrange small images in rows and columns so that a larger image may be created.

The term "cell" is used to describe the unit for application of images to image maps (i.e., one item in the image map). ImageMap indicates the grouping of images which comprise the map (i.e., array) and map data which determines the images to be assigned to each individual cell. The following diagram illustrates a situation where images are assigned to an image map comprising 6 x 6 cells.



The values in the matrix refer to image array elements.

This type of image map is ideal for situations where the minimum amount of data is to be used in the creation of a larger image which can be realized by combining a number of smaller images.

Image maps are rendered on graphics contexts using the `Graphics.drawImageMap()` method. Although Image Map can be applied with respect to the entire rendered image, it is also possible to setup a rectangular area on the image map called a "window" so that rendering may be performed only for the area

inside this window. It is possible to freely modify the position of windows within the boundaries of the image map, thus making it possible to easily create applications where larger images can be seen to scroll within a display.

Image maps were defined in the i-αppli optional API from the DoJa-2.0 Profile. It was later moved to the i-αppli standard API in the DoJa-3.5 Profile, and the following points are strengthened in the DoJa-4.0 Profile.

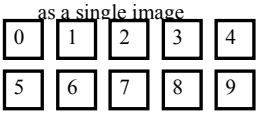
- **Extension of the number (kinds) of images that make up an image map**

  Since a byte array was used for an index array to assign an image to a cell in previous profiles, the number of images consisting of an Image map was restricted to 128 at most. This restriction is no longer available because a constructor method using an int array for the index array was added in the DoJa-4.0 Profile and later. By adopting an int array for the index array, a constructor method specifying the index array with a byte array became deprecated.

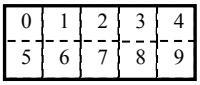
- **Setting of image map images in a connected format**

  In an image map in previous profiles, each configuration image was supposed to be independent image data, and it was necessary to prepare as many images files as the number of configuration images. In addition to it, the configuration images became able to consist of a large image which each configuration image is combined in the DoJa-4.0 Profile. The amount of data might be able to be minimized by combining configuration images into an image file.

Example of preparing 10 configuration images as a single image



Example of preparing 10 individual configuration images



4.5.5 Sprites

The i-appli Runtime Environment provides a simple sprite display function as one of its image processing functions. The sprite functionality consists of the `com.nttdocomo.ui.Sprite` class and the `com.nttdocomo.ui.SpriteSet` class.

The class Sprite represents each sprite moving on the screen and has an image object and a display position as attributes. In addition, when an image which has the class `Sprite` is actually displayed on the screen, the following operations are available:

- Setting of visibility and invisibility
- Setting of reversal and rotation

In many situations, an application will be displaying multiple sprites on a single screen. Therefore, the `SpriteSet` class is also included to manage multiple sprites as a group. An application will simply register all the Sprites they want to use simultaneously to a SpriteSet. To display these Sprites on the screen, you would then call the `Graphics.drawSpriteSet()` method and specify the SpriteSet you want to draw.It is not possible to carry out independent rendering of `Sprite` objects.

Using a SpriteSet, you can perform collision detection between the Sprites stored in a particular SpriteSet. Collision detection is performed based on the locations of each sprite or the rectangular size of the images registered as Sprites.

Sprite was defined in the i-αppli option APIs from the DoJa-2.0 Profile. It was later moved to the i-αppli standard API in the DoJa-3.5 Profile. However, the raster operation that existed in the i-αppli option API (`Sprite.setRenderMode()` : calculation of the pixel-color of Sprite and the pixel-color of the background) is no longer defined in the Sprite functions in the i-αppli standard API.

4.5.6 Images with Palettes

A paletted image (the `com.nttdocomo.ui.PalettedImage` and the `com.nttdocomo.ui.Palette` class) is an image to which a color palette can be specified from an application program.

A paletted image can be obtained by specifying the GIF data as a byte array as a parameter to the `PalettedImage.createPalettedImage()` method. (From DoJa-5.0 Profile and later, you can also use 1, 4, or 8-bit Windows BMP images in addition to GIF data).

Referring and setting of the palette can be performed by using methods `getPalette()` and `setPalette()` from a paletted image. The color information of a palette is specified with a model-specific integer representing color (a returned value of a method `Graphics.getColorOfRGB()`).

The class `PalettedImage` inherits the `Image` class. However, the method `getGraphics()` defined in the `Image` class cannot be called from a `PalettedImage` object. If an application calls this method, an exception occurs.

Paletted image was defined in the i-αppli option API from the DoJa-2.0 Profile. It was later moved to the i-αppli standard API in the DoJa-3.5 Profile.

[DoJa-5.0]

In DoJa-5.0 Profile, the following methods were added to the `PalletedImage` class.

- A variation of the `createPalettedImage()` method which allows specification of a width and heigh size only to create a blank image
- The `changeData()` method to enable the ability to switch out the image data of an already created `PalettedImage` object with different image data

Note that you can also specify a color for transparency in the `PalettedImage` class as well, however this specification is not handled the same way as for the class' parent `Image` class (see the next section for details). Instead, the `setTransparentIndex()` method is used to specify a palette index value for transparency.

4.5.7 Setting a Transparent Color and Specifying Semi-transparent Rendering for Images

In DoJa-5.0 Profile and later, specification of a transparency color for images and the specification of semi-transparent rendering is supported as part of the i-appli Standard API. These functions are provided as API added to the `Image` class.

(1) Specifying a Transparent Color for an Image

This functionality is used to specify a certain color so that the application program will treat that color as transparent when rendering the image. When the image is drawn to the screen, the color specified as transparent will not be rendered when it appears in that particular image.

To draw an image with a specified color set as transparent, first call the `setTransparentColor()` method for the `Image` object and specify the color you want to be transparent. Next, specify true as a parameter to the set`TransparentEnabled()` method to enable transparency for that image. The color data specified as a parmeter to the `setTransparentColor()` method must be a maker-specific color value as returned from the `Graphics.getColorOfName()` or `Graphics.getColorOfRGB()` methods.

Note that the `setTransparentEnabled()` method must be called after every time the `setTransparentColor()` method is used to change the transparent color (i.e., any time after the transparent color has been changed).

You can check to see what color is currently set to be transparent by using the `getTransparentColor()` method. You can also disable transparency for an image by specifying false as a parameter to the `setTransparentEnabled()` method.

Notes:

- Even if an alpha value is included in the color value passed to the `setTransparentColor()` method, the alpha value will be ignored. The method will always handle the alpha value as if it were specified as 255.
- For `Image` objects which were created from transparent GIF data, the transparent color information taken from the original GIF data will remain valid until specifying true for the `setTransparentEnabled()` method on that Image object. The transparent color information obtained from the original GIF data will be lost if the `setTransparentEnabled()` method is called with true specified as a parameter.
- In the i-appli Optional API the `com.nttdocomo.opt.ui.TransparentImage` class is defined as an API to handle images with transparency in the same way as the functionality described in this section.In DoJa-5.0 Profile and later, use the functionality discussed in this section to handle transparency in images, not the `TransparentImage` class.Using the `TransparentImage` class in DoJa-5.0 Profile and later is not recommended.

(2) Specifying Semi-Transparent Rendering for an Image

This functionality is used when rendering an image to specify an alpha value which affects how that image is rendered. Using this function you can have the API perform transparency calculations on the render source (the image) and render target pixels when drawing an image to the screen.

To perform semi-transparent rendering, simply use the `setAlpha()` method on the `Image` object you want to render and set an alpha value for that image. You can specify any number between 0 and 255 as a parameter to this method. An alpha value of 255 will cause the image to be completely opaque, while a value of 0 will treat the image as completely transparent.

Chapter 5
Controlling Communications

A networking API is included with the i-appli Runtime Environment. This API consists of a series of classes used by the developer when communicating with components on a Web server. The objective of these classes is to make the development of Java client applications as simple as possible (see Figure 5 in Chapter 2, “Generic Connection Class Hierarchy”). In the i-αppli Runtime Environment, application programs can use the HTTP(S) protocol to communicate with web browsers. Standard J2SE libraries provide a rich set of functionality for handing input/output and communications with storage devices. The i-αppli Runtime Environment provides basic functionality for connecting the mobile phone to a Web server based on the Generic Connection framework of the CLDC specification (subsection 2.2.2). The following sections describe how to use the HTTP(S) Connection interface to communicate with the server.

The i-αppli Runtime Environment Networking APIs differ significantly from the J2SE `java.net.HttpURLConnection` in the following ways:

- There is no concept of content handler factories for plugging in handlers capable of recognizing message content type and parsing content into object form.
- We are targeting memory footprint reduction by weeding out the number of supported content types. Ordinary applications only use a limited number of content types, and so we do not employ a general framework supporting multiple content types.
- There is no generic `URLConnection` (`Java.net.URLConnection` class). To keep the number of classes to a minimum, the generic `URLConnection` API is merged with the `HttpConnection` interface. This reduces memory footprint, and simplifies the API developers are to use.
- There is no `URLStreamHandlerFactory` or `URLStreamHandler` API. The i-αppli Runtime Environment limits the number of available network protocols to HTTP and HTTPS, and so a general framework supporting a large number of communication protocols is not employed. This reduces memory footprint, and eliminates security issues.
- To use `HttpConnection` in the i-αppli API a strict call sequence for the Generic framework API is defined. See Section 5.1 for details.

Notes:

- In the i-αppli Runtime Environment for the PDC mobile phones, GET, POST and HEAD request methods are available for the use of the HTTP(S) protocol. On the other hand, the HEAD method is not supported for the i-αppli Runtime Environment on FOMA mobile phones, and only the GET and the POST methods can be used.
- The HTTP protocol version for PDC mobile phones is 1.0. The HTTP protocol version for FOMA mobile phones is 1.1.
- The number of communication links which can be established at any one time from a single i-αppli is one only. Also, if any packet communication is being performed in the background, including that of native functions, the i-appli cannot use any HTTP communication functions.

5.1 Client-Server Programming

Many programs work as client-server pairs. In the case of i-αppli service, the server is on a remote host system accessible via the Internet, and the client is a Java application running on the mobile phone. A client and server typically communicate by knowing the IP address (a unique identifier for the server) and a port number, which is an identifier that allows a connection to be routed to a specific process. For simplicity, network connections are made to look like I/O streams as defined by the CLDC specification. You simply read and write data by using the usual Java stream classes and methods. For this reason, there is a method to get the input stream of a connection, and another to get the output stream. This allows the client and server to talk to each other. When the client wants an Internet service (such as retrieving a Web page from an HTTP server), it uses the URL to access the service. The i-αppli Runtime Environment provides the `HttpConnection` interface as a subset to `HttpURLConnection` on J2SE. The following code shows an example of setting up and working with (an implementation object of) the `HttpConnection` interface.

Example: Establish and work with a connection

```
package spaceinv;

import java.io.*;
import javax.microedition.io.*;
import com.nttdocomo.io.*;

private String[] getScoreList(int score) {
         try {
                 // Retrieve the HTTP connection (HttpConnection) object.
                 HttpConnection conn = (HttpConnection)Connector.open(
                      "http://backflip.sfbay:8080/scores",Connector.READ_WRITE,
                      true);

                 // set the request method and content type.
                 conn.setRequestMethod(HttpConnection.POST);
                 conn.setRequestProperty("Content-Type","text/plain");

                 // retrieve the output stream
                 OutputStream out = conn.openOutputStream();

                 // write data (because the actual connection is not established yet,
                 // the written data is temporarily stored in the buffer).
                 out.write("Highscore-Game: Space Invaders¥n".getBytes());
                 out.write(("Highscore-Name: " + SpaceInvaders.USER.getName() +
                                                        "¥n").getBytes());
                 out.write(("Highscore-Value: " + score + "¥n").getBytes());

                 // close the output stream.
                 out.close();

                 // make the actual connetion to the remote resource.
                 conn.connect();

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

                 // retrieve the input stream.
                 InputStream in = conn.openInputStream();
                 if (in == null)
                         throw new IOException("high score servlet unreachable");

                 // read data.
                 for (int i=0; i<HighScoreScreen.MAX_ENTRIES; i++) {
                         result[i] = readLine(in);
                 }
```

```
                // close input stream.
                in.close();

                // close the http connection.
                conn.close();

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

`HttpConnection` is an interface used for accessing network resources with the HTTP or HTTPS protocol. Instances of classes implementing this interface can be used to both read from and write to resources specified with a URL. In general, connection with the URL follows the following procedure:

1. Call the `Connector.open()` method, and create a connection object. The following correspond to the open mode and request methods:

   `Connector.READ` : Corresponds to GET and HEAD methods (only on PDC mobile phones)

   `Connector.READ_WRITE`: Corresponds to POST method
2. Set the setup parameter (request method and request header). Two request headers can be set from the application program: `If-Modified-Since` and `Content-Type`.
3. Call the `HttpConnection.openOutputStream()` method, and get the output stream of this connection.
4. Write to the output stream using the `OutputStream.write()` method.
5. Close the output stream using the `OutputStream.close()` method.
6. Actually connect to the remote object using the `HttpConnection.connect()` method. After this, it is possible to refer to the response header as needed. For information regarding the response headers that can be referenced from the application program, refer to the API Reference (`HttpConnection.getHeaderField()` method).
7. Get the input stream using the `HttpConnection.openInputStream()` method.
8. Read from the input stream using the `InputStream.read()` method.
9. Close the input stream using the `InputStream.close()` method.
10. Close the connection using the `HttpConnection.close()` method.

Steps 3 – 5, above, correspond to writing the HTTP request body when using the POST method. Steps 7 – 9, meanwhile, correspond to reading the HTTP response body using both the POST or GET methods.

None of these processes are compulsory and should be used whenever so required; however, when such processing is being carried out, please ensure compliance with the overall sequence as described above. For example, calling `HttpConnection.openOutputStream()` after calling `HttpConnection.connect()` could generate an exception on some devices.

A figure illustrating the state transitions from connection start to finish is shown below:

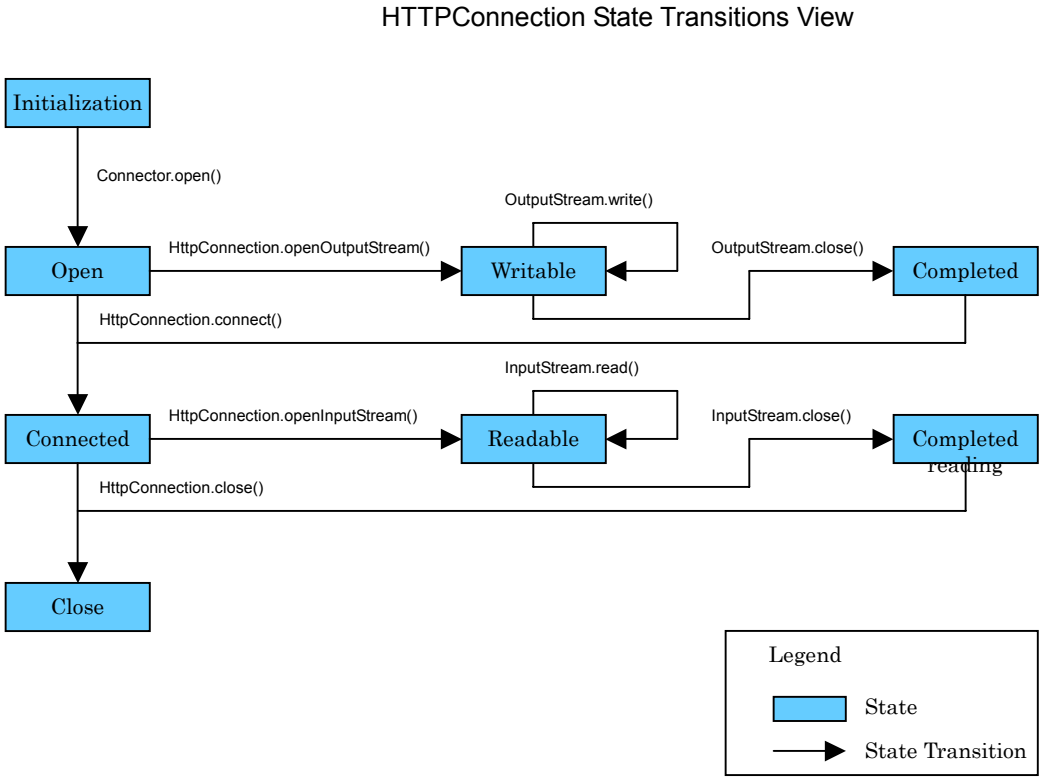


Figure 8: HttpConnection State Transitions

Notes:

- The URL passed to `HttpConnection` has the following limitations:
  - The IP address must be symbolic (e.g. "www.nttdocomo.co.jp"). `HttpConnection` cannot use numerical IP addresses like "127.0.0.1".
  - The URL protocol, host name, and port passed to `HttpConnection` must be the same as that of the URL from which the i-αppli was downloaded (specified by the ADF's `PackageURL` key).
- The multiplicity of `HTTPConnection` from any one i-αppli at any one time is one.
- The i-αppli service client-server system interposes an i-mode server between the client (mobile phone) and server (Web server). The i-mode server sometimes returns a response status as 200 when it detects one of the errors shown below. When this happens, the client side can't check for normality based on the response status. In order for the client side to conduct a strict normality check, have it make a decision based on what was added to the unique HTTP response header and HTTP response body content.
  - When the internet proxy server for the i-mode server is congested.
  - During maintenance of the public menu site (upon request from a public menu site content provider)

  Note that if a response status is returned indicated an error, the i-αppli Execution will throw an exception.
- Use the following utilities to encode and decode HTTP request/response bodies to and from x-www-form-urlencoded format:
  - `com.nttdocomo.net.URLEncoder`
  - `com.nttdocomo.net.URLDecoder`
- The size of request body that can be sent as well as the size of response body that can be received using HTTP(S) communications are limited as follows by profile versions:

Profile	PDC Mobile Phones		FOMA Mobile Phones	
	Request	Response	Request	Response
DoJa-1.x	5 Kbytes	10 Kbytes	80 Kbytes	100 Kbytes
DoJa-2.x	5 Kbytes	10 Kbytes	80 Kbytes	150 Kbytes
DoJa-3.x	10 Kbytes	20 Kbytes	80 Kbytes	150 Kbytes
DoJa-4.x / 5.x	――――	――――	80 Kbytes	150 Kbytes

These restrictions apply not only to situations where application programs are performing direct `HTTPConnection` operations, but also when media data is acquired through HTTP communication

- Sometimes, the URL accessed by the i-αppli will have a BASIC authentication setting on the server side. When this type of URL is accessed, the mobile phone will automatically display a dialog prompting the user to enter a user ID and password. The application program cannot independently send BASIC authentication information (e.g. an `Authorization` request header) to the server.
- The i-αppli will respond to an incoming phone call during communication in accordance with the mobile phone's i-mode incoming call setting.
- When sending an HTTP request to a Web server using the POST method, be sure to specify an appropriate content type in the HTTP request body settings. Set the content type (`Content-Type` header) using the `HttpConnection.setRequestProperty()` method. Furthermore, if there is no data to be written into the request body, the GET method is to be used instead of the POST method. When using the former, please ensure that no setting is made for the content type.
- For i-mode services, it is imperative that a content-length setting (i.e., `Content-Length` header) be made in the response from the web server. Also in the case of HTTP(S) communication from an i-αppli, please be sure to set the content length in the web server's response.
- If an i-αppli is suspended (see item 3.6) while it is carrying out data communication, this communication will also be suspended. In such a case, an exception will be generated as soon as the i-αppli is restarted.
- The `getHeaderField()` is included in the `HttpConnection` inferface and can be used to view the contents of the response header included in the HTTP response. This method can also retrieve user-defined headers that begin with "x-"; however, when dealing with i-mode, i-mode servers will use their own unique user defined headers to control communication with the mobile phone and therefore it is strongly not recommended that you use user defined headers in your i-applis. If a header with the same name as one used by an i-mode server is used in HTTP communications within an i-appli, the i-mode server may delete that header.

[DoJa-2.0]

- Mobile phones which are compatible with DoJa-2.0 Profile feature a Self Mode function which terminates all external communication and allows usage only of functions which do not carry out such communication. If the user sets the mobile phone to Self Mode, the launching of i-αpplis will be possible, but it will not be possible for these i-αpplis to make use of HTTP(S) communication.
- In terms of the URL lengths specified for HTTP(S) communication with DoJa-2.0 Profile or later, settings of up to 255 bytes are possible for scheme and pass, and up to 512 bytes for queries.
- With DoJa-1.0 Profile, the following definitions are made with regard to status codes for HTTP response.
  - `HttpConnection.HTTP_SERVER_ERROR (500)`
  - `HttpConnection.HTTP_INTERNAL_ERROR (501)`

  Compared to this, in case of DoJa-2.0 Profile and later, the status codes are modified as listed below.
  - `HttpConnection.HTTP_INTERNAL_ERROR (500)`
  - `HttpConnection.HTTP_NOT_IMPLEMENTED (501)`

  Note that in order to guarantee compatibility upon compilation, the definition of `HttpConnection.HTTP_SERVER_ERROR (500)` is continued in DoJa-2.0 Profile.

[DoJa-4.0]

Some exception statuses defined in the ConnectionException class indicating an occurrence of a communication exception have been changed in names to integrate with other exception classes in DoJa-4.0.

Old: RESOURCE_BUSY -> New: BUSY_RESOURCE

Old: NO_RESOURCE -> New: NO_RESOURCES

Former names are defined in the DoJa-4.0 to sustain the compatibility, but its use is deprecated.

5.1.1 Terminating Connections

Close the connection between the server and client by using the `close()` method of the `HttpConnection` interface. However, this method will not free the actual physical resource until all I/O streams obtained from `HttpConnection` have been closed.

Note that with DoJa-2.0 Profile or later, if the `connect()` method is in the communicating condition, the `close()` method can be called for that `HttpConnection` instance from other threads so that communication can be terminated.

Notes:

- You need to explicitly close EVERY connection (implementation object of the `Connection` interface) that you retrieve by calling `Connector.open()`, and EVERY I/O stream. Note that this also applies to situations where exceptions are generated for communication errors and the like as a result of the use of `Connection` implementation objects acquired with the `Connector.open()` method. Failing to do so will cause a resource leak, and may prevent you from opening a new connection.

5.2 Session Management

The HTTP protocol does not maintain long-term connections. A connection is established only when client needs to exchange data with a server. On the other hand, user verification is required on servers which permit usage by specified users only; however, it is not practical to login to servers from user each time communication is made. In order to facilitate the repeated exchange of data with servers by a client, it will be necessary to maintain the condition where the user remains logged into the system.

The method of solving this problem is called session management. With the Session Management method, when a user logs onto the system, the session manager creates a string (session ID) uniquely identifying that session. Then after this, the server and client exchange session IDs to identify each HTTP connection owner (logged-on user). In order to maintain security, session IDs are invalidated after a set amount of time has passed without any access. Owners of invalidated session IDs have to go back and log on to use the system again.

In general, cookies are widely used for the actual session management implementation. Cookies maintain a session between the client and server by setting user information – in this case, the session ID – in specific HTTP request/response headers. In the i-αppli Runtime Environment, however, the only information that can be appended to an HTTP request header is `Content-Type` and `If-Modified-Since`, so cookies cannot be used. In the i-αppli Runtime Environment, set the session ID in the request's target URL or HTTP body instead of using cookies.

In the following example, session IDs are exchanged using HTTP request/response bodies instead of cookies.

Example: Sample session management implementation

```
package banking;
```

```
import java.io.*;
import javax.microedition.io.*;
import com.nttdocomo.io.HttpConnection;

/**
 * This class implements a communication process with the bank server.
 */
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;

         /**
         * Create an instance of this class. At this time, establish the first connection
to the server
         * to check the entered ID and password.
         * @param url  URL to the bank server
         * @param acct  Account name of the user
         * @param pin  Account password of the user
         * @exception  When a connection cannot be established to the server, or
         * the entered ID and password are rejected, IOException is thrown.
         */
         public BankingClient(String url, String acct, String pin)
                                                          throws IOException {
                 this.url = url;
                 conn = (HttpConnection)Connector.open(url,
                                                 Connector.READ_WRITE, true);
                 conn.setRequestMethod(HttpConnection.POST);
                 conn.setRequestProperty("Content-Type","text/plain");
                 OutputStream out = conn.openOutputStream();

                 // Set the account information to the HTTP request body.
                 out.write((VERIFY + " " + acct + " " + pin).getBytes());
                 out.close();

                 conn.connect();

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

         private String getCookie(String line) {
                 String header = "Cookie: ";
                 // Acquire session ID from the line that is read from the HTTP
                 // response body.
                 if (!line.startsWith(header) || line.length() <= header.length()))
                         return null;
                 return line.substring(header.length());
         }

        private String contactServer(String cmd) {
                 try {
                         // Close an old connection if there is any.
                         if (in != null) in.close();
                         if (conn != null) conn.close();

                         // Reopen the connection to the server. At this time,
                         // set the session ID acquired from the initial connection
```

```
                        // by the contractor of the instance
                        // to the HTTP request body.
                        conn = (HttpConnection)Connector.open(url,
                                                Connector.READ_WRITE, true);
                        conn.setRequestMethod(HttpConnection.POST);
                        conn.setRequestProperty("Content-Type","text/plain");
                        OutputStream out = conn.openOutputStream();
                        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 readLine(in);

                       // Do not close the input stream and connection
                       // because more data is read from this connection in the
                       // subsequent processing.
                       // The connection is closed when this method is called next
                       // time or by calling the close() method of this class.
                } catch (IOException ioe) {
                        return null;
                }
        }
}
```

5.3 Secure Communication Using HTTPS

Client server applications that require secure connections will use the HTTPS protocol. The client program code for connecting to an HTTPS server will be exactly the same as for the HTTP protocol, except that the protocol session in the URL string will be set to "`https`" instead of "`http`". For example, to communicate the Space Invaders high scores and names between client and server securely, the only change needed is the URL string. In other words, if a service using HTTP protocol is provided at Port No. 8080 on the server (i.e., `backflip.sfbay`), and if a similar service using HTTPS protocol is provided at Port No. 7070, then the URL: "`http://backflip.sfbay:8080/`" is to be changed to "`https://backflip.sfbay:7070/`".

Notes:

- The URL passed to HttpConnection must specify the same protocol, host, and port number as the URL when the i-appli was downloaded. Therefore, if the i-appli was not downloaded using the HTTPS protocol, the i-appli itself will be unable to use the HTTPS protocol. This may also mean that an i-αppli downloaded using https will not be able to use the HTTP protocol.
- Web servers using HTTPS communications are required to install a server certificate containing the signature of a certification authority which supports i-appli compatible mobile phones. i-appli compatible mobile phones support certificates from the following root certification authorities.
  1. Verisign
     - VeriSign class 3 Primary CA root certificate
     - VeriSign class 3 Primary root certificate G2
     - RSA secure server root certificate
  2. Cybertrust (Formerly Betrusted Japan)
     - GTE CyberTrust Root CA (*2)

– GTE CyberTrust Global Root
– Baltimore CyberTrust Root (*1)

3. GlobalSign (Formerly GeoTrust)
   - Equifax Secure Certificate Authority (*1)
   - Equifax Secure eBusiness CA-1 (*1)
   - GeoTrust Global CA (*1)
   - GlobalSign Root CA (*4)
   - GlobalSign Root CA – R2 (*4)
4. RSA Security
   - ValiCert Class 3 Policy Validation Authority (*3)
   - RSA Security 2048 V3 (*3)
5. Secom Trust Systems
   - Security Communication Root CA 1 (*3)

(*1) These certificates are implemented only in mobile phones, which correspond to DoJa-4.0 Profile and later.

(*2) Certificates installed in mobile phones for GTE CyberTrust Root CA will expire on February 24, 2006 due to the closure of the certification authority's services. Models manufactured after this data will not contain certificates for this certification authority.

(*3) These certificates are installed on FOMA 902iS series and newer phones. The actual list of models containing these certificates will be announced by NTT DOCOMO separately.

(*4) These certificates are installed on FOMA 905i series and newer phones. The actual list of models containing these certificates will be announced by NTT DOCOMO separately.

[DoJa-2.0]

With DoJa-2.0 Profile or later, if SSL data for a verification error or the like is generated during the execution of SSL communication, `ConnectionException(SSL_ERROR)` will be thrown. The exception status `SSL_ERROR` was added with DoJa-2.0 Profile.

Chapter 6
Text Processing

In general application text processing, in addition to the ASCII character set, the character set specific to each country is often used. The i-mode service employs Shift-JIS as the encoding method for Japanese text, but the i-αppli runtime environment uses the CLDC internationalized stream input/output structure, making it simple to handle Shift-JIS Japanese text.

6.1 Text Processing

The Java Virtual Machine performs all internal text processing in Unicode. For this reason, any text data encoded in a format other than Unicode must be converted to Unicode to become a Java character stream. CLDC posseses an internationalized stream input/output structure similar to that of J2SE, and classes are provided by the `java.io` package for character code conversion between Unicode character streams and other text byte streams.The `InputStreamReader` class converts a byte stream into a character stream, and the `OutputStreamWriter` class converts a character string into a byte stream. The following figure illustrates the conversion process.

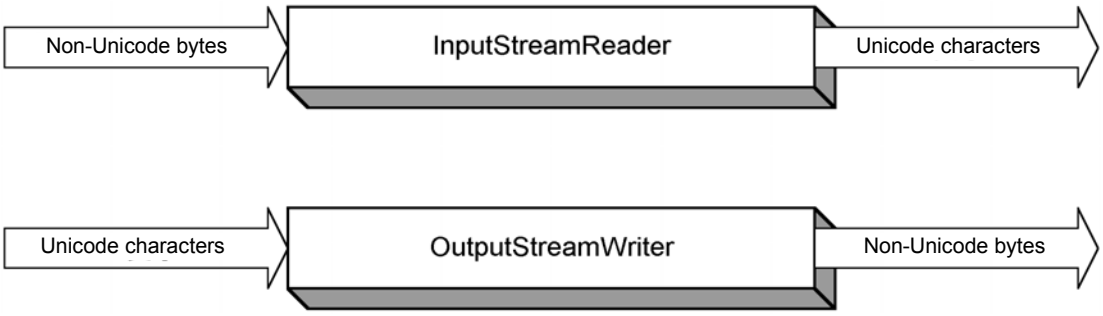


Figure 9: I/O Stream Conversion

The application program must use the `InputStreamReader` and `OutputStreamWriter` classes to ensure proper conversion of input and output text from the network. The i-mode service employs Shift-JIS as the encoding method for Japanese text, and the default encoding for these classes is SJIS.

The following code sample shows the conversion methods.

Example: Conversion of SJIS byte data into Unicode characters

```
java.io.*;
import com.nttdocomo.io.*;
import javax.microedition.io.*;
```

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

// Convert SJIS byte data into Unicode characters. (HTTP Response Body)
isr = new InputStreamReader (conn.openInputStream()); //default encoding is SJIS
 ...
```

Example: Conversion of Unicode characters into SJIS byte data.

```
java.io.*;
import com.nttdocomo.io.*;
import javax.microedition.io.*;

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

// Convert Unicode characters into SJIS byte data. (HTTP Request Body)
osw = new OutputStreamWriter (conn.openOutputStream()); //default encoding is SJIS
 ...
conn.connect();
```

[DoJa-3.0]

The following two items have been added in DoJa-3.0 Profile's `com.nttdocomo.io` package:

Buffered reader (`BufferedReader`)

Writer for formatting output (`PrintWriter`)

These classes offer functionality corresponding to a subset of the classes of the same names contained in the J2SE `java.io` package.

6.2 Pictographic Characters on i-αppli Compatible Mobile Phones

Pictographic characters (i-mode external character set) supported by traditional i-mode content can also be used in i-αppli services. Below is a code table (SJIS/Unicode matrix) showing the pictographic characters defined by NTT DOCOMO, and installed on every mobile phone. Codes use hexadecimal notation.

Be aware that the font displayed in the table is slightly different from the one actually installed on mobile phones. In general, the font installed on actual mobile phones is more grainy, due to limitations on the number of dots available for drawing characters.

<Basic Pictographic Characters>

Character	SJIS	Unicode	Title	Character	SJIS	Unicode	Title
	F89F	E63E	Sunny		F8CA	E669	Hotel
	F8A0	E63F	Cloudy		F8CB	E66A	Convenience Store
	F8A1	E640	Rain		F8CC	E66B	Gas Station
	F8A2	E641	Snow		F8CD	E66C	Parking
	F8A3	E642	Lightning		F8CE	E66D	Traffic Light
	F8A4	E643	Typhoon		F8CF	E66E	Bathroom
	F8A5	E644	Fog		F8D0	E66F	Restaurant
	F8A6	E645	Drizzle		F8D1	E670	Coffee Shop
	F8A7	E646	Aries		F8D2	E671	Bar
	F8A8	E647	Taurus		F8D3	E672	Beer
	F8A9	E648	Gemini		F8D4	E673	Fast Food
	F8AA	E649	Cancer		F8D5	E674	Boutique
	F8AB	E64A	Leo		F8D6	E675	Beauty Salon
	F8AC	E64B	Virgo		F8D7	E676	Karaoke
	F8AD	E64C	Libra		F8D8	E677	Movie
	F8AE	E64D	Scorpio		F8D9	E678	Right Upward Diagonal
	F8AF	E64E	Sagittarius		F8DA	E679	Amusement Park
	F8B0	E64F	Capricorn		F8DB	E67A	Music
	F8B1	E650	Aquarius		F8DC	E67B	Art
	F8B2	E651	Pisces		F8DD	E67C	Play
	F8B3	E652	Sports		F8DE	E67D	Event
	F8B4	E653	Baseball		F8DF	E67E	Ticket
	F8B5	E654	Golf		F8E0	E67F	Smoking
	F8B6	E655	Tennis		F8E1	E680	No Smoking
	F8B7	E656	Soccer		F8E2	E681	Camera
	F8B8	E657	Skiing		F8E3	E682	Bag/Tote
	F8B9	E658	Basketball		F8E4	E683	Book
	F8BA	E659	Motor Sports		F8E5	E684	Ribbon
	F8BB	E65A	Pager		F8E6	E685	Present
	F8BC	E65B	Train		F8E7	E686	Birthday
	F8BD	E65C	Subway		F8E8	E687	Telephone
	F8BE	E65D	Bullet Train		F8E9	E688	Mobile Phone
	F8BF	E65E	Car (Sedan)		F8EA	E689	Memo
	F8C0	E65F	Car (SUV)		F8EB	E68A	TV
	F8C1	E660	Bus		F8EC	E68B	Game
	F8C2	E661	Boat		F8ED	E68C	CD
	F8C3	E662	Airplane		F8EE	E68D	Heart
	F8C4	E663	House		F8EF	E68E	Spade
	F8C5	E664	Building		F8F0	E68F	Diamond
	F8C6	E665	Post Office		F8F1	E690	Club
	F8C7	E666	Hospital		F8F2	E691	Eye
	F8C8	E667	Bank		F8F3	E692	Ear
	F8C9	E668	ATM		F8F4	E693	Hand (Rock)

Character	SJIS	Unicode	Title	Character	SJIS	Unicode	Title
	F8F5	E694	Hand (Scissors)		F984	D6DF	Free Dial (Toll Free)
	F8F6	E695	Hand (Paper)		F985	E6E0	Pound Key Dial
	F8F7	E696	Right Downward Diagonal		F986	E6E1	Mobile Q
	F8F8	E697	Left Upward Diagonal		F987	E6E2	1
	F8F9	E698	Foot		F988	E6E3	2
	F8FA	E699	Shoe		F989	E6E4	3
	F8FB	E69A	Glasses		F98A	E6E5	4
	F8FC	E69B	Wheelchair		F98B	E6E6	5
	F940	E69C	New Moon		F98C	E6E7	6
	F941	E69D	Quarter Moon		F98D	E6E8	7
	F942	E69E	Half Moon		F98E	E6E9	8
	F943	E69F	Crescent Moon		F98F	E6EA	9
	F944	E6A0	Full Moon		F990	E6EB	0
	F945	E6A1	Dog		F991	E6EC	Black Heart
	F946	E6A2	Cat		F992	E6ED	Fluttering Heart
	F947	E6A3	Resort		F993	E6EE	Broken Heart
	F948	E6A4	Christmas		F994	E6EF	Hearts (Multiple Hearts)
	F949	E6A5	Left Downward Diagonal		F995	E6F0	Cheering (Happy Face)
	F950	E6AC	Clapperboard		F996	E6F1	Grr! (Angry Face)
	F951	E6AD	Bag		F997	E6F2	Aww! (Disappointed Face)
	F952	E6AE	Pen		F998	E6F3	Sniffle (Sad Face)
	F955	E6B1	Doll		F999	E6F4	Dizzy
	F956	E6B2	Chair		F99A	E6F5	Good (Upward Arrow)
	F957	E6B3	Night		F99B	E6F6	Music
	F95B	E6B7	soon		F99C	E6F7	Feelin' Good (Hot Spring)
	F95C	E6B8	on		F99D	E6F8	Cute
	F95D	E6B9	end		F99E	E6F9	Kiss Mark
	F95E	E6BA	Clock		F99F	E6FA	Shiny (New)
	F972	E6CE	phone to		F9A0	E6FB	Idea
	F973	E6CF	mail to		F9A1	E6FC	Disgust (Anger)
	F974	E6D0	fax to		F9A2	E6FD	Punch
	F975	E6D1	i-mode		F9A3	E6FE	Bomb
	F976	E6D2	i-mode (w/ Frame)		F9A4	E6FF	Mood
	F977	E6D3	E-mail		F9A5	E700	Bad (Downward Arrow)
	F978	E6D4	Provided by DOCOMO		F9A6	E701	Sleepy (Asleep)
	F979	E6D5	DOCOMO Point		F9A7	E702	Exclamation
	F97A	E6D6	Fee Service		F9A8	E703	Exclamation & Question
	F97B	E6D7	Free		F9A9	E704	Two Exclamations
	F97C	E6D8	ID		F9AA	E705	Wham! (Collision)
	F97D	E6D9	Password		F9AB	E706	Sweat Beads (Flying Off Face)
	F97E	E6DA	More...		F9AC	E707	Phew! (Sweat Beads)
	F980	E6DB	Clear		F9AD	E708	Dash (Running)
	F981	E6DC	Search (Find)		F9AE	E709	- (Long Vowel Symbol 1)
	F982	E6DD	NEW		F9AF	E70A	- (Long Vowel Symbol 2)
	F983	D6DE	Position Information		F9B0	E70B	Select

<Extended Pictographic Characters>

Character	SJIS	Unicode	Title	Character	SJIS	Unicode	Title
	F9B1	E70C	i-αppli	™	F9D7	E732	Trademark
	F9B2	E70D	i-αppli (Bordered)		F9D8	E733	Running Person
	F9B3	E70E	T-Shirt (Striped)		F9D9	E734	Top Secret
	F9B4	E70F	Purse		F9DA	E735	Recycle
	F9B5	E710	Makeup	®	F9DB	E736	Registered Trademark
	F9B6	E711	Jeans		F9DC	E737	Danger / Warning
	F9B7	E712	Snowboard		F9DD	E738	Prohibited
	F9B8	E713	Chapel		F9DE	E739	Vacant Room, Seat, Parking Space
	F9B9	E714	Door		F9DF	E73A	Pass Mark
	F9BA	E715	Money Bag		F9E0	E73B	No Vacant Room, Seat, Parking Space
	F9BB	E716	PC		F9E1	E73C	Arrow Left/Right
	F9BC	E717	Love Letter		F9E2	E73D	Arrow Up/Down
	F9BD	E718	Wrench		F9E3	E73E	School
	F9BE	E719	Pencil		F9E4	E73F	Wave
	F9BF	E71A	Crown		F9E5	E740	Mount Fuji
	F9C0	E71B	Ring		F9E6	E741	Clover
	F9C1	E71C	Hourglass		F9E7	E742	Cherry
	F9C2	E71D	Bicycle		F9E8	E743	Tulip
	F9C3	E71E	Teacup		F9E9	E744	Banana
	F9C4	E71F	Wrist Watch		F9EA	E745	Apple
	F9C5	E720	Thinking Face		F9EB	E746	Flower Bud
	F9C6	E721	Relieved Face		F9EC	E747	Maple
	F9C7	E722	Cold Sweat		F9ED	E748	Cherry Blossom
	F9C8	E723	Cold Sweat 2		F9EE	E749	Rice Ball
	F9C9	E724	Puffy Face		F9EF	E74A	Short Cake
	F9CA	E725	Dumb Face		F9F0	E74B	Sake Bottle and Cup
	F9CB	E726	Heart-shaped Eyes		F9F1	E74C	Rice Bowl
	F9CC	E727	Finger-Gesture OK		F9F2	E74D	Bread
	F9CD	E728	Sticking Out Tongue		F9F3	E74E	Snail
	F9CE	E729	Wink		F9F4	E74F	Chick
	F9CF	E72A	Happy Face		F9F5	E750	Penguin
	F9D0	E72B	Enduring Face		F9F6	E751	Fish
	F9D1	E72C	Cat 2		F9F7	E752	Delicious!
	F9D2	E72D	Crying Face		F9F8	E753	Laughter
	F9D3	E72E	Tears		F9F9	E754	Horse
	F9D4	E72F	No Good		F9FA	E755	Pig
	F9D5	E730	Clip		F9FB	E756	Wine Glass
©	F9D6	E731	Copyright		F9FC	E757	Thin Face

Notes:

On mobile phones which do not support extended pictographic characters, they will be displayed as spaces or the equivalent. The actual character that will be displayed will depend on the model in question.

Chapter 7
Using the ScratchPad and Resources

This chapter describes how to use the ScratchPad and resources accessible from an i-αppli compatible mobile phone.

The ScratchPad is a storage area on mobile phones used by i-αpplis to store persistent data. Resources are data files (e.g. image and sound files) bundled in a JAR file. As with the discussion of HTTP communications in Chapter 5, both the ScratchPad and resources are accessed using the Generic Connection framework.

7.1 Reading from and Writing to the ScratchPad

i-αppli uses its server or the ScratchPad as the primary location for persistent data (e.g. data that needs to be stored and saved for a long period of time.) In the i-αppli Runtime Environment, a special storage area known as a ScratchPad is assigned to each i-αppli so that the storing of data can be continued even after completion of the i-αppli, and that this data can be used again when the i-αppli is subsequently launched. The limitations on the size of the ScratchPad available are described in Section 3.2. Also note that the only i-αppli that can access a ScratchPad is the one that created it.

If an i-αppli wishes to use the ScratchPad it must request it by specifying the ScratchPad size using the SPsize key in the ADF. The JAM will combine the ScratchPad size with the overall size of the JAR file to determine if there is enough room to download the i-αppli. See Section 15.5.1 for details on how to write an ADF.

The ScratchPad memory model implements a portion of the transaction concept. Any changes you make to the ScratchPad will either be written in full or totally rejected.

The JAM creates and manages storage for each i-αppli whose ADF specifies a ScratchPad size. The points at which the JAM actually writes the data to storage will depend on the optimization of the manufacturer's memory and system model.

The ScratchPad is accessed using the Generic Connection framework.

The format for ScratchPad URLs is as follows:

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

The ScratchPad assigned to the application can be divided and managed up to 16 sections by specifying an ADF `SPsize` key.<In <*number*>, which ScratchPad will be accessed is specified with numbers 0 — 15.

Further, <*start-point*> value in the pos option specifies the access start offset in the specified ScratchPad, while the length option value <*access-length*> specifies a number of bytes from the access start offset. When the pos option is specified, data will be read or written beginning from that position. When the length option is specified with the pos option, the length of the stream will be the length specified in the length option (the length option cannot be specified without also specifying a pos option). As a result of applying the `pos` and

`length` options, no value can be specified which falls outside of the ScratchPad area (when using segment-management, the area of the individual segments).

If the ScratchPad URL is specified to `Connector.open()`, this method returns an implementation instance of `StreamConnection`. The application program can then access the ScratchPad by retrieving I/O streams from this `StreamConnection` instance.

[DoJa-3.0]

ScratchPad segmentation management and the length option was newly added with the DoJa-3.0 Profile.In DoJa-2.0 Profile and earlier, the value of <*number*> specified in the ScratchPad's URL is fixed to 0.See Section 15.5.1 for information about how to declare the ADF (`SPsize` key) to manage the ScratchPad in segments.

In addition, the ScratchPad access is a high-cost process for mobile phones which have restrictions on capabilities and resources. For an i-αppli using a large ScratchPad area, enable access to only a necessary area with the minimum memory use by properly specifying the pos option and length option in the URL when accessing the ScratchPad. Furthermore, depending on the phone model, when reading media data from the ScratchPad (i.e. when specifying a ScratchPad URL in the media data retrieval method of the MediaManager class) in some cases the application's execution performance can be improved by setting an appropriate length in the URL.

The following two examples show how to read from and write to the ScratchPad.

Example: Writing to the ScratchPad area

```
import java.io.*;
import javax.microedition.io.*;

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) {
                // handle ScratchPad I/O exceptions
         }
}
```

Example: reading from the ScratchPad area

```
import java.io.*;
import javax.microedition.io.*;

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) {
                // handle ScratchPad I/O exceptions
         }
}
```

Example: Reading from the ScratchPad area

```
// retrieve the UTF (i.e. Unicode) string from the ScratchPad
// The first four bytes are used for other applications (in the previous example, the
value for the four bytes.)
package spaceinv;

import java.io.*;
import javax.microedition.io.*;

public class UserNameScreen extends Panel implements TimerListener {
        private String name;

        /** Create a new instance of this class. */
        public UserNameScreen() {
                try {
                        DataInputStream in = Connector.openDataInputStream(
                                                "scratchpad:///0;pos=4");
                        name = in.readUTF();
                        in.close();
                } catch (IOException e) {
                        // handle ScratchPad I/O exceptions
                }
                …
         }
         …
}
```

Notes:

- If a `StreamConnection` is opened to a ScratchPad, then `Connector.READ`, `Connector.WRITE`, or `Connector.READ_WRITE` may be specified for the open mode.Either input streams or output streams may be acquired from a ScratchPad which has been opened in `Connector.READ_WRITE` mode. However, it is not possible with a single `StreamConnection` for both the input stream and the output stream to be active at the same time. When one of these is opened, ensure that the other is set to the closed condition.

  [DoJa-2.0]

  ScratchPad opening with `Connector.READ_WRITE` mode is possible only with DoJa-2.0 Profile or later. In the case of DoJa-1.0 Profile, the operations which result when a ScratchPad is opened with `Connector.READ_WRITE` mode will depend on the manufacturer's specifications.

- DoJa-3.0 Profile and later versions support the Jar inflator function (the `com.nttdocomo.util.JarInflater` class) in utilities, and entries of files compressed in the Jar format can be expanded on i-αpplis.When giving `JarInflater` a Jar file from ScratchPad as an input stream, strictly specify the Jar file storage location using the `pos` option and `length` option in the ScratchPad's URL. If this is not done, a file format error exception may be thrown.

  In the example below, the contents of a Jar file stored on the `ScratchPad` are read using the Jar inflater feature.

  Example: Using a Jar file stored in the ScratchPad

  ```
  // Using a Jar file 1867 bytes long, beginning at the 1K byte

  try {
      String url = "scratchpad:///0;pos=1024,length=1867";

      // Open the read stream from ScratchPad.
      InputStream is = Connector.openInputStream(url);
      // Create Jar inflater object.
      JarInflater ji = new JarInflater(is);
      // After creating the Jar inflater object, the input stream
      // specified in the constructor is not needed.
      is.close();

      // Specify the the entry name of the file contained in the Jar
  ```

```
        // file and open the input stream to access the target file.
        InputStream is2 = ji.getInputStream("datafile.bin");
        for (;;) {
            // Process the input data.
            ...
        }
        // Close the Jar entry input stream.
        is2.close();
        // Close the Jar inflater.
        ji.close();
    } catch (IOException ioe) {
        // This exception occurs if there was a ScratchPad access error.
        ...
    } catch (JarFormatException jfe) {
        // This exception occurs if a file format error was detected
        // in the Jar inflater.
        ...
    }
```

Please refer to the API reference for the details on the `JarInflater` class.

7.2 Loading Resources

Image, sound, and other data files bundled in the JAR file are called resources. You can access resources using the Generic Connection framework. Resources can only be opened as READ only. Explicitly request the READ only mode as a parameter to the `Connector.open()` method, or use the convenience API `Connector.openInputStream()` to obtain an input stream without obtaining a `Connection` object. If one of these is not used, an exception will be thrown. The static method `Connector.open()` accepts three parameters:

The URL you want to connect to

The open mode

Timeout exception request flag (true or false)

For accessing the resource, the first parameter is a URL character string with the "`resource:///`<*jar-relative-path*>" format. The <*jar-relative-path*> portion specifies a relative path to the location in the JAR file in which the resource is stored. Set the second parameter to `Connector.READ` (because resources are READ only). Ordinarily, unless opening an HTTP(S) connection or OBEX connection, set false as the third parameter. Do not use the short version of the `open()` method (the one that only takes the URL as a parameter), since the default mode is `Connector.READ_WRITE`.

The resource URL can be the URL string used to access the media data. This allows the application to easily play back GIF and sound files bundled in the JAR file.

The following example shows a resource being used as media data. In this example, the application retrieves a file called "bank.gif" that is bundled in the JAR file, and sets it to the `VisualPresenter`.

Example: Using a resource

```
package banking;

import com.nttdocomo.io.ConnectionException;
import com.nttdocomo.ui.*;
…
/** present the welcome screen. */
class WelcomePanel extends Panel {

        /** Create a new instance of this class. */
        public WelcomePanel() {
```

```
                final Panel instance = this;
                MediaImage bank;

                VisualPresenter vp1 = new VisualPresenter();
                try {
                        bank = MediaManager.getImage(
                        "resource:///BankingDemo/img/bank.gif");
                        bank.use();
                } catch (ConnectionException ce) {
                        // handle resource access exceptions
                }
                vp1.setImage(bank);
                …
        }
        …
}
```

7.3 Error Processing

i-αpplis that utilize the ScratchPad and resource URLs should handle the following error exceptions:

ConnectionNotFoundException

Will be thrown if the URL is invalid. Invalid means that the i-αppli Runtime Environment cannot parse the specified URL format, or that the URL type and open mode are in conflict.

ConnectionException

Status: SCRATCHPAD_OVERSIZE

Will be thrown if an attempt is made to write past the end of the ScratchPad.

Status: ILLEGAL_STATE

Will be thrown if a method call violates the state of the connection object. A method call violating the state of a connection object means that (for instance) the `openOutputStream()` method was called on a connection object that was created with READ mode.

Status: BUSY_RESOURCE

Will be thrown if more than one thread tries to open a ScratchPad or resource at the same time.

For other error conditions, an `IOException` or a sub-class of `IOException` will be thrown.

[DoJa-2.0]

With DoJa-1.0 Profile, it is specified that an `EOFException` should occur when reading is carried out beyond the end of the ScratchPad.In contrast, even if this should occur with DoJa-2.0 Profile or later, operations will conform with those for each stream class's methods as defined for CLDC. For example, the value "-1" will be returned if the `InputStream.read()` method is instructed to read a byte at a position beyond the end of the ScratchPad.

[DoJa-4.0]

There have been name changes for some exception statuses defined in the class ConnectionException changed in names in order to integrate with other exception classes in DoJa-4.0.

Old: RESOURCE_BUSY -> New: BUSY_RESOURCE

Old: NO_RESOURCE -> New: NO_RESOURCES

Former names are defined in the DoJa-4.0 to sustain compatibility, but their use is not recommended.

Chapter 8
Accessing Platform Resources

The `PhoneSystem` class defined in the `com.nttdocomo.ui` package is provided for the purpose of controlling the mobile phone's platform resources from an i-áppli. Contained within the `PhoneSystem` class are the `setAttribute()` method for setting attribute values for platform resources, the `getAttribute()` method for referencing the attribute values of platform resources, and the `isAvailable()` method for testing whether a specific platform resource can be controlled on the model in question. To control the platform resources, specify the platform resource type and the corresponding attribute value. The following lists the standard platform resource types that are defined in the i-αppli Runtime Environment.

- Backlight for the display (can be switched on or off)
- Vibrator (can be switched on or off)
- Unit fold-up status (only for referring to attribute values. Only valid for fold-up and flip-style mobile phones)
- Mail/messages received status (only for referring to attribute values)
- Battery level (only for referring to attribute values)
- Signal strength (only for referring to attribute values)
- Status of manner mode setting (only for referring to attribute values)

All of these attributes are defined in the `PhoneSystem` class.

Furthermore, the `PhoneSystem` class also contains the method `playSound()` which facilitates the playback of effect sounds that have already been included in the mobile phone. Although the different types of effect sound are defined in the `PhoneSystem` class, the sounds which may actually be played will depend on the manufacturer in question.

[DoJa-2.x] / [DoJa-3.x] / [DoJa-4.x] / [DoJa-5.x]

The following describes the implemented profile and the positioning for each platform resource. In the chart, "Standard" indicates the resource is located in the i-αppli standard API, while "Option" indicates that it's located in i-αppli Options. Also, "x" indicates that the function is not defined in that profile.

Platform Resource	DoJa-1.x	DoJa-2.x	DoJa-3.x	DoJa-4.x/5.0	DoJa-5.1
Display backlight	Standard	Standard	Standard	Standard	Standard
Vibrator	×	Optional	Standard	Standard	Standard
Fold-up Status	×	Optional	Standard (*1)	Standard (*1)	Standard (*1)
Mail/message received status	×	Standard	Standard	Standard	Standard
Battery level	×	×	Standard	Standard	Standard
Signal strength	×	×	Standard	Standard	Standard

					(*2)
Manner (silent) mode setting status	×	×	Standard	Standard	Standard
Effect sound playback	×	Optional	Standard	Standard	Standard
Additional Key Group	×	×	Optional	Optional	Optional
Screen Visible/Invisible Status	×	×	Optional	Optional	Optional
Audio Output Surround-sound Control	×	×	×	Optional	Optional
Area Information	×	×	×	×	Standard (*2)

(*1) Phone fold-up status is only enabled in folding or flip-type cellular phones.

(*2) Using the signal strength resource (`DEV_SERVICEAREA`) is not recommended for DoJa-5.1 Profile or newer.In DoJa-5.1 Profile or later, the area information resource (`DEV_AREAINFO`) was newly defined in order to more comprehensively examine signal status.

Notes:

- Mail/messages received status, Battery Level, Signal Strength (or the related Area Information functionality) and status of Manner Mode settings operate the various icons shown on the cellular phone's display. For i-αppli to reference this information, it must be declared that the i-αppli will reference the cellular phone's icon information in the ADF `GetSysInfo` key, and the user must grant permission for the i-αppli to reference the icon information. See Section 15.5.1 for details on how to write an ADF.
- Please also be aware that as a result of linking of the mail and message receipt condition with the mobile phone's icon for mail and message receipt, cases may occur where notification of mail and messages may not be received as a result of signal conditions, and in this type of situation, the desired effect may not be achieved.
- When i-αppli continuously turns on a platform resource with high power consumption, such as the backlight or vibrator of the screen, for a long time, depending on the model, the system automatically turns it off to save power. In the platform resource condition as returned by the method `PhoneSystem.getAttribute()`, the results of automatic condition switching by such a function will also be reflected.
- The backlight cannot be controlled by inactive stand-by applications.
- When the key operation of the mobile phone is disabled due to the settings of the mobile phone, i-αppli can not control the vibrator.
- In DoJa-5.1 Profile or later, the area information resource (`DEV_AREAINFO`) was newly defined in order to more comprehensively examine signal status. Using the area information resource, you can find out detailed signal status information such as whether the mobile phone is within a FOMA HSDPA area or not. However, even if the mobile phone is in a HSDPA area, whether or not high speed HSDPA communication can be utilized is dependent upon the usage status of that area's base station at that particular time. Note that if a large number of users are already using HSDPA, HSDPA's high speed communication may not be available even if the mobile phone is in a HSDPA area.

Chapter 9
Stand-By Applications

DoJa-2.0 Profile or later provides support for stand-by application functionality, thus allowing the calendars, images, and other similar items displayed on the mobile phone's stand-by screen to be replaced with resident system i-αpplis which execute to provide display content. With DoJa-1.0 Profile, in order to satisfy the desire to make i-αpplis resident, a solution was necessary in which i-αpplis would be capable of continuous execution. However, in this method, there are some problems, which degrade the operability of i-αpplis and mobile phone functions. As a solution to these problems, the stand-by application functionality from DoJa-2.0 Profile or later uses a mechanism which realizes easy-to-use resident applications.

9.1 Overview of Stand-by Application Mode

9.1.1 Features of Stand-by Applications

When the creation of resident applications using stand-by application functionality is compared with the creation of resident applications within the range of DoJa-1.0 Profile functions, the following advantages can be identified.

Improved Operability

When a normal i-αppli is executed, key operation on the mobile phone is received as an event of the i-αppli. For example, if the number key is pressed on a mobile phone in the stand-by condition, the user-interface mechanism for the native operating system (hereinafter the "native OS") would capture the event and would then switch automatically to phone call mode. However, during the execution of an i-appli, all number-key operations are forwarded to the i-appli as key events. Accordingly, if users of resident applications created using conventional methods desire to make a phone call or to use the web browser, it has been necessary to manually stop the current application. Also, after these operations are completed, users have had to restart the i-αppli manually.

In contrast to this situation, stand-by application functionality allows users to make simple key operations which switch between the receipt mode for key events by i-αpplis (i.e., active mode) and the receipt mode for key events by the mobile phone's native OS (see the following pages). Generally, users implement stand-by applications in a manner where the said i-αpplis are made resident in the receipt mode for key events by the native OS, and where mode switching is carried out whenever necessary to allow interaction with the i-αppli.

Note that in a condition where applications are resident, if users launch a phone call, a web browser, or some other type of native function, the stand-by application will be automatically suspended or terminated. When users finish these operations and the system returns to the stand-by status again, the stand-by application automatically restarts.

Curbing Power Consumption

When a normal i-αppli is executed or a stand-by application is in active mode, the battery power continues to be consumed as the application is executed, and it is not possible to continue execution for extended periods of time. For stand-by applications, it is possible to design resident applications, which can run for a long time, by selectively using the following two modes other than the active mode:

- Inactive mode in which only key event occurrences are not notified to the i-αppli, and processing within the application program can continue
- Dormant mode in which key event occurrence is not notified to the i-αppli and processing within the application program stops (sleeps) so that extra power consumption can be minimized

In the dormant mode, the execution of the application program stops. However, when a specific system event occurs (for example, when a certain amount of time has elapsed), the inactive mode is restored, and it is possible to execute the application program again.

Making the Reception of Mail and Messages Real-Time

When a normal i-αppli is executed, even when a notification is received to indicate that a mail or message has arrived at the server, the execution of the i-αppli takes precedence and the mail or message is not received. Therefore, the users of resident applications that are created by the conventional method sometimes do not notice that a mail or message has arrived at the server.

Just as is the case when a normal i-αppli is executed, in the active mode, a standby application does not receive a mail or message even when it is notified that the message or the mail has arrived at the server (*1). However, in the inactive or dormant mode, it receives mails and messages (in this case, the execution of standby applications is suspended.)

On mobile phones which are compatible with DoJa-2.0 Profile or later, an on-screen mark or the equivalent will provide notification of the receipt of mail or messages even during execution of normal i-αpplis or when in the active mode. In this, real-time user notification has been realized.

(*1) Among FOMA mobile phones, some models may have the ability to receive mail or messages even when a normal i-appli is running or when a stand-by application is in active mode.

As an example of a standby application, the following stock price alarm is considered.

1. Users select beforehand several prices of interested stocks and set stock price conditions that are used to generate an alarm for each stock.
2. The application program will retrieve the latest stock price for the selected stock from the server once an hour and refresh the display. If any stock should exceed the set price conditions, the user will be notified by a sound or vibration alarm.
3. A user who receives the alarm notification can interact with the i-αppli by using key operations, and thus view detailed information about the relevant stock.

From the above example, the processing for step 2 can be realized in the inactive and sleep modes. The application program carries out communication with the server and screen updating while in the inactive mode, and it repeatedly enters the sleep mode at hourly intervals. While this processing is being carried out, all key operations performed by the user are relayed to the native OS, and for this reason, voice-call mode can be entered simply by pressing a number key.

Furthermore, the processing for step 3 is realized by switching the application to the active mode in response to user operations. In the active mode, because the key operations of the user are notified to the i-αppli, the user can interact with the i-αppli.

Notes:

- The mode switching key (hereinafter referred to as the switching key) operation performed by a user supports only a transition from the inactive mode or dormant mode to the active mode. Other mode transitions are implemented by calling a method from the application program.
- The assignment of the switching key differs according to the manufacturer.

9.1.2 User Settings for Stand-by Applications

Users can manually start stand-by applications as well as normal i-αpplis, or these applications can be made resident by registering them in the mobile phone as stand-by applications. The execution of such system-resident stand-by applications will be suspended or ended in situations where voice-call, a web browser, or some other native function is launched as a result of user operations. When users finish these operations and the stand-by status is restored again, the stand-by applications automatically restart. A launch mode of this type is differentiated from a normal i-αppli launch, and is called “Standby Launch”.

The following describes the flow in which a user downloads a stand-by application and specifies the stand-by settings, and then the actual Standby Launch is performed. Note that the flow described here is a typical process flow, and the details of how to operate the mobile phone vary according to the manufacturer.

1. Downloading a stand-by application

Download a stand-by application using the same procedure as a normal i-αppli. If the link for i-αppli download on the browser is clicked, the system will first of all check the ADF content, and if the corresponding i-αppli is determined to be suitable for installation on the mobile phone in question, the JAR file will be downloaded. (For more details, refer to Section 1.4.1.)

Note that the ADF downloaded in this way will contain a declaration (i.e., `MyConcierge` key) that its i-αppli is actually a stand-by application.For more details regarding the `MyConcierge` key and the ADF, refer to Sections 9.2.4 and 15.5.1.

2. Application settings

After the i-αppli has been downloaded, application settings will be carried out in accordance with the content of the ADF. The items of the application settings include stand-by setting, network permission setting, and setting for granting permission for icon information reference, etc. If the ADF declares that its application is a stand-by application, it will also be possible for the user to perform stand-by settings.

To enable Standby Launch for a stand-by application, register the i-αppli to the mobile phone in the stand-by setting. Note that it is not possible to simultaneously enable Standby Launch for multiple i-αpplis. Also, users can later change the stand-by setting that is specified here.

3. Restoring the stand-by status and starting a stand-by application

After performing stand-by settings for an i-αppli in accordance with Step 2 above, if the mobile phone is returned to stand-by condition through, for example, operation of the user's hang-up key, stand-by launching will be carried out automatically for the stand-by application.

Notes:

- A normal launch, as opposed to a “Standby Launch”, refers to all launch modes other than Standby Launch. Specifically, the following applies to a normal launch.
  - When an i-αppli is launched by a user’s direct i-αppli launch instruction such as a menu operation at the i-αppli list.
  - When an i-αppli is launched using the ADF LaunchAt key. (See Section 15.5.1)
  - αppli is launched from another application or external devices through application linking (see Chapter 11).
  - When an i-αppli that launches immediately after download is automatically launched immediately after download.

9.2 Creating a Stand-by Application

This section describes the points for creating stand-by applications, which should be borne in mind and are different from the case of creating a normal i-αppli.

9.2.1 The Stand-by Application Class

In stand-by applications, the i-αppli's main class (i.e., the class specified by the ADF's AppClass key) is created by inheriting `MApplication`. The `MApplication` class is a class inheriting `com.nttdocomo.ui.IApplication`, and it provides methods which, in addition to controlling the life cycle of normal i-αpplis, also control the complicated life cycle of stand-by applications.

The `MApplication` class can, in addition to those methods from the `IApplication` class, also make use of the following methods:

- Control of Mode Transition
  - `deactivate()` : Transitions from the active mode to the inactive mode.
  - `sleep()` : Transitions from the inactive mode to the dormant mode.
  - `isActive()` : Examines whether the current state is active or inactive.
- System Event Control
  - `setWakeupTimer()` : Starts a wake-up timer.
  - `getWakeupTimer()` : Obtains the remaining time of the wake-up timer.
  - `resetWakeupTimer()` : Cancels a wake-up timer.
  - `setClockTick()` : Specifies the generation or non-generation of clock events.
  - `processSystemEvent()` : A handler method for processing system events.

i-αpplis which inherit `MApplication` are not restricted to stand-by launching; rather, they can be launched in the same way as normal i-αpplis by, for example, manual user operations or timer launching. However, in this case, it is not possible to call these methods from the application program. If an i-αppli that is not launched in stand-by calls these methods, an exception occurs.

An explanation of the life cycle and mode transitions for stand-by applications is provided in Section 9.2.2, and an explanation of system events is provided in Section 9.2.3.

[DoJa-5.1]

The `isActive()` method was newly added from DoJa-5.1 Profile. As with the other methods discussed in this section, this method may only be used when a stand-by application is in stand-by launch mode.

9.2.2 Life Cycle and State Transition of Stand-by Applications

This section will provide an explanation of the three types of modes that stand-by applications can adopt.

1. Active mode

A mode in which a stand-by application can receive key and softkey events in the same way as a normal i-αppli is executed. Because softkey event processing is entrusted to a stand-by application, the display of the softkey label is performed at the stand-by application end.

2. Inactive mode

Although the stand-by application continues to execute and the screen display is rendered in accordance with the stand-by application's instructions, key events and soft-key events are not notified to the stand-by application, and are instead notified to the native OS. Since the processing of soft-key events is entrusted to the native OS in such a case, it is not possible for the stand-by application to carry out the setting of soft-key labels.

Immediately after a stand-by application is launched in stand-by, it enters the inactive mode.

3. Dormant mode

To control the battery consumption, the operation of stand-by applications is stopped. On the screen, the rendered content that existed immediately before the stand-by application entered dormant mode are retained. A transition to the dormant mode is possible only from the inactive mode, and it is not possible to transition from the active mode to the dormant mode.

Notes:

- Although key events are notified to the native OS in the inactive mode, screen display will be entrusted to the application. Therefore, when a stand-by application in the inactive mode displays a screen prompting a user for key operation, the user may perform incorrect operations. To create an easy-to-use stand-by application, it is necessary to pay attention to the displayed contents of the screen in the inactive mode.
- During i-αppli execution, a system dialog (JAM native component) rather than the application program's will sometimes be displayed.
  - During HTTP communication, an ID and password are requested for BASIC authentication
  - During HTTPS communication, the server's certification is determined to be not-completely safe and it is necessary to confirm continued operation with the user
  - The application program calls an API with features that require user verification to be executed, such as application linking functionality

  When a stand-by application is in the inactive mode, the system cannot display dialogs. Note that an exception occurs in this case.

During execution of a stand-by application, transition between these three modes will take place as a result of factors such as the calling of mode-transition control methods as described in Section 9.2.1 or the occurrence of system events.

The following figure shows the mode transition among the active, inactive, and dormant modes.

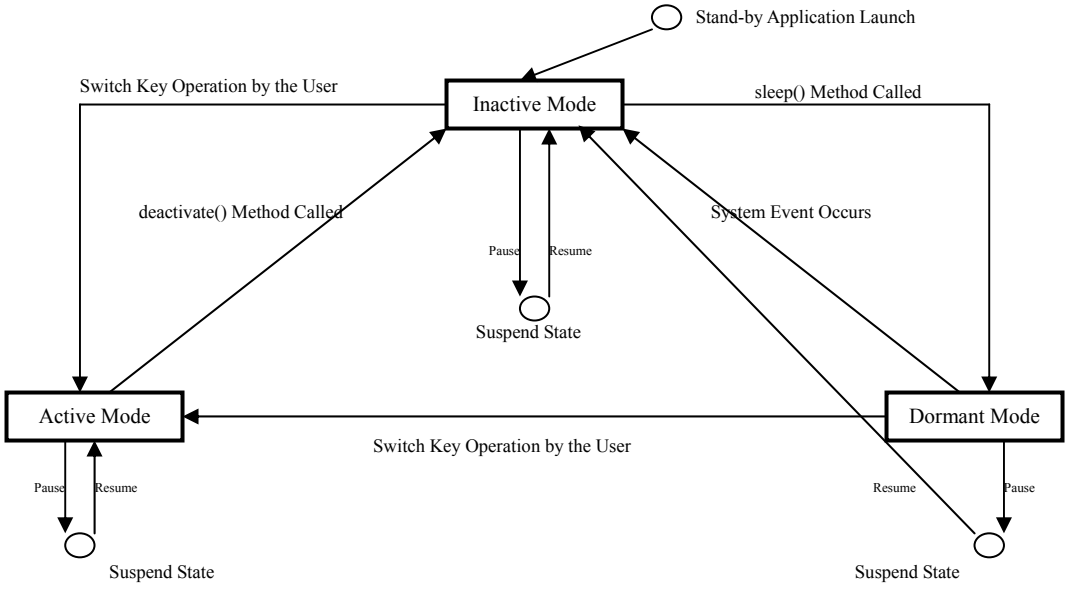


Figure 10: Transition Among Each Mode in a Stand-By Application

The following describes mode transitions among the active, inactive, and dormant modes.

1. Transition from inactive mode to active mode

When a stand-by application is in the inactive mode, if a switching key operation is performed by a user, the stand-by application transitions to the active mode. At this time, the stand-by application receives a system event `MODE_CHANGED_EVENT` notification.

2. Transition from active mode to inactive mode

When a stand-by application executing in the active mode calls the method `MApplication.deactivate()`, transition to the inactive mode will occur. If a stand-by application in the inactive mode calls this method, the method does not execute any operations.

Note that although switching-key operations when in the inactive mode or the dormant mode are not notified to the stand-by application as key events, they are notified as key events (i.e., `KEY_IAPP`) when in the active mode. If the application program is set up in such a way that the `MApplication.deactivate()` method is called in response to the receipt of such a key event, the user will be able to use the switching key to toggle between modes. In such a case, ensure that calling of the `MApplication.deactivate()` method is carried out after first receiving the `KEY_IAPP` release event. If this precaution is not observed, then in the case of certain manufacturers, the system will receive the release event which occurs after the press event, thus resulting in the active mode being adopted once again.

3. Transition from inactive mode to dormant mode

A stand-by application which is executing in the inactive mode can call the `MApplication.sleep()` method to initiate transition to the dormant mode. In this case, if unprocessed events are stored in the event queue, all of them are discarded. A stand-by application that has entered the dormant mode stops until a system event occurs.The thread which called the `sleep()` method will return from this method upon the occurrence of such an event, and the subsequent logic will be started.

If it is desired to return to the inactive mode after a specific period of time has elapsed, the `MApplication.setWakeupTimer()` method can be called in advance of the `sleep()` method. Alternatively, if it is desired to return to the inactive mode based on changes in the native clock's time, it will be necessary to use the `Mapplication.setClockTick()` method in advance to activate clock events.

Note that all stand-by application threads will be stopped upon the adoption of the dormant mode. If the dormant mode is adopted during event processing, rendering methods (i.e., `Canvas.paint()`), or any other threads which may run asynchronously, be aware that these threads will be stopped. If the application program enters the dormant state during communications handling, that communications handling will fail.

4. Transition from dormant mode to active mode

When a stand-by application is in the dormant mode, if a switching key operation is performed by a user, the stand-by application transitions to the active mode. At this time, the stand-by application receives a system event `MODE_CHANGED_EVENT` notification. The thread which called the `sleep()` method will return from this method upon the occurrence of transition from the dormant to the active mode, and processing for the subsequent logic will be started.Return from the `sleep()` method will take place in advance of the `MODE_CHANGED_EVENT` notification.

For more information regarding system events, refer to Section 9.2.3.

5. Transition from dormant mode to inactive mode

If a system event other than `MODE_CHANGED_EVENT or FOLD_CHANGED_EVENT` (when the mobile phone is closed from an open state) occurs while an application is executing in the dormant mode, then transition to inactive mode will take place. The thread which called the `sleep()` method will return from this method upon the occurrence of transition from the dormant to the inactive mode, and processing for the subsequent logic will be started.Return from the `sleep()` method will take place in advance of any system events.

For more information regarding system events, refer to Section 9.2.3.

It is not possible to transition from the active mode to the dormant mode. If a stand-by application in the active mode calls the `sleep()` method, an exception occurs. If it is desired to transfer from the active mode to the dormant mode, the `deactivate()` method should first be used to initiate transition to the inactive mode, and then the `sleep()` method should be called.

A stand-by application needs to understand what kind of mode it is currently in.Appropriately manage the application's own status when there are opportunities for the `MODE_CHANGED_EVENT` to be notified or to call the `deactivate()` method.

A stand-by application is suspended from each mode in the following cases: As with the restart of normal i-αpplis, the `MApplication.resume()` method will be called upon the restarting of a suspended stand-by application.

Mail Status	Cases in which a stand-by application is suspended
Active Mode	An event with a priority higher than normal i-αppli execution occurs, such as when a mobile phone receives a phone call
	That stand-by application calls an application linking method, other than the `MApplication.launch()` method, which requires a user operation (confirmation dialog operation, etc.) (A method like this cannot be called in the inactive mode.)
Inactive Mode Dormant Mode	An event with a priority higher than the execution of an i-αppli in the inactive or dormant mode occurs, such as when a mobile phone receives a phone call, or is notified that a mail or message arrives at the server, or when an alarm notification of the scheduler alarm occurs.
	When a voice-call is made by user operation
	The user launches an application that does not use packet communications (*1).

(*1) By "applications that do not use packet communication" the native scheduler and calculator applications that are loaded onto the mobile phone and have no packet communication functionality (packet communication is not possible) are indicated.

If the mode of a stand-by application during suspension is the active mode or inactive mode, the mode after restart is the same as the one before suspension. In contrast, note that the mode after restart changes to the inactive mode if the mode of a stand-by application during suspension is the dormant mode. If a stand-by application executing in the dormant mode is suspended and restarted, then the thread which is being blocked by the `MApplication.sleep()` method will return from the `sleep()` method and the subsequent processing will be restarted.

Also, a stand-by application in each mode terminates in the following cases: The terminated stand-by application automatically restarts when it is restored to stand-by status again.

Mail Status	Cases in which a stand-by application terminates
Active Mode	`MApplication.terminate()` is called from the stand-by application
	That stand-by application performs application linking with the `MApplication.launch()` method (*1).
	The user forces the termination of an operation
	An external device is connected via a connection cable
	A non-caught exception or error occurs within the stand-by application (*2) (*3)
Inactive Mode	`MApplication.terminate()` is called from the stand-by application
	A forced termination of the executing stand-by application is requested by the user's menu operation
	User operation resulting in the launch of an application that uses packet communication.
	An external device is connected via a connection cable
	A non-caught exception or error occurs within the stand-by application (*2) (*3)
	i-mode functionality or key operation is locked by the user's menu operation In this case, a stand-by application is not launched in stand-by until the lock is released.
Dormant Mode	A forced termination of the executing stand-by application is requested by the user's menu operation
	User operation resulting in the launch of an application that uses packet communication.
	An external device is connected via a connection cable

	i-mode functionality or key operation is locked by the user's menu operation In this case, a stand-by application is not launched in stand-by until the lock is released.

(*1) On DoJa-4.0 Profile and earlier mobile phones, this does not include attempts to launch the Browser via the `MApplication.launch()` method. On DoJa-4.0 Profile and earlier mobile phones, stand-by launched stand-by applications cannot launch the Browser. From DoJa-4.1 Profile onward, in active mode even a stand-by launched stand-by application can launch the Browser. (At that time, the stand-by application will close and the Browser will launch just as if any other application was executed via the linked launch function.)

(*2) When a stand-by application terminates due to an uncaught exception/error, the i-αppli runtime environment will ask the user whether or not to restart that application in stand-by. As a result, if the user selects not to launch the application, the stand-by application settings of the mobile phone are canceled. A stand-by application needs to properly handle exceptions that occur with their own life cycles.
(Note that if a stand-by application violates security severely, the stand-by settings of the mobile phone are canceled without user confirmation.)

Also, in DoJa-2.0 Profile, when a stand-by application terminates due to an uncaught exception/error, that mobile-phone's stand-by settings will be erased without user confirmation.

(*3) When a stand-by application in the inactive mode terminates due to an uncaught exception/error, the mobile-phone logs infomration indicating which stand-by application terminated and when. The user can refer to this log information by a menu operation on the mobile phone, etc.

(*4) By "applications that use packet communication" the browser, mailer, i-αppli etc. that have packet communication functionality (packet communication is possible) are indicated. Note that, regarding the launching of the browser and mailer by user operation, depending on the hardware capacity (for example, installed memory capacity), some models do not terminate the stand-by application in the inactive or dormant mode but suspend it.

As shown in the table, a stand-by application may terminate during execution due to various external factors. Ensure that when data which should be stored is generated, it is immediately saved to either the ScratchPad or the server.

9.2.3 System Events

A stand-by application can handle specific events (system events), which control life cycles. Notification of system events is provided using the `processSystemEvent()` class for each instance of the stand-by application's main class (i.e., the `MApplication` sub-class). A stand-by application can implement system event processing by overriding this method. Furthermore, this method's first parameter provides notification of the type of occurring event, whereas the second parameter provides notification of the event parameters associated with said occurring event.

The following describes the types of system events. The `MApplication` class defines constants which indicate the types for these events.

- `MODE_CHANGED_EVENT`

  Events that indicate a switch key operation has been performed by the user and a transition occurs from the inactive or dormant mode to the active mode. In other mode transitions, this event does not occur.

  Event parameters are not used in this event.

- `CLOCK_TICK_EVENT`

  Event that indicates the time of the native clock changes. The current Profile supports triggering events by the minute. In other words, this event occurs when the time of the native clock indicates 00 seconds every minute. In order to facilitate the generation of this event, it is necessary to call the `MApplication.setClockTick()` method so that the event may be activated.

  This event occurs regardless of the mode of a stand-by application; however, events during which a stand-by application is suspended are discarded.

  The number of seconds which have elapsed from 0 hours, 0 minutes, and 0 seconds on the current day is passed to this event's parameters.

- `WAKEUP_TIMER_EVENT`

This event indicates expiration of the wake-up timer (i.e., the timer which provides notification using a system event that the time set by the stand-by application has expired). A wake-up timer does not stop even when a stand-by application is suspended or restarted. Note that if a timer expires while the stand-by application is suspended, the event notification is delayed until the execution of the stand-by application restarts.

Operation of the wake-up timer is started by calling the `MApplication.setWakeupTimer()` method (Operation occurs only once; does not function as an interval timer). Furthermore, if the `MApplication.resetWakeupTimer()` method is called, it will be possible to stop the wake-up timer which is currently operating. If the `MApplication.getWakeupTimer()` method is called while a wake-up timer is operating, it will be possible to acquire the time from the current time until said timer will expire.

This event occurs regardless of the mode of the stand-by application.

Upon generation of this event, a value of 0 will be passed to its parameters if the event occurred as scheduled, and a value of 1 will be passed if occurrence of the event was delayed as a result of the stand-by application's execution being suspended.

- `FOLD_CHANGED_EVENT`

  Relevant to mobile phones with a fold or flip design, this event indicates when there is a change in the folded condition or the open/closed condition. Note that this event will only be generated in the case of mobile phones with a fold or flip design.

  This event occurs regardless of the mode of the stand-by application; however, events during which the stand-by application is suspended are discarded.

  Upon generation of this event, a value of 0 will be passed to its parameter if the closed condition was adopted, and a value of 1 will be passed if the open condition was adopted.

9.2.4 Declaring a Stand-by Application via the ADF

Stand-by applications are required to use the ADF's `MyConcierge` key to declare that they are in fact stand-by applications (i.e., by setting the value "`Yes`"). An i-αppli for which this key is not set cannot be registered in a mobile phone as a stand-by application.

It is not possible to use an ADF set with a new `MyConcierge` key to upgrade a downloaded normal i-αppli on the cellular phone as a stand-by application. Conversely, it is not possible to use an ADF with its `MyConcierge` key deleted to upgrade a downloaded stand-by application on the cellular phone as a normal i-αppli. In both of these cases, it will be necessary to delete the installed i-αppli from the mobile phone and then to perform a fresh download.

9.3 Launching Other Functions During Execution of a Stand-by Application

When an attempt is made to launch other functions of a mobile phone during the execution of a stand-by application, if the stand-by application is not in the active mode, other functions take precedence as a rule. The following describes the behavior of a stand-by application when an attempt is made to launch other functions during the execution of the stand-by application.

Function to be launched		Behavior of a stand-by application	
		In active mode	In inactive or dormant mode
Launch the other function by user operation			
	Browser Launch	Not possible (*1)	The stand-by application is terminated and the browser is launched. When the user finishes work and the stand-by status is restored, the stand-by application restarts.
	Mailer Launch	Not possible (*1)	The stand-by application is terminated and the mailer is launched. When the user finishes work and the stand-by status is restored, the stand-by application restarts.
	Outgoing voice-call	Not possible (*1)	The stand-by application is suspended and the system moves to an incoming voice-call operation. After the call ends, the stand-by application restarts and enters the inactive mode.
	i-αppli launch	Not possible (*1)	The stand-by application is terminated and the specified i-αppli is launched. When the user finishes work and the stand-by status is restored, the stand-by application restarts.
Launch from within the stand-by application to other functions by application links.			
	Browser Launch	Earlier than DoJa-4.0 Profile: Impossible DoJa-4.1 Profile or Newer: The stand-by application is terminated and the browser is launched. When the user finishes work and the stand-by status is restored, the stand-by application restarts. (*2)	Not possible
	Outgoing voice-call	The stand-by application is suspended and the system moves to an incoming voice-call operation. After the call ends, the stand-by application restarts and enters the active mode.	Not possible (*3)
	i-αppli launch	The stand-by application is terminated and the specified i-αppli is launched. When the user finishes work and the stand-by status is restored, the stand-by application restarts.	Not possible (*3)
	i-αppli update function launch	The stand-by application is ended and JAM (i-αppli update functionality) is launched. When the user finishes work and the stand-by status is restored, the stand-by application restarts. (*4)	Not possible (*3)
	Call native function associated with user operation	The stand-by application is suspended and the native function is called. After the user finishes the operation and ends the native function, the stand-by application restarts and enters the active mode.	Not possible (*3)
Launch other functions using methods other than the above			
	Incoming voice-call	The stand-by application is suspended and the system moves to an incoming voice-call operation. After the call ends, the stand-by application restarts and enters the active mode.	The stand-by application is suspended and the system moves to an incoming voice-call operation. After the call ends, the stand-by application restarts and enters the inactive mode.
	Mail and message receipt in the server	The mail or message is not received and the stand-by application continues to be executed. A mobile phone shows a mark in the screen and notifies the user of the existence of unreceived mail on the i-mode server. In addition, FOMA mobile phones supporting the DoJa-3.5 Profile and later undertake reception of mail messages in the background while running an active standby application.	The stand-by application is suspended and the system moves to mail and message receive operation. After the receipt ends, the stand-by application restarts and enters the inactive mode. However, if mail or message reception is carried out while an application in the inactive mode is carrying out HTTP communication processing, actions for the active mode will be implemented.
	Timer launch of an i-αppli	The i-αppli is not launched by the timer and the stand-by application continues to be executed. By menu operations, etc, a user can know that there is an i-αppli that cannot be launched by a timer.	The stand-by application is ended and the i-αppli is launched by the timer. When the i-αppli that was launched by the timer ends, and the stand-by status is restored, the stand-by application restarts.

	Automatic launch of native applications, which are launched according to schedules, such as an alarm and scheduler	The stand-by application is suspended and a native application is launched. After the native application ends, the stand-by application restarts and enters the active mode.	The stand-by application is suspended and a native application is launched. After the native application ends, the stand-by application restarts and enters the inactive mode.

(*1) Since key events are notified to the stand-by application when executing in active mode, it is not possible for the user to carry out these operations. However, for FOMA mobile phones, which can simultaneously execute multiple tasks, there are cases in which these functions can be used even if a standby application is active by using the task switch function.

(*2) On DoJa-4.0 Profile and earlier mobile phones, stand-by launched stand-by applications cannot launch the Browser. From DoJa-4.1 Profile onward, stand-by launched stand-by applications in active mode can launch the Browser.

(*3) Inactive state stand-by applications cannot launch these features.

(*4) When an i-αppli has finished updating with the i-αppli update feature, that i-αppli will be automatically launched. In case of a stand-by application, this launch is performed as a normal launch. After the stand-by application that has been launched in normal mode ends and the stand-by status is restored, that stand-by application is launched in stand-by.

9.4 Precautions for i-αppli API Usage

In situations where a stand-by application is launched from stand-by mode, the resulting characteristics lead to a different set of precautions for i-αppli API usage than are relevant for normal i-αpplis. This section describes these precautions.

9.4.1 User Interfaces

- The softkey label that is set in the frame is not displayed while a stand-by application is in the inactive or dormant mode. Specifically, the native OS uses the display area for soft-key labels when in either of these modes. When the stand-by application moves to the active mode, the softkey label that is set in the current frame is displayed in the softkey label display area.
- When using a Panel, it will not be possible to carry out operations relevant to focus (i.e., setting of the focus manager or switching of focus between components) from a stand-by application in the inactive mode. Even if the stand-by application in the inactive mode calls a method for focus operation, that method does not execute any operations and the application program is restored.
- When using a Canvas, it will not be possible to acquire the keypad condition (i.e., using the Canvas.getKeyPadState() method) when in the inactive mode. If a stand-by application calls this method while in the inactive mode, the value 0 will be returned, indicating that no key is being pressed.
- The following tables indicate whether or not notification is carried out for stand-by applications in the inactive mode in terms of both high-level events and low-level events. As a fundamental principle, events associated with key operations are not notified to stand-by applications executing in the inactive mode.

High-Level Events

Type of event		Existence of notification	Description
Component events			
	BUTTON_PRESSED	×	
	SELECTION_CHANGED	○	Notification is provided when the `select()` or `deselect()` methods from the ListBox class are used to change the selection condition.
	TEXT_CHANGED	○	Notification is provided when the set`Text()` method from the `TextBox` class is used to perform the resetting of text.

Key Events	×	
Softkey Events	×	
Media Events	○	
Timer Events	○	

Low-Level Events

Type of event	Existence of notification	Description
Key Press Events	×	Including softkey operations
Key Release Events	×	Including softkey operations
Resume Events	○	
Update Events	○	
Timer Events	○	

- Although scroll operations can be carried out for Ticker in the active and inactive modes, this will not be possible upon the adoption of the sleep mode. Scroll operations will become possible once again upon transition from the sleep mode to either active mode or inactive mode.
- Although presenter objects (i.e., VisualPresenter and AudioPresenter) can perform playback processing when in the active mode or inactive mode, any presenter objects currently being played back will be stopped when the stand-by application adopts inactive mode. At that time, a playback stop event does not occur. Also, even if the stand-by application subsequently returns from the dormant mode to either active mode or inactive mode, playback will not start again automatically.
- A timer (i.e., Timer or ShortTimer) will be able to operate when a stand-by application is in either active mode or inactive mode. If a stand-by application which is operating a timer switches to the sleep mode, said timer will be stopped (i.e., as if the stop() method had been called). Even if the stand-by application subsequently returns from the sleep mode to either active mode or inactive mode, operation of the timer will not start again automatically.
- Inactive mode stand-by applications cannot launch the IME from Canvas (call the Canvas.imeOn() method). If inactive mode stand-by applications attempts to launch the IME, an exception will occur.

9.4.2 Input and Output

- When a stand-by application in the inactive mode is carrying out HTTP(S) communication, this communication will be suspended not only when a phone call is received, but also when a phone call is initiated as a result of user operations. In this case, similar to when a communication is suspended by an incoming phone call, it is necessary to retry the communication processing. For more details on the conditions for the suspension and restart of i-αpplis, refer to Section 3.6.
- Stand-by applications are capable of HTTP(S) communication only when in the active mode or inactive mode. If while a thread is carrying out HTTP(S) communication, another thread calls the `MApplication.sleep()` method to implement the sleep mode, said communication will be suspended and an exception will be generated. After that, when the stand-by application restores from the dormant mode to the other mode, a thread that is establishing communication restarts the operation from the exception processing of the communication.
- If while the ScratchPad is writing data to non-volatile memory, a thread calls the `MApplication.sleep()` method to implement the sleep mode, transition to this mode will be delayed until the writing of said data has been completed. Note that if the sleep mode is adopted while the ScratchPad is open, this open condition will be maintained even upon

transition from sleep mode to either active mode or inactive mode, and it will be possible to continue with the writing or reading of data.

- Communication with external devices through the use of OBEX external connection will be possible only when the stand-by application is in the active mode. If a stand-by application using OBEX external connection to perform communication switches to inactive mode, said communication will be suspended and an exception will be generated. Note that this applies both to cases where the stand-by application is operating as an OBEX client and where it is operating as an OBEX server.

9.4.3 Hardware Control

- A stand-by application in inactive mode may not control the backlight. Even if a stand-by application in this mode calls the `PhoneSystem.setAttribute()` method to specify modification of the back-light condition, that method will be returned to the application without having done anything.
- A stand-by application in the active mode can control the backlight; however, the backlight that is turned on by a stand-by application is automatically turned off by the system after a certain period of time elapses.
- Vibrator control can be implemented when a stand-by application is in the active or inactive mode. When the stand-by application transitions to the dormant mode while the vibrator is on, the vibrator stops. Even after the stand-by application is restored to the active or inactive mode, the vibrator operation is not restarted automatically.
- For others such as hardware resource for which a user can set the mode by menu operation, even though an attempt is made to change the mode from a stand-by application, the user settings take precedence as a rule.

9.4.4 Application Linking

- Inactive mode stand-by applications cannot call application linking methods which will require a user operation (such as confirmation of permission to place an outgoing voice-call with the `Phone.call()` method). If an inactive mode stand-by applications attempts to call these methods, an exception will occur.
- In the i-αppli Runtime Environment of profiles proceeding DoJa-4.0 Profile, when a stand-by application has been launched in stand-by, it cannot use the `MApplication.launch()` method to launch the browser under any circumstance. An exception occurs when a stand-by application that has been launched in stand-by attempts to launch a browser.

  Compared to this, in the i-αppli Runtime Environment of DoJa-4.1 Profile and later, even though a stand-by application is launched in stand-by, if it is in the active mode, it can launch a browser by the `MApplication.launch()` method.

Chapter 10
OBEX External Connection

OBEX external-connection functionality enables i-αppli to connect to, and communicate with, external devices via the infra-red port, and this functionality is supported as i-αppli standard API with DoJa-2.0 Profile or later. In this, the term “external device” also includes mobile phones which are compatible with DoJa-2.0 Profile or later; furthermore, support is provided in i-αppli specification for both client API and server API in an OBEX external-connection setup. Accordingly, i-αpplis for the exchange of business cards, schedules, and other small data items between mobile phones may be easily developed.

This chapter will provide a description of the important points that should be known when developing systems intended for OBEX client API, OBEX server API, and external devices other than mobile phones within the i-αppli Runtime Environment.

Note that although OBEX external-connection functionality is implemented in accordance with IrDA standards, said standards will not be dealt with directly in this manual. For more details regarding the IrDA standards, refer to the relevant documentation as published by the Infrared Data Association. Furthermore, OBEX external-connection functionality within DoJa-2.0 Profile to the current profile conforms to each standard of the following versions:

- Infrared Data Association "Link Manager Protocol version 1.1"
- Infrared Data Association "Serial Infrared Physical Layer Specification version 1.3"
- 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 Data Communication

OBEX (IrOBEX) standardizes the procedures for data communication using the infrared port in a format known as “object exchange.” The objects described here refer to general data entities, for example, files that have some meaning when they are grouped.

OBEX itself standardizes the procedures of sending and receiving data (protocols) and does not standardize APIs. In the i-αppli specifications, APIs for handling data transmission/reception by OBEX are defined in the Generic Connection framework.

HTTP was referenced in the determination of the OBEX specification, and this specification conforms with the client-server model wherein the OBEX client sends a request and the OBEX server returns the corresponding response. In particular, the OBEX client requests that the OBEX server carries out processing of an operation (which corresponds to a method in terms of HTTP), and the OBEX server then returns either a response status as the result of the operation or an object’s contents as the results of processing. Within the i-αppli Runtime Environment are provided with the following two operations in order to facilitate the exchange of objects between clients and servers.

- GET operation: The OBEX client requests the OBEX server to transfer an object.
- PUT operation: The OBEX server receives an object from the OBEX client.

With the exception of the fact that it is represented in binary format, this protocol resembles HTTP particularly in terms of the stipulated response-status content and also in that both requests and responses comprise a header and body.

The OBEX standard sets forth detailed procedures for the division and reconstruction of objects which are difficult to handle as a result of, for example, the packet-length restrictions applicable to infra-red communication. However, since the API provided by the i-αppli Runtime Environment conceal these complicated regulations, as a developer, you will be able to create this type of i-αppli with relative ease.

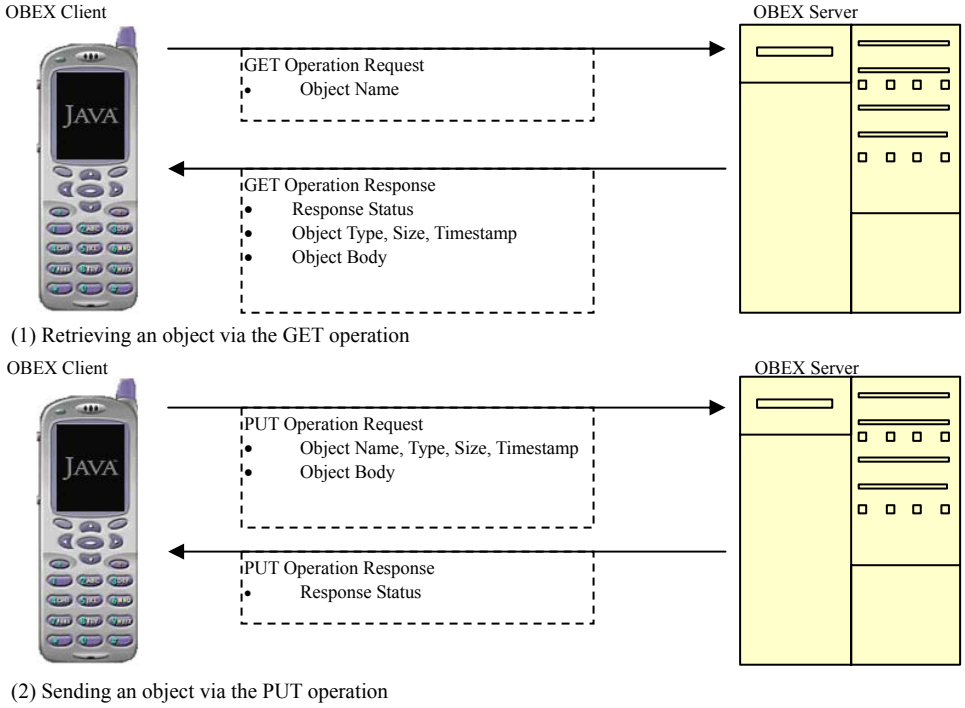


Figure 11: Image of Usage of OBEX Client and OBEX Server

10.2 OBEX External-Connection API

The API for OBEX external connections is provided as a unique i-αppli API set, constructed on the Generic Connection framework. The makeup of the OBEX external-connection API is shown in the following diagram.

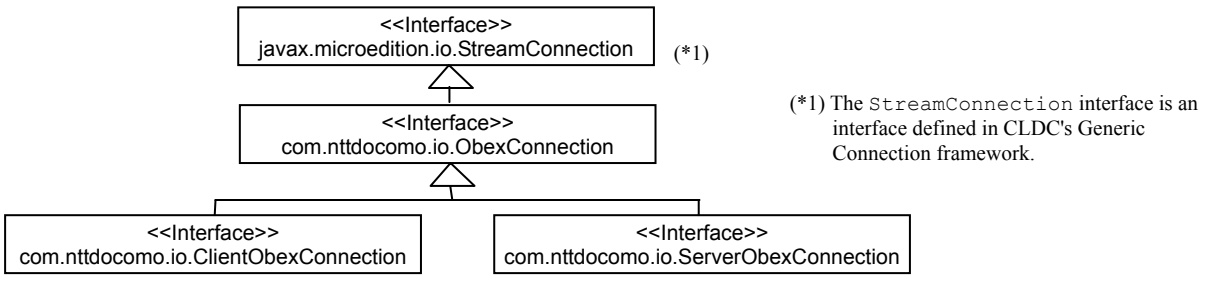


Figure 12: Structure of the OBEX External-Connection API

The `ObexConnection` interface defines the constants and access methods as used commonly by the OBEX client and the OBEX server; furthermore, `ClientObexConnection` and `ServerObexConnection` are provided as sub-interfaces inherited from the ObexConnection interface. The specific methods for OBEX clients and OBEX servers are defined using these two sub-interfaces.

Although the connection objects used for OBEX external connection are acquired through the use of the same `Connector.open()` method as for HTTP communication and the ScratchPad, this connection implements either `ClientObexConnection` or `ServerObexConnection`. The URL specified for the `Connector.open()` method is different for OBEX client applications and OBEX server applications.

The following sections will provide individual descriptions of OBEX client API and OBEX server API.

10.2.1 OBEX Client API

The connection object used by the OBEX client implements the `ClientObexConnection` interface. In order to acquire this connection object, application programs must specify the URL string "`obex:/irclient`" and call the method `Connector.open()`.

The following presents an example of acquisition and usage of the ClientObexConnection implementation interface in an application program.

Example: Usage of `ClientObexConnection`

```
ClientObexConnection coc;
int response;
try {
          // Acquire the ClientObexConnection instance and connect to the OBEX server.
          coc = (ClientObexConnection)Connector.open("obex:/irclient",
                                                     Connector.READ_WRITE, true);
          coc.connect();

          // Execute the GET operation.
          coc.setOperation(ObexConnection.GET);       // Set GET for operation.
          coc.setName("userdata_1");                  Set the name of the object for GET.
          coc.sendRequest();                          // Send the operation.

          // Check the response code that is returned from the OBEX server and if it is normal,
          // read the received object body.
          response = coc.getResponseCode();
          if (response==ObexConnection.SUCCESS) {
                    InputStream in = coc.openInputStream();
                              :        // Read and process the data requested with the GET operation.
                    in.close();
          } else {
                    throw new IOException();
          }

          // Execute the PUT operation
          coc.setOperation(ObexConnection.PUT);       // Execute the PUT operation.
          coc.setName("userdata_2");                  // Set PUT for operation.
          coc.setType("text/plain");                  // Set the type for the PUT object
          coc.setTime(System.currentTimeMillis());    // Set the timestamp for the PUT object.
          OutputStream out = coc.openOutputStream();
                    :        // Write and process the data to be sent with the PUT operation.
          out.close();
          coc.sendRequest();                          // Send the operation.
          // If the response code is normal, close ClientObexConnection and end processing.
          response = coc.getResponseCode();
```

```
            if (response!=ObexConnection.SUCCESS) {
                    throw new IOException();
            }
            coc.close();

} catch (IOException ioe) {
            // Exception processing.
            // Perform processing for IOException and ConnectionException thrown by API and
            // for IOException thrown by the application program.
                    :
}
```

When an application program is using `ClientObexConnection`, it is first of all necessary to call the `connect()` method to establish an infra-red communication link with the OBEX server. Once a communication link has been established, it will be possible to process multiple operations in sequence. In the above sample code, a GET operation and a PUT operation are each processed once on the established one-off communication link, and after that, the link is severed using `ClientObexConnection.close()`.

The processing of operations on the OBEX client is carried out in accordance with the following sequence.

1. The `setOperation()` method is used to set the operation type.
2. The access method defined by `ObexConnection` is used and the setting of OBEX request data is carried out. A Name header, Type header, and Time header are contained within the OBEX request headers which may be set using the access method.
3. In the case of PUT operations, the content of the object body to be sent is written. This operation is carried out using the output stream retrieved using the connection object's `openOutputStream()` or `openDataOutputStream()` method.
4. The `sendRequest()` method is used to send an operation-processing request to the OBEX server. This method is blocked until a response is returned from the OBEX server. When returning after the calling of this method, the content of the OBEX response code or OBEX response header as returned from the OBEX server may be referenced if so required. A Name header, Type header, and Time header are contained within the OBEX response headers which may be referred from the OBEX client. The OBEX response codes are defined in `ObexConnection`. The OBEX response code as defined by `ObexConnection` conforms with the response code regulations from the OBEX standards. For more details regarding the meaning of the various codes, refer to the OBEX specifications. In situations where the OBEX server can perform normal processing for the OBEX client's request, it is standard for the OBEX client application to receive `ObexConnection.SUCCESS` from the i-αppli Runtime Environment (i.e., `ClientObexConnection.getResponseCode()` method) as an OBEX response code.
5. In the case of GET operations, the content of the returned object body is read. This operation is carried out using the input stream retrieved using the connection object's `openInputStream()` or `openDataInputStream()` method. The length of the object body can be acquired using the connection object's `getContentLength()` method.

Please ensure that processing on the OBEX client conforms with the above sequence as a whole. For example, if an attempt is made to acquire the OBEX response header in a situation where no operation processing request is issued to the OBEX server, an exception will be generated.

The following diagram illustrates condition transitions for `ClientObexConnection`.

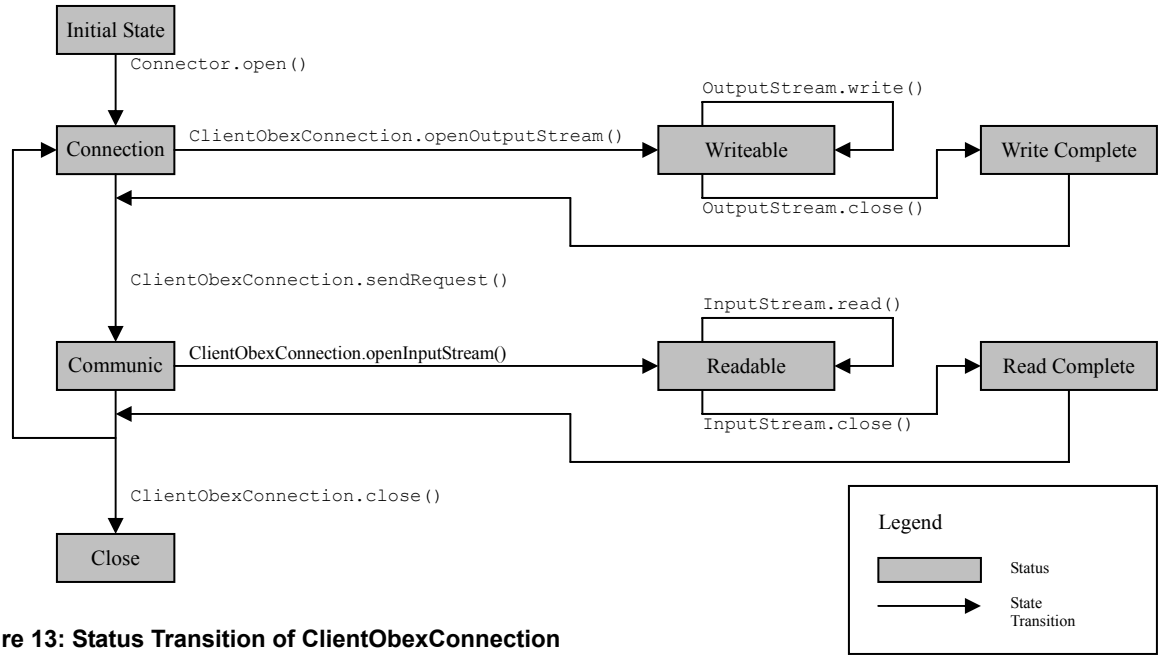


Figure 13: Status Transition of ClientObexConnection

Notes:

- With OBEX external connections, it is not possible to retrieve an I/O stream using the methods `openInputStream()`, `openDataInputStream()`, `openOutputStream()`, or `openDataOutputStream()` directly from the `Connector` class. In order to do this, the application program must first of all acquire an `ClientObexConnection` object using the `Connector.open()` method, and the I/O stream can be retrieved from this object.
- Communication using the infrared port of a mobile phone is carried out automatically by the system when `ClientObexConnection.connect()` is called from the application program.
- It is not possible to use a single operation for an OBEX server to carry out both the transmission of an object and the receipt of objects from the server. OBEX clients may only carry out the transmission of objects using the PUT operation, and the receipt of objects using the GET operation.
- In terms of the open mode for connection objects on OBEX servers, the READ mode corresponds to PUT operations, and the WRITE mode corresponds to GET operations. If it is intended to use both GET and PUT operations during a single connection, please ensure that READ_WRITE mode is used.
- Values for Name headers and Type headers can be specified up to 63 characters, respectively. The Type header value can use ASCII characters only.
- The number of communication links which can be established at any one time from a single i-αppli is one only. Also, if any packet communication is being performed in the background, including that of native functions, no OBEX external connection communication links can be established.
- In a situation where a thread has called either the `connect()` or `sendRequest()` method and is waiting for an action from the OBEX server, if another thread calls the `ClientObexConnection` object's `close()` method, the connection will be severed. Furthermore, it is also possible that the connection may be severed as a result of factors such as time-out occurrence or severing operations by the user.
- For both the GET and PUT operations, the handling of objects up to 100 Kbytes in size is possible. It is not possible to send or receive an object with a size that exceeds this limit.

  [DoJa-3.0]

  Size restrictions for objects that can be sent/received have been relaxed in DoJa-3.0 Profile. In DoJa-2.0 Profile, the size of objects that can be sent/received is limited to 30 KB.

- The operation type, OBEX header, object-data content, and other similar items as set by the application program are valid for a single operation only. In situations where operations are to be continued for a single connection, please ensure that these settings are redone for each operation.
- If the response code returned from an OBEX server contains a value indicating an error (i.e., a code not of the order 0x20), then the i-αppli Runtime Environment will throw an exception.
- Mobile phones which are compatible with DoJa-2.0 Profile feature a Self Mode function which terminates all external communication and allows usage only of functions which do not carry out such communication. Although it will be possible for i-αpplis to launch when the mobile phone has been set to Self Mode by the user, it will not be possible for these i-αpplis to carry out OBEX external connections.
- Stand-by applications in the inactive mode cannot perform OBEX external connections. When a stand-by application in the active mode is performing an OBEX external-connection, the OBEX external-connection will fail if the application attempts to transition into the inactive mode.
- When i-αpplis are carrying out communication using the infrared port, the i-αppli Runtime Environment will display a message which informs of this situation. Similarly, in situations where a communication link cannot be established or where communication is discontinued as a result of user operations or the like, the i-αppli Runtime Environment will display a message which informs of the corresponding situation. It is not possible to prevent the display of these messages.

10.2.2 OBEX Server API

The connection object used by the OBEX server implements the `ServerObexConnection` interface. In order to acquire this connection object, application programs must specify the URL string "`obex:/irserver`" and call the method `Connector.open()`.

The following presents an example of acquisition and usage of the `ServerObexConnection` implementation interface in an application program.

Example: Usage of `ServerObexConnection`

```
ServerObexConnection soc;
try {
          // Acquire the ServerObexConnection instance
          soc = (ServerObexConnection)Connector.open("obex:/irserver",
                                                     Connector.READ_WRITE, true);
          // Await a request for establishment of a communication link from the OBEX client.
          soc.accept();

          // Process the operation request from the OBEX client.
          while (true) {
                   soc.receiveRequest();              // Await an operation request
                   int operation = soc.getOperation(); //Acquire the type for the arriving operation

                   if (operation==ObexConnection.GET) {        // GET operation request
                            // Acquire the name of the requested object.
                            String name = soc.getName();
                            // Set the header data for the object corresponding to the
                            // specified name.
                            soc.setType("text/plain");
                            soc.setTime(System.currentTimeMillis());
                            // Carry out write processing for the object corresponding to the
                            // specified name.
                            OutputStream out = soc.openOutputStream();
                                     :
                            out.close();
                            // Return a status code to the OBEX client signaling a successful finish
                            soc.sendResponse(ObexConnection.SUCCESS);
                   } else if (operation==ObexConnection.PUT) {  // PUT operation request
                            // Acquire data for the transmitted object.
```

```
                        String name = soc.getName();
                        String type = soc.getType();
                        long time = soc.getTime();
                        // Carry out read processing for the transmitted object.
                        InputStream in = soc.openInputStream();
                                :
                        in.close();
                        // Return a status code to the OBEX client signaling a successful finish
                        soc.sendResponse(ObexConnection.SUCCESS);
                } else if (operation==ObexConnection.DISCONNECT) {    // Disconnection request
                        break;
                }
        }

        // Sever the communication link and end communication.
        soc.close();
} catch (IOException ioe) {
        // Exception processing.
        // Processing for IOException and ConnectionException as thrown by API.
                :
}
```

When an application program is using `ServerObexConnection`, it is first of all necessary to call the `accept()` method to await a link establishment request for infrared communication link from the OBEX client (i.e., execution of the OBEX client's `connect()` method). Once a communication link has been established, it will be possible to process multiple operations in sequence.In the above sample code, the processing of operations is carried out repeatedly on a single established link until a disconnection request is received from the OBEX client.

The processing of operations on the OBEX server is carried out in accordance with the following sequence.

1. Receive a request by using the `receiveRequest()` method. This method is blocked until the receipt of a request from the OBEX client.
2. The `getOperation()` method is used to acquire the request operation's type. If `ObexConnection.DISCONNECT` is received as this type, it indicates notification that the OBEX client is disconnecting the communication link. In such a case, please ensure that the OBEX server also carries out connection-object closure.
3. The access method defined by `ObexConnection` is used and the contents of the OBEX request header are acquired. A Name header, Type header, and Time header are contained within the OBEX request headers which may be set using the access method.
4. In the case of a PUT operation, the content of the body for the object sent from the OBEX client is read. This operation is carried out using the input stream retrieved using the connection object's `openInputStream()` or `openDataInputStream()` method.The length of the object body can be acquired using the connection object's `getContentLength() method`.
5. The access method defined by `ObexConnection` is used to set the content of the OBEX response header. A Name header, Type header, and Time header are contained within the OBEX request headers which may be set using the access method.
6. In the case of a GET operation, the body for the object requested by the OBEX client is written. This operation is carried out using the output stream retrieved using the connection object's `openOutputStream()` or `openDataOutputStream()` method.
7. A response is sent to the OBEX client using the `sendResponse()` method. This method's parameters are specified using the OBEX response code as defined by ObexConnection. Note that the OBEX response codes defined by ObexConnection conform with the response code rules from the OBEX specifications. For more details regarding the meaning of the various codes, refer to the OBEX specifications. Generally speaking, when an OBEX server is correctly able to process the request from the OBEX client, the OBEX server application will return `ObexConnection`.SUCCESS to the i-αppli Runtime Environment (`ServerObexConnection.sendResponse()` method) as an OBEX response code.

Please ensure that processing on the OBEX server conforms with the above sequence as a whole. For example, if an attempt is made to acquire the OBEX request header in a situation where no operation processing request has been received, an exception will be generated.

The following diagram illustrates condition transitions for `ServerObexConnection`.

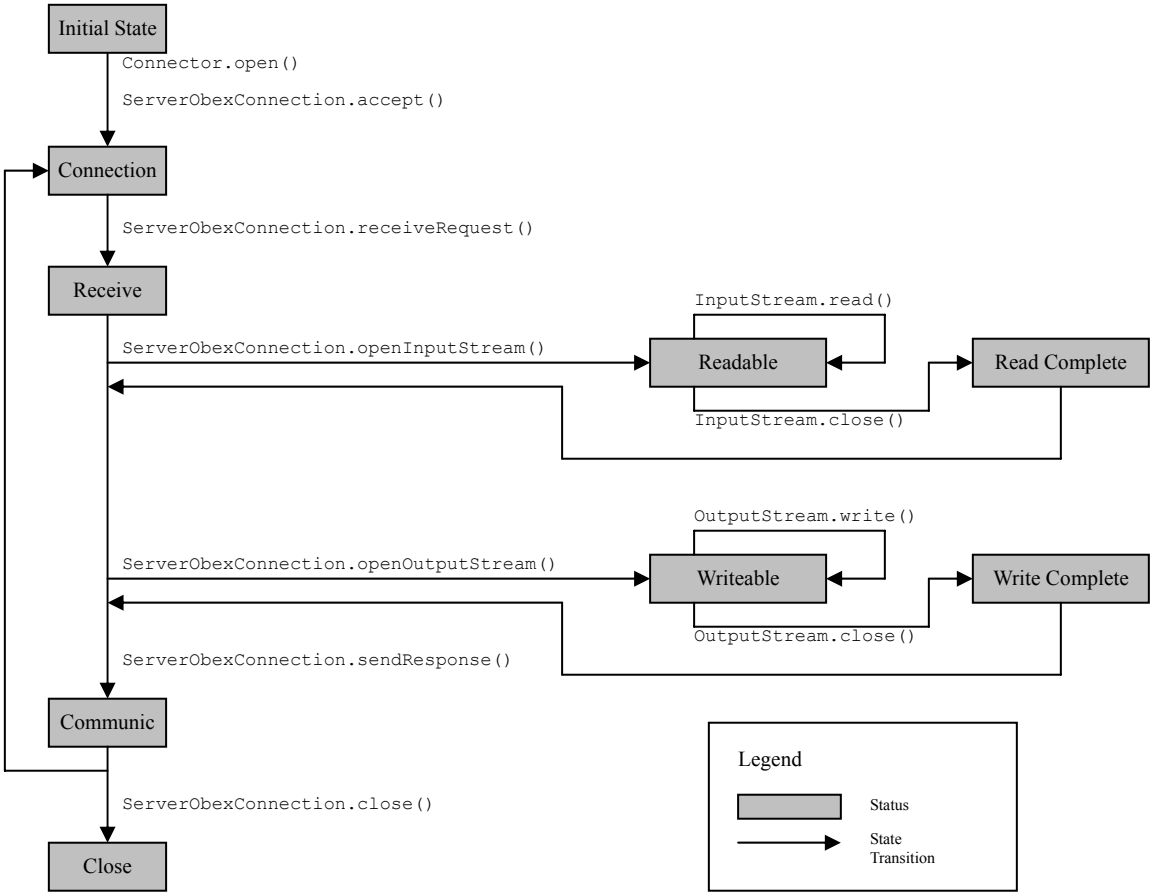


Figure 14: Status Transition of ServerObexConnection

Notes:

- With OBEX external connections, it is not possible to retrieve an I/O stream using the methods `openInputStream()`, `openDataInputStream()`, `openOutputStream()`, or `openDataOutputStream()` directly from the `Connector` class. In order to do this, the application program must first of all acquire an `ClientObexConnection` object using the `Connector.open()` method, and the I/O stream can be retrieved from this object.
- Communication using the infrared port of a mobile phone is carried out automatically by the system when `ServerObexConnection.accept()` is called from the application program.
- It is not possible for an OBEX server to carry out the transmission of an object in response to a PUT operation from the OBEX client. The OBEX server can only carry out object transmission in response to a GET operation.
- In terms of the open mode for connection objects on OBEX clients, the READ mode corresponds to GET operations, and the WRITE mode corresponds to PUT operations. If it is intended to use both GET and PUT operations during a single connection, please ensure that READ_WRITE mode is used.
- Values for Name headers and Type headers can be specified up to 63 characters, respectively. The Type header value can use ASCII characters only.

- The number of communication links which can be established at any one time from a single i-αppli is one only. Also, if any packet communication is being performed in the background, including that of native functions, no OBEX external connection communication links can be established.
- In a situation where a thread has called either the `accept()` or `receiveRequest()` method and is waiting for an action from the OBEX client, if another thread calls the `ServerObexConnection` object's `close()` method, the connection will be severed. Furthermore, it is also possible that the connection may be severed as a result of factors such as time-out occurrence or severing operations by the user.
- For both the GET and PUT operations, the communication of objects up to 100 Kbytes in size is possible. It is not possible to send or receive an object with a size that exceeds this limit.

  [DoJa-3.0]

  Size restrictions for objects that can be sent/received have been relaxed in DoJa-3.0 Profile. In DoJa-2.0 Profile, the size of objects that can be sent/received is limited to 30 KB.
- The OBEX header, object-data content, and other similar items as set by the application program are valid for a single operation only. In situations where operations are to be continued for a single connection, please ensure that these settings are redone for each operation.
- Mobile phones which are compatible with DoJa-2.0 Profile feature a Self Mode function which terminates all external communication and allows usage only of functions which do not carry out such communication. Although it will be possible for i-αpplis to launch when the mobile phone has been set to Self Mode by the user, it will not be possible for these i-αpplis to carry out OBEX external connections.
- Stand-by applications in the inactive mode cannot perform OBEX external connections. When a stand-by application in the active mode is performing an OBEX external-connection, the OBEX external-connection will fail if the application attempts to transition into the inactive mode.
- When i-αpplis are carrying out communication using the infrared port, the i-αppli Runtime Environment will display a message which informs of this situation. Similarly, in situations where a communication link cannot be established or where communication is discontinued as a result of user operations or the like, the i-αppli Runtime Environment will display a message which informs of the corresponding situation. It is not possible to prevent the display of these messages.

10.3 Tips for Connecting an External Device

The following describes the lower layer specifications for an infra-red communication with the i-αppli specifications as well as precautions when establishing an infra-red communication between an i-αppli on a mobile phone and an external device other than a mobile phone.

When developing an i-αppli for establishing infra-red communication between mobile phones, it is not necessary to keep the specifications and precautions in mind. Consider the specifications and precautions for designing the external device side when configuring a system in which an infra-red communication is established between an i-αppli on a mobile phone and an external device.

Note that more details regarding the following content are available in the IrDA specifications.

<Physical conditions>

Communication speed: The maximum communication speed with common support is 115.2 Kbps.

Communication distance: The infrared devices installed on mobile phones conform with the IrMC Low Power Option from IrDA specifications. Therefore, the communication distance is limited to about 20 cm in indoor environment.

<IrLAP Connection Parameters>

Baud Rate: 9600, 38400, and 115200 bps are supported.

Data Size: 64-byte, 128-byte, and 256-byte are supported.

Link Disconnect / Threshold Time: 3 seconds (threshold = 0 seconds) and 8 seconds (threshold = 3 seconds) are supported.

<IrLMP Device Names and Nicknames>

A character string as shown below is set for the device name and device nickname:

`DoCoMo/`<device-name>`/j` (Example: `DoCoMo/X505i/j`)

A name identical to that included in the user agent is set for <device-name>. By referring to this name from an external device, it is possible to determine the model of a mobile phone that is currently establishing the communication.

<IrLMP Service Hints>

The below flags are set to "on" in the service hints bit-map.

```
Byte1: Bit1  PDA/Palmtop
Byte1: Bit7  Extension
Byte2: Bit8  Telephony
Byte2: Bit13  IrOBEX
```

[DoJa-3.0]

In DoJa-3.0 Profile, a 115200 bps Baud Rate has been added to the IrLAP connection parameters. Note that the Baud Rates supported in common with DoJa-2.0 Profile are 9600 bps and 38400 bps; connections at 115200 bps are not guaranteed.

The details of IrLMP service hint settings in DoJa-2.0 Profile may vary depending on the manufacturer.

Notes:

- Of the operations set by OBEX, the i-αppli Runtime Environment does not support SetPath operations for either the OBEX client or OBEX server. Furthermore, the ABORT operation is not supported for the OBEX client.
- OBEX authentication is not supported within the i-αppli Runtime Environment.
- Five different types of headers may be received by the OBEX client and the OBEX server within the i-αppli Runtime Environment, and these are Name, Type, Time, Body, and End of Body. OBEX headers other than these cannot be used. Furthermore, please ensure that the object body (i.e., Body header and End of Body header) is sent after the other headers.
- The infrared communication functionality of mobile phones has been certified for mutual IrDA connection. However, this does not necessarily guarantee that connection will be possible with all external devices conforming to the IrDA specifications. Depending on the environment to be used, and hardware and software characteristics of the other party’s device, a connection may not be established.

10.4 IrSimple

On DoJa-5.1 Profile or later mobile phones, the ability to use IrSimple from an i-appli is supported as an optional feature. IrSimple can be used to achieve high speed, large capacity infrared communications, but you must take into account that the use of some functions are limited.

The following section discusses how to use IrSimple from an i-appli and some notices regarding its use.

In order to comply with mobile phone models which support IrSimple, the following methods and fields were added to DoJa-5.1 Profile's OBEX External Connection API.

- `ClientObexConnection.connect(int mode)`

  Specifies a communications mode and establishes an infrared communications link from the OBEX client. ("Communications mode" refers to the `ObexConnection COMM_MODE` constants.)The specification you are required to make to use IrSimple is made via the communications mode parameter for setting up the `connect()` method.

- `ObexConnection.getCommMode()`

  Retrieves the current communications mode from a connection object that has established a link.

- `ObexConnection.COMM_MODE_IRDA`

  A communications mode constant. Represents backwards-compatible IrDA communication.

- `ObexConnection.COMM_MODE_IRSIMPLE_INTERACTIVE`

  A communications mode constant. Represents two-way IrSimple communication. Under this connection mode you can achieve highly reliable communication via the confirmation of delivery of data transferred between the OBEX server and the OBEX client or via retransmission control, just like traditional IrDA communication.

- `ObexConnection.COMM_MODE_IRSIMPLE_UNILATERALLY`

  A communications mode constant. Represents one-way IrSimple communication. Under this connection mode data is only transmitted from the OBEX client and mutual client/server delivery confirmation and retransmission control is not performed. As a result, this mode may not have the reliability of two-way communication but can achieve faster, more efficient communication.

When using IrSimple, you must specify either `ObexConnection.COMM_MODE_IRSIMPLE_INTERACTIVE` or `ObexConnection.COMM_MODE_IRSIMPLE_UNILATERALLY` as a parameter to the `connect(int)` method when establishing a communication link from the OBEX client side i-appli.

When using IrSimple, the following points differ compared to traditional IrDA.

- OBEX clients cannot use GET operations.When IrSimple is used, the OBEX client may only use PUT operations. Furthermore, when using IrSimple one-way communication, only one operation may be executed per established communication link.
- The limitation on the maximum transmittable data size for an OBEX client's single PUT operation has been relaxed to 2 MB.

Please note that if the OBEX server does not support IrSimple and the OBEX client attempts to use IrSimple, the following resulting operations will occur.

1. If the OBEX client attempts to connect via IrSimple two-way communication

If an OBEX client attempts to connect to an OBEX server that does not support IrSimple via IrSimple two-way communication, a backwards-compatible IrDA communication link will be established instead of IrSimple.

2. If the OBEX client attempts to connect via IrSimple one-way communication

Under IrSimple one-way communication, data will be sent one-way from the client successfully without the occurrence of any exceptions regardless of whether or not the OBEX server supports IrSimple or if there are any IrDA devices on the other side. If the OBEX server on the other side does not support IrSimple, the OBEX client will successfully be able to complete its data transmission but the OBEX server will not respond to the connection request and will remain in its state of waiting for a transmission.

Notes:

- When using IrSimple the URL specified to the `Connector.open()` method is the same as that specified when conducting standard IrDA communications ("`obex:/irclient`" or "`obex:/irserver`").
- If the `ClientObexConnection.connect(int)` method or the `ObexConnection.getCommMode()` method is used on a mobile phone which does not support IrSimple, an `UnsupportedOperationException` will be thrown.
- Under IrSimple one-way communication, the OBEX server does not respond to the OBEX client's request due to the fact that the OBEX client simply communicates with the OBEX server in a one-way fashion. However, in order to process IrSimple one-way communications in an i-appli OBEX server, always use the `ServerObexConnection.sendResponse()` method to respond after you have received a request via the `ServerObexConnection.receiveRequest()` method. (See Section 10.2.2 for more information on how to use these methods.)

[DoJa-5.1]

The API for IrSimple was newly added with DoJa-5.1 Profile.

Chapter 11
Application Linking

In DoJa-2.0 Profile, linked launching has been supported between an i-αppli and native applications on a mobile phone such as a browser and a mailer. Furthermore, in DoJa-3.0 Profile and later, application linking has been enhanced, for example, by supporting the calling of native functions, including bookmark management function and image data management function, from an i-αppli, as well as by expanding the range of linked launching (such as between an i-αppli and an i-αppli). By utilizing these functions, it is possible to provide service that fully utilizes not only the functions of an i-αppli, but also the functions of a mobile phone by an i-αppli.

Note that some parts of the application linking functions are included in the range of a Trusted i-αppli because of the characteristics of these functions. This document does not describe the functions that are included in a Trusted i-αppli. The functions described in this document can also be used in general i-αpplis other than a Trusted i-αppli.

The following describes the details of the application linking functions that are supported by each profile.

Application linking functions

Overview of the linking functions	DoJa-2.0 and later	DoJa-3.0 and later	DoJa-4.1 and later	DoJa-5.0 and later
Launch i-αppli from browser	○	○	○	○
Launch browser from i-αppli	○	○	○	○
Launch i-αppli from mailer	○	○	○	○
Launch i-αppli from an external device (an infra-red port)	○	○	○	○
Launch i-αppli from i-αppli	×	○	○	○
Launch i-αppli update function (JAM) from i-αppli	×	○	○	○

Call native function from i-αppli

Overview of the linking functions	DoJa-2.0 and later	DoJa-3.0 and later	DoJa-4.1 and later	DoJa-5.0 and later
Calling communication function (Outgoing phone call)	○	○	○	○
Calling communication function (see unique identification information)	△(*1)	○	○	○
Call phone book management function (add new phone book entry and phone book group)	×	○	○	○
Call bookmark management function (add new bookmark)	×	○	○	○
Call schedule management function (add a new schedule item, launch the Scheduler from an i-appli)	×	○(*3)	○(*3)	○(*3)
Call image data management function (add new image, read the image selection, and read the image with a specified ID)	×	○	○	○
Call camera function (shoot an image and acquire camera image)	△(*2)	○	○	○
Call movie data management function (add new movie data (iMotion))	×	×	○	○

(*1) The reference of unique identification information is added to i-αppli standard APIs in DoJa-2.1 Profile.

(*2) The calling of the camera function is supported by some models as an i-αppli optional API in DoJa-2.0 Profile. In the DoJa-3.0 Profile, those functions were partially stregthened and moved into the i-αppli standard API.

(*3) Regarding calling schedule management functions, launching the Scheduler from an i-appli was part of the i-appli Optional API up until DoJa-4.1 Profile. This function was moved to the i-appli standard API from DoJa-5.0 Profile.

This chapter describes how to use the application linking functions and various restrictions.

11.1 Browser Linked Launching

In browser linked launching, it is possible to launch an i-αppli from a browser, or to launch a browser from an i-αppli. Note that, for launching an i-αppli from a browser, the i-αppli to be launched must be installed in the mobile phone beforehand.

11.1.1 Launching an i-αppli from the Browser

To launch an i-αppli from a browser, a developer must perform the following tasks.

(1) Create an HTML file for launching the i-αppli on an accessible website via the Internet, and describe an i-αppli launch tag in the file. Depending on the descriptions of the i-αppli launch tag, the i-αppli that will be launched on a mobile phone when selecting the tag is determined.

(2) In the `LaunchByBrowser` key of the ADF, declare that the i-αppli permits launching from the URL of the HTML file created in (1). It is not possible to launch the i-αppli from an HTML file with a URL that does not receive permission from the LaunchByBrowser key. Note that the URL specified by the `LaunchByBrowser` key and that of an HTML file are compared by a forward match, and if the contents specified by the `LaunchByBrowser` key are all included in the URL of the HTML, launch permission is granted to the HTML file.

To launch an i-αppli from a browser, display the HTML file for launching an i-αppli in the browser. The i-αppli launch tag is displayed in a link format. When a user selects this link, the mobile phone launches the corresponding i-αppli.

The following describes a descriptive example of a tag in an HTML file for launching an i-αppli.

```
<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-appli”>
</OBJECT>
To launch an i-αppli,
click <A ista=”#application.declaration” href=”notapplicable.html”> here </A>.
```

The following describes tags that are used for launching an i-αppli.

- The `A` tag and the `ista` attribute

  The `A` tag is used to refer to the `OBJECT` tag that corresponds to the i-αppli to be launched. When the user selects this A tag, the i-αppli that has been installed in the mobile phone is launched.For the ista attribute of the A tag, specify the name that is specified in the id attribute of the `OBJECT` tag. Which i-αppli is launched on the mobile phone depends on the ADF URL specified in the `OBJECT` tag that is described later.

  The `ista` attribute is newly added to DoJa-2.0 Profile and is not parsed in mobile phones that support DoJa-1.0 Profile.

  Note that for the `A` tag that has the `ista` attribute, be sure to specify the `href` attribute as a pair. Furthermore, for details on the A tag and attributes (`ijam` attribute and `ilet` attribute) that are used for downloading an application, see Chapter 16.

- The `OBJECT` tag

The OBJECT tag is used to identify an ADF that supports an i-αppli.For the id attribute of the OBJECT tag, specify the name (a unique name in an HTML file), which is referenced in the ista attribute of the A tag. Furthermore, for the data attribute, specify an URL that indicates the location of the ADF.When a mobile phone downloads an i-αppli, it stores the URL of the ADF that corresponds to the i-αppli.For launching an i-αppli from a browser, an i-αppli that refers to the same ADF as the URL specified in the data attribute is to be launched.For the type attribute, specify the content type of data (in this case, ADF) indicated by the data attribute. The content type of the ADF is "application/x-jam".

For the inside of the OBJECT tag, the following PARAM tag can be specified.

- The PARAM tag

  The PARAM tag is used to specify a parameter that can be obtained by the IApplication.getParameter() method when launching an i-αppli from a browser.

  A maximum of 16 PARAM tags can be included in the OBJECT tag. Also, the total length of the values of the name and value attributes of all the PARAM tags specified in one OBJECT tag is limited up to 255 bytes. For the values of name and value attributes, SJIS Japanese text can be specified respectively.

The following diagram describes the relationship between an i-αppli that has been downloaded in a mobile phone and the contents of an HTML file for launching an i-αppli.

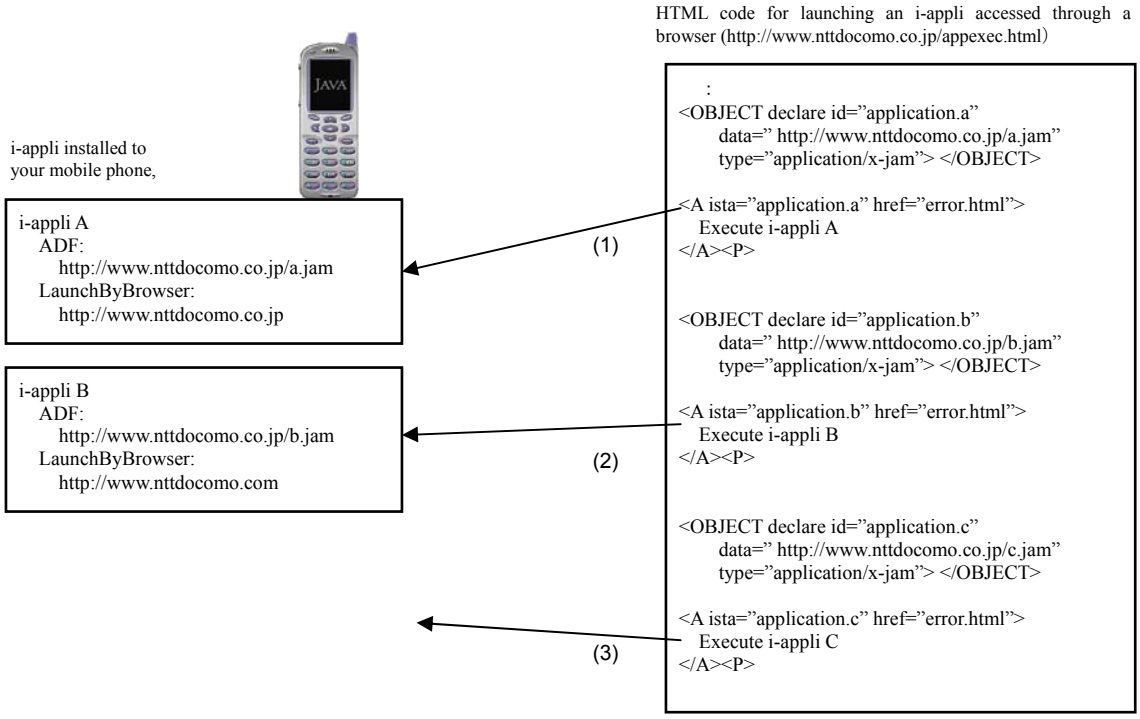


Figure 15: Relationship between a Downloaded i-αppli and the Content of an HTML File for Launching an i-αppli

Notes:

- The mobile phone user can configure the mobile phone to prohibit the launch of i-αpplis via a browser. Furthermore, before launching an i-αppli from a browser, the i-αppli Runtime Environment asks a user if it is okay to launch an i-αppli. If the user's permission is not received through this mechanism, the i-αppli is not launched.
- When the operation of an i-αppli that has been launched from a browser ends (by `IApplication.terminate()`), the browser status before launching the i-αppli is restored. Note that if a user forces the termination of an i-αppli that has been launched from a browser, and the i-αppli that has been launched from a browser launches or calls the function of other native applications by using the application linking functions, the status that appears after the i-αppli terminates varies, depending on the manufacturer.

[DoJa-5.0]

In DoJa-5.0 Profile or later on mobile phones which support ToruCa version 2.0 format, you can use the tags discussed here in this chapter in the HTML content section of ToruCa (Details) to launch an i-appli from the ToruCa Viewer. In this case, be sure not to specify a relative URL for the ADF URL specified in the `data` attribute for the `OBJECT` tag placed in the HTLM content section of ToruCa (Details).

To allow an i-appli to be launched from the ToruCa viewer, the `LaunchByToruCa` key must be specified instead of the `LaunchByBrowser` key in that i-appli's ADF. The URL for retrieving the ToruCa (Details) in the ToruCa data is supported, but if the ToruCa (Details) retrieval URL forward matches the `LaunchByToruCa` key, that particular i-appli can be launched from that ToruCa.

In the same way that the i-appli launch type for an i-appli launched from the browser (i.e., the launch type obtained via the `IApplication.getLaunchType()` method) is `IApplication.LAUNCHED_FROM_BROWSE`R, the i-appli launch type when an i-appli is launched from the ToruCa viewer is `IApplication.LAUNCHED_FROM_TORUCA`.

[DoJa-5.1] (FOMA 906i or newer)

When the `ijam` attribute is also specified in addition to the `ista` attribute in an `A` tag used for i-appli linked launching, if an update is found for that particular i-appli when it is about to be launched, the i-appli update process will be conducted. In that situation, the i-appli launched after the i-appli update process is complete will have a launch type of either `IApplication.LAUNCHED_FROM_BROWSER` (if launched from a browser) or `IApplication.LAUNCHED_FROM_TORUC`A (if launched from the ToruCa Viewer) on FOMA 906i series or newer mobile phones. However, on older mobile phone models the launch type of the i-appli launched after the update process will vary depending on the manufacturer.

Note that even if both the ista attribute and the ijam attribute are specified, if the target i-appli is not installed on the mobile phone the i-appli launched after it is downloaded will have a launch type of `IApplication.LAUNCHED_AFTER_DOWNLOAD`.

11.1.2 Launching a Browser from an i-appli

Use the `IApplication.launch() method` to launch a browser from an i-αppli. To launch a browser, set `IApplication.LAUNCH_BROWSER` in the first argument (type of an application to be launched) of this method and set the URL to be connected (supporting the http or https scheme) in the first element of the second argument (String array indicating the launching argument). A maximum of 255 bytes can be specified for a URL.

Note that an i-αppli for launching a browser must declare, by using the `UseBrowser` key of the ADF, that it will launch a browser (set the value "`launch`").An i-αppli that does not make this declaration cannot launch a browser.

Calling the `IApplication.launch()` enters the i-αppli to the suspended state. After that, a browser is launched by satisfying the following two points.

- Packet communication is possible (not out of service area or in a state unable to perform packet communication).
- Before launching the browser, the user agrees with a launch confirmation requested by the i-αppli Runtime Environment.

If either one of these conditions is not satisfied, the i-αppli resumes and the `IApplication.launch()` method returns to the caller without throwing an exception. Also, when both conditions are satisfied and a browser is launched, the i-αppli terminates.

[DoJa-5.0]

In DoJa-5.0 Profile and later, an API is provided that allows the developer to instruct an i-appli not to close when the Browser is launched from that i-appli, but instead enter a suspended state. However, support for this function is optional, and in some cases may not be provided depending on the manufacturer.

To use this function simply follow the procedure above but for the first parameter of the `launch()` method, specify `IApplication.LAUNCH_BROWSER_SUSPEND` instead of `IApplication.LAUNCH_BROWSER`. Using this method to launch the Browser, the i-appli will stay suspended even after the Browser is launched, and will resume operation once the Browser is closed.

Notes:

- In the i-αppli Runtime Environment of profiles proceeding DoJa-4.0 Profile, when a stand-by application has been launched in stand-by, it cannot use the `MApplication.launch()` method to launch the browser under any circumstance. An exception occurs when a stand-by application that has been launched in stand-by attempts to launch a browser.

  Compared to this, in the i-αppli Runtime Environment of DoJa-4.1 Profile and later, even though a stand-by application is launched in stand-by, if it is in the active mode, it can launch a browser by the `MApplication.launch()` method.
- When a browser that has been launched from an i-αppli is terminated by user operation, the stand-by status is restored.
- Unlike in the case of HTTP communication using `HttpConnection`, for launching a browser from an i-αppli, it is possible to access sites other than the site from which the i-αppli is downloaded. Note that the request method is limited to GET and the contents of the URL are presented to the user when the i-αppli Runtime Environment asks the user about the browser launching.

11.2 Mail Linked Launching

In mail-linked launching, it is possible to launch an i-αppli from a mail that is transmitted to a user. Note that an i-αppli to be launched must be installed in the mobile phone beforehand.

11.2.1 Launching an i-αppli from a Mail

To launch an i-αppli from a mail, a developer must perform the following tasks.

(1) In the `LaunchByMail` key of the ADF, declare that the corresponding i-αppli permits launching from a mail received from a specific sender.It is not possible to launch the i-αppli from a mail from a mail sender that does not receive permission from the `LaunchByMail` key. Note that the mail address specified by the `LaunchByMail` key and that of the mail sender are compared by a reverse match, and if the contents specified by the `LaunchByMail` key are all included in the address of the mail sender, launch permission is granted to the mail.

(2) Mail is sent from the mail address that has been granted launch permission to the mobile phone of the user that downloads an i-αppli for which (1) is specified. In this mail, describe the information for determining the i-αppli to be launched and the launch parameters of the i-αppli (i-αppli launch information).

When the mail in which the i-αppli launch information is described is viewed in a mailer, a link for launching the i-αppli, etc. is displayed. When a user selects this link, the mobile phone launches the corresponding i-αppli.

The following describes a descriptive example of a mail that includes the i-αppli launch information.

```
・・・・・・・・
・・・・・・・・・・
・・・・・
(Until here is a mail body viewable from the mailer)
--B:A
TEXT = "i-appli A"
ADF="http://www.nttdocomo.co.jp/java/abc.jam"
"Param1"="i-mode"
“Param2” = ”i-appli”
```

Note: Each line is terminated by <CR><LF> (0x0d0a)

The i-αppli launch information is written as follows:

- `--B:A`

  "`--B:A`" is an identifier marking the boundary between the mail's body and i-αppli launch information. The contents after a line containing only this identifier (letters must be capital) are recognized as the i-αppli launch information. The contents of the i-αppli launch information cannot be directly viewed when a user views the mail. Also, the i-αppli launch information is not quoted when replying to or forwarding the mail.

- `TEXT` key

  For the TEXT key, set the character string that is displayed in the link for launching the i-αppli. In the previous descriptive example, a character string “i-αppli A” is displayed as a link when viewing the mail, and when the user selects the link, the i-αppli is launched.

- `ADF` key

  For the ADF key, specify the URL of the ADF that corresponds to the i-αppli to be launched. When a mobile phone downloads an i-αppli, it stores the URL of the ADF that corresponds to the i-αppli. In mail linking, the i-αppli that refers to the same ADF as the URL of the ADF key described in a mail is launched.

- Parameter specification key

  A developer can define parameters to be passed to an i-αppli by enclosing the name and value of a key with double quotations. Parameters specified like this can be acquired by the `IApplication.getParameter()` method (equivalent to the `PARAM` tag for launching an i-αppli from a browser).

  A maximum of 16 parameter specification keys can be included in one mail.Also, the total length of the names and values of all the parameter specification keys that are used in one mail is limited up to 255 bytes. For the name and value of the key, Japanese text can be specified. When Japanese text is used for a parameter, the limit of the length applies to the SJIS representation.

The following diagram describes the relationship between an i-αppli that has been downloaded on a mobile phone and the contents of the i-αppli launch information described in a mail.

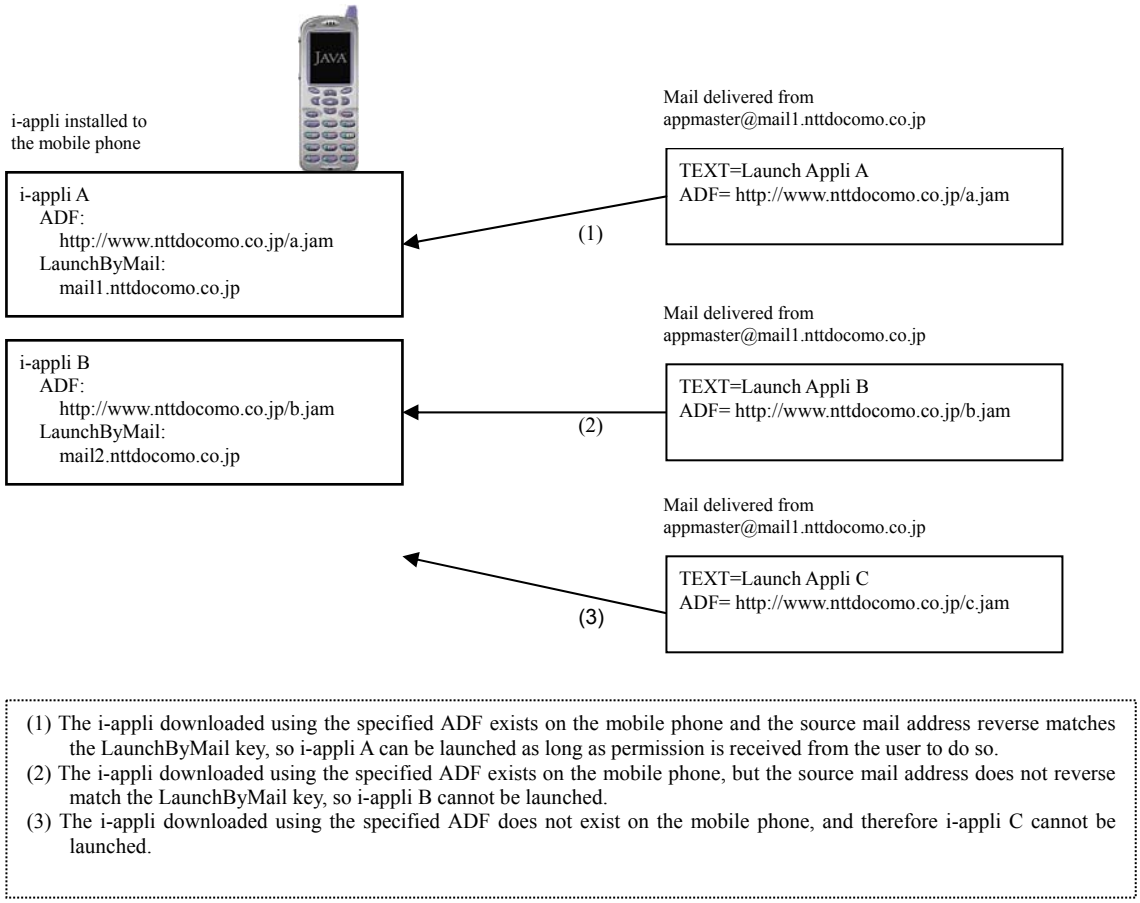


Figure 16: Relationship between a Downloaded i-αppli and the Descriptions of the i-αppli Launch Information

Notes:

- The mobile phone user can configure the mobile phone to prohibit the launch of i-αppli s via mail. Furthermore, before launching an i-αppli from a mail, the i-αppli Runtime Environment asks a user if it is okay to launch the i-αppli. If the user's permission is not received through this mechanism, the i-αppli is not launched.
- When the operation of an i-αppli that has been launched from a mail terminates (by IApplication.terminate()), the mailer status before launching the i-αppli (status in which the original mail is displayed) is restored. Note that if a user forces the termination of an i-αppli that has been launched from a mail, and the i-αppli that has been launched from the mail launches or calls a function of other native applications by using the application linking functions, the status that appears after the i-αppli terminates varies, depending on the manufacturer.
- It is not possible to attach both the i-αppli launch information and other attachments (data added to the body of the mail separated by boundary markers, attached melodies and the like) to a mail simultaneously. If both types of data are attached, they are both considered to be invalid.
- The size of the mail, including the i-αppli launch information must not exceed the limits for i-mode mail. As this function does not support multi-part i-mode mail, the size of the mail must be kept to a size below that triggering a split of the mail.

11.3 External Device Linked Launching

In external device linked-launching, it is possible to launch an i-αppli on a mobile phone by instructing the mobile phone to launch it via an infrared port from an external device. Note that an i-αppli to be launched must be installed in the mobile phone beforehand.

11.3.1 Launching an i-αppli from an External Device

To launch an i-αppli from an external device, a developer must perform the following tasks.

(1) Create and place an OBEX application for instructing the mobile phone to launch an i-αppli via an infra-red port on an external device. For requirements of infrared communication on the external device side, see Section 10.3. An instruction for launching an i-αppli from an external device is provided by sending the `vTrigger` object, which is described later, to the mobile phone.

(2) In the `AllowPushBy` key of the ADF that corresponds to an i-αppli to be launched from an external device, declare that the i-αppli permits launching from an external device that sends the `vTrigger` object, which includes the specific launch command.If the launch command declared in AllowPushBy and the launch command sent by the external device do not match completely, that i-αppli cannot be launched. The `AllowPushBy` key is specified as described below.See Section 15.5.1 for details on how to write an ADF.

```
AllowPushBy = Irda:<launch command>
```

Communication of the infrared port on a mobile phone is carried out by menu or button operation by the user. When the user faces the external device and the infra-red port on a mobile phone with each other in this state, and sends and receives the vTrigger object, the i-αppli is launched.

`vTrigger` is an object format that is defined to support external device linked-launching in a i-αppli-supported mobile phone. `vTrigger` is text data that is represented as a group of multiple properties (a pair of the attribute name and value). By sending the `vTrigger` object to a mobile phone via an infra-red port, it is possible to instruct an i-αppli on the mobile phone to launch.

The following describes the `vTrigger` formats.

```
BEGIN:VTRG
VERSION:<vTrigger format version>
ADFURL:<ADF URL of the i-αppli to be launched>;
COMMAND:<launch command>;
<Parameter>
END:VTRG
```

* Set <CR><LF>（0x0d0a）at the end of the line.

The following describes the notation.

- `BEGIN:VTRG`

  Indicates the start of the `vTrigger` object.

- `VERSION:`<vTrigger format version>

  Specify the format version of the `vTrigger` object that is being used. In all previous and current profiles, specify "1.0" as the `vTrigger` format version.

- `ADFURL:`<ADF URL of the application to be launched>

  Specify the URL of the ADF for the i-appli you want to launch via this `vTrigger` object in ASCII format (maximum 255 bytes).

- COMMAND:<launch command>

  The launch command, in ASCII format (maximum 250 bytes). If the launch command specified here does not exactly match the launch command for which launch permission is assigned by the AllowPushBy key of the i-αppli's ADF (the string in the AllowPushBy key following "irda:"), the i-αppli cannot be launched.

- <Parameter>

  Specify parameters to pass to the i-αppli to be launched (optional). One parameter has the following format:

  PARAM [;<encode specification>][;<character set specification>]:<parameter name>;<parameter value>

  * [ ] indicates that the parts enclosed in [ ] can be omitted. Note that you may switch the locations of the encode specification and character set specification.

  The parameter name is used to specify a parameter that can be obtained by the i-αppli on launch via the IApplication.getParameter() method. A maximum of 16 numbers of <parameter> can be specified for one vTrigger object.Note, however, that the total length of all the names and values of <parameter> is restricted to 255 bytes.

  The encode and character set are specified as described below.

  - Encoding specification

    Specify in the format of ENCODE=<encoding name>. For the encoding name, either "QUOTED-PRINTABLE" or "BASE64" can be specified. If one of these is not specified, it is regarded that no encoding is specified.

  - Character set specification

    Specify in the format of CHARSET=<character set name>. In all previous and current files, the valid character set name is only "SHIFT_JIS". Also, even when a character set is not specified, it is regarded that "SHIFT_JIS" is specified.

  Note that for the vTrigger object, the contents of the body must be described in 7-bit code. If Japanese (SHIFT_JIS) is specified for the name or value of the parameter, be sure to encode in the QUOTED-PRINTABLE or BASE64 format. The encoded Japanese parameter is automatically decoded in the i-αppli Runtime Environment according to the specified encoding type, converted to String of Java and passed to the IApplication.getParameter() method.

- END:VTRG

  Indicates the end of the vTrigger object.

The following is an example of how to code a vTrigger object.

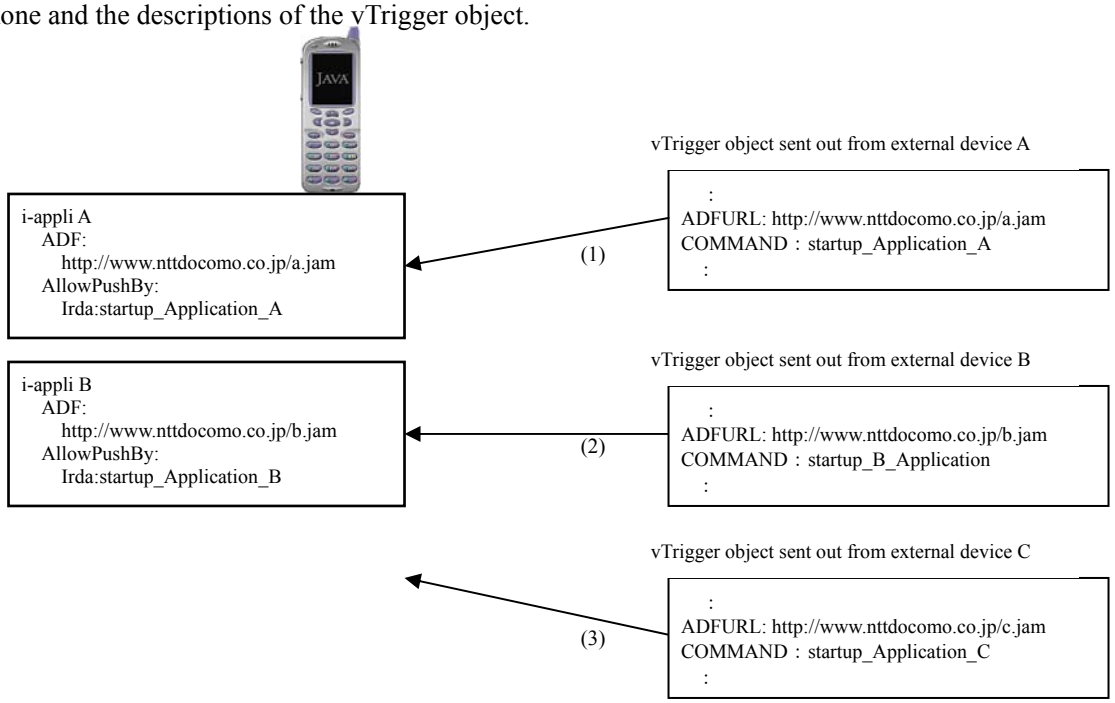
```
BEGIN:VTRG
VERSION:1.0
ADFURL:http://www.nttdocomo.co.jp/java/a.jam
COMMAND:startup_Application_A
PARAM:Param1;i-mode
PARAM;ENCODING=BASE64;CHARSET=SHIFT_JIS:Param2;gomDQYN2g4o=
END:VTRG
```

Sending the vTrigger object in the OBEX layer intends to set the contents of the vTrigger object in the Body header and set it by the PUT operation. The mobile phone determines whether the i-αppli will be launched or not in response to the i-αppli launch request from the external device, and returns the following OBEX response code.

Response code	Requirements of OBEX	Requirements of external device linking
0xA0	OK, Success	It is possible to launch the specified i-αppli.
0xC4	Not Found	The specified i-αppli does not exist.
0xE1	Database Locked	Launching the specified i-αppli is prohibited from an external device, or the launch command of the AllowPushBy key does not match the launch command received by the vTrigger object.

In cases where the response code 0xA0 is returned to the external device, the i-αppli is not launched on the mobile phone. The i-αppli is launched after the mobile phone returns the response code, and the infrared communication ends.

The following diagram describes the relationship between an i-αppli that has been downloaded in a mobile phone and the descriptions of the vTrigger object.



The i-appli downloaded using the specified ADF file exists on the mobile phone and the value of COMMAND exactly matches the AllowPushBy key, so i-appli A can be launched.
By holding their mobile phone over the external device's infrared port, the user agrees to launch the i-appli. Therefore, in this case the mobile phone (the i-appli Runtime Environment) does not perform launch verification for the i-appli.
(2) The i-appli downloaded using the specified ADF exists on the mobile phone, but the COMMAND value does not exactly match the AllowPushBy key, so i-appli B cannot be launched.
(3) The i-appli downloaded using the specified ADF does not exist on the mobile phone, and therefore i-appli C cannot be launched.

Figure 17: Relationship between a Downloaded i-αppli and the Descriptions of the vTrigger object

Notes:

- The mobile phone user can configure the mobile phone to prohibit the launch of i-αpplis via an external device. If launching from an external device is prohibited, even though the vTrigger object is received, an i-αppli is not launched.

- To receive a launch command (the vTrigger object) from an external device, a user must establish communication to the infrared port of the mobile phone beforehand. The method of establishing communication to an infrared port differs according to the manufacturer.
- When the operation of an i-αppli that has been launched from an external device terminates, the stand-by status is restored. Note that when a user forces the termination of an i-αppli that has been launched from an external device, the status that appears after the i-αppli terminates varies, depending on the manufacturer.
- When sending the vTrigger object from an external device to a mobile phone, set "`.vtg`" for the end (extension) of the value of the Name header of the vTrigger object. When the extension of the name of the received object is "`.vtg`", a mobile phone recognizes the object as a vTrigger object.

11.4 i-αppli Linked Launching

In i-αppli linked-launching, it is possible to launch other i-αpplis by calling an API from the application program. For i-αppli linked-launching, there are two types of modes as described below.

- Link Mode

  In link mode, when an i-αppli launches another i-αppli, the i-αppli to be launched can be directly specified by the URL of the ADF. Furthermore, the original i-αppli can pass parameters to the i-αppli to be launched. Note that in link mode, the original i-αppli and the i-αppli to be launched must be downloaded from the same host.

- Launcher Mode

  Unlike in link mode, even though the host from which an i-αppli has been downloaded is different from that of the original i-αppli, that i-αppli can be launched in launcher mode. Note that when launching an i-αppli, do not use the URL of the ADF, but use an entry ID, which corresponds to each i-αppli downloaded in a mobile phone and specify the i-αppli to be launched. Therefore, when launching an i-αppli in launcher mode, it is necessary to acquire the entry ID of the i-αppli to be launched beforehand. An entry ID of an i-αppli is acquired by calling an API and obtaining user confirmation. In launcher mode, the original i-αppli cannot pass parameters to the i-αppli to be launched.

Note that an i-αppli to be launched must be installed in the mobile phone beforehand.

11.4.1 i-αppli Linked Launching in Link Mode

Use the `IApplication.launch()` method to launch an i-αppli in link mode. When launching an i-αppli in link mode, set `IApplication.LAUNCH_IAPPLI` in the first argument (type of an application to be launched) of this method and set the URL of the ADF for an i-αppli to be launched in the first element of the second argument (`String` array indicating the launching argument).

When passing parameters to the i-αppli to be launched, use the second or later element (the array subscript is 1 or more) of the `String` array of the second argument. The parameters passed from the original i-αppli can be acquired by using the `IApplication.getParameter()` method on the side of the i-αppli to be launched. One parameter consists of two array elements that indicate a variable name and a value. The following describes an example of transferring parameters from the original i-αppli and to the i-αppli to be launched.

*Set parameters in the i-appli that is the

```
String[] args = new String[5];
// Set launch target i-appli's ADF URL
args[0] = "http://www.nttdocomo.co.jp/app.jam"

// Parameter settings
args[1] = "Param1";  // Name of first parameter
args[2] = "i-mode";  // Value of second parameter
args[3] = "Param2";  // Name of second parameter
args[4] = " i -appli"; // Value of second parameter

// i-appli linked launch (link mode)
IApplication.getCurrentApp().launch(
         IApplication.LAUNCH_IAPPLI, args);
```

Launc

*Acquire parameters in the i-appli that is the

```
String param1, param2;

// param1 contains "i-mode"
param1 = IApplication.getCurrentApp().
         getParameter("Param1");

// param2 contains "i-appli"
param2 = IApplication.getCurrentApp().
         getParameter("Param2");
```

Figure 18: Transferring Parameters from the Original i-αppli to the i-αppli to be Launched in Link Mode

A maximum of 16 parameters (16 sets of combinations of parameter names and values) can be specified. Also, the total length of parameter names and values to be specifed is limited up to 20480 bytes (the number of bytes is evaluated in the default encoding).

Note that an i-αppli that performs an i-αppli linked launching must declare, by using the `LaunchApp` key of the ADF, that it will perform an i-αppli linked launching (set the value "`yes`"). An i-αppli that does not make this declaration cannot perform an i-αppli linked-launching.

Notes:

- The host names of the ADF URLs of the original i-αppli and the i-αppli to be launched must be the same.Also, it is not possible to launch an i-αppli that has the same ADF URL as the original i-αppli.
- This function cannot be used from i-αpplis of immediate launch after download, or from inactive stand-by applications.
- When the i-appli to be launched is launched, the i-appli it was launched from closes.
- If the operation of the i-αppli to be launched terminates, the mobile phone returns to the stand-by state. Note that when a user forces the termination of the i-αppli to be launched, the status that appears after the i-αppli terminates varies, depending on the manufacturer.
- Even if the i-αppli to be launched is a stand-by application, when it is launched by i-αppli linked-launching, the launch mode is not Standby Launch, but Normal Launch.
- If the original i-αppli is a Trusted i-αppli, the i-αppli to be launched must also be a Trusted i-αppli.

[DoJa-5.0]

In profiles earlier than DoJa-4.x Profile, when an i-appli was link launched the i-appli it was launched from would enter a suspended state and a dialog to confirm the launch of the i-appli would be displayed by the system. However, from DoJa-5.0 Profile onward, this launch confirmation is not conducted.

[DoJa-5.1]

In regards to i-appli linked launching in link mode, the length restrictions for parameters that can be passed from the launch source i-appli to the launch target i-appli have been relaxed in DoJa-5.1 Profile.In DoJa-5.1 Profile and later, parameters which have a combined parameter name and value length of 20480 bytes or less can be passed to the launch target i-appli, whereas in profiles DoJa-5.0 Profile and earlier this was limited to only 255 bytes.

11.4.2 i-αppli Linked Launching in Launcher Mode

When performing i-αppli linked-launching in launcher mode, a developer must implement the following two processes in the original i-αppli (hereinafter referred to as launcher i-αppli).

Note that, as well as in link mode, when performing an i-αppli in launcher mode, it is necessary to declare, by using the `LaunchApp` key of the ADF, that an i-αppli linked launching will be performed (set the value "yes").

(A) Acquiring the entry ID of the original i-αppli

To launch an i-αppli in launcher mode, use an entry ID, which respectively corresponds to each i-αppli downloaded in the mobile phone. An entry ID is a random integer value. In order to acquire a valid entry ID, it is necessary to have an i-αppli selection process performed by a user (when the application program specifies an invalid entry ID and attempts to launch an i-αppli, the i-αppli Runtime Environment displays a dialog which shows a warning to the user). Use the `com.nttdocomo.system.ApplicationStore` class to perform the i-αppli selection process. When calling the static selectEntry() method of this class, an i-αppli is suspended and the list of i-αpplis is displayed on the screen. When a user selects any i-αppli from the list of i-αpplis, the i-αppli resumes and the `selectEntry()` method returns the `ApplicationStore` object, which corresponds to the selected i-αppli. By calling the `getId()` method in response to the returned `ApplicationStore` object, the entry ID of the selected i-αppli can be acquired.

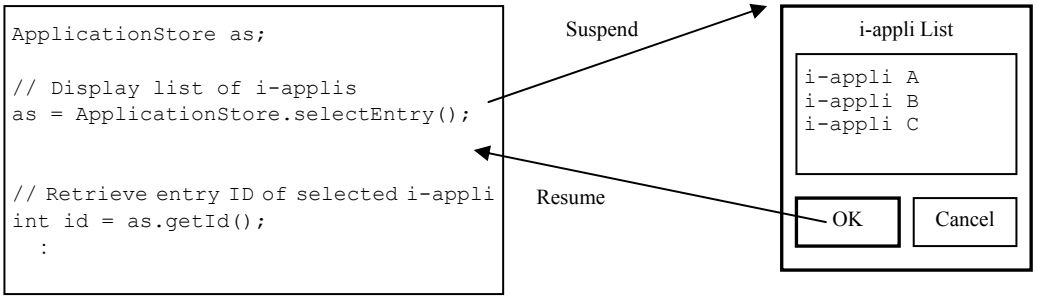


Figure 19: Acquiring the Entry ID of an i-αppli

An entry ID, which corresponds to an i-αppli, is valid as long as the corresponding i-αppli exists on the mobile phone. Even if the i-αppli is updated, the updated i-αppli inherits the ID.

Note that for an i-αppli selected by a user, the information other than the entry ID (for example, the i-αppli name) cannot be acquired.

(B) Launching an i-αppli with the entry ID specified

To launch an i-αppli in launcher mode with a specified entry ID, use the `IApplication.launch()` method.Set `IApplication.LAUNCH_AS_LAUNCHER` in the first argument (type of an application to be launched) of the `launch()` method and set the entry ID of a character string in the first element of the second argument (`String` array indicating the launching argument).

Unlike in link mode, it is not possible to pass parameters to the i-αppli to be launched in launcher mode.

Notes:

- These functions cannot be used from i-αpplis of immediate launch after download, or from inactive stand-by applications.
- When the i-appli to be launched is launched, the i-appli it was launched from closes.
- If the operation of the i-αppli to be launched terminates, the mobile phone returns to the stand-by state. Note that when a user forces the termination of the i-αppli to be launched, the status that appears after the i-αppli terminates varies, depending on the manufacturer.

- Even if the i-αppli to be launched is a stand-by application, when it is launched by i-αppli linked-launching, the launch mode is not Standby Launch, but Normal Launch.
- Depending on the content of the i-αppli, for example, a stand-by application which does not assume that it will be launched in normal mode, it may not be suitable for the application to be launched in launcher mode. An i-αppli like this can reject linked launching in launcher mode. In this case, declare a rejection of linked launching by using the `LaunchByApp` key of the ADF (set the value "`deny`"). An i-αppli for which this setting is specified will not be available for selection during the i-αppli selection process.
- In the i-αppli selection process, when a selectable i-αppli does not exist on the mobile phone or the user cancels the selection, the `ApplicationStore.selectEntry()` method returns null.
- For a newly downloaded i-αppli, a random entry ID is assigned to the i-αppli when it is downloaded. The entry ID continues to be valid as long as the i-αppli is not deleted (even after updating the i-αppli). When deleting the downloaded i-αppli and downloading the i-αppli again, that i-αppli is considered to be a new i-αppli and a random entry ID is assigned again.

[DoJa-5.0]

In profiles earlier than DoJa-4.x Profile, when an i-appli was link launched the i-appli it was launched from would enter a suspended state and a dialog to confirm the launch of the i-appli would be displayed by the system. However, from DoJa-5.0 Profile onward, this launch confirmation is not conducted.

11.5 i-αppli Update Function Linked Launching

In i-αppli update function linked-launching, it is possible to launch the i-αppli update function included in JAM from an i-αppli. The i-αppli update function of JAM corresponds to the “version upgrade” item provided in the sub-menus of the list of i-αpplis on a mobile phone. An i-αppli uses this function and prompts a user to update itself.

Use the `IApplication.launch()` method to launch the i-αppli update function. Set `IApplication.LAUNCH_VERSIONUP` in the first argument (type of an application to be launched) and set null in the second argument (`String` array indicating the launching argument).

When launching the i-αppli update function, a mobile phone performs the following operations.

(1) The application program calls the `IApplication.launch()` method. When this method is called, the i-αppli suspends.

(2) The i-αppli Runtime Environment checks whether it is possible to establish packet communication (on the display of a mobile phone, the “i” mark that indicates a communication status is lit up or blinking), and if it is possible, it prompts the user to confirm the update of the i-αppli. If packet communication is not possible or the user’s agreement is not obtained, the i-αppli resumes and the `IApplication.launch()` method returns to the caller without throwing an exception.

(3) When a user’s agreement is obtained in (2), the i-αppli ends. Then, the i-αppli update function is launched and the ADF that corresponds to the caller’s i-αppli is downloaded.As a result of downloading the ADF, if it is determined that the update of the i-αppli is not necessary (for example, the `LastModified` key of the ADF is not updated), the i-αppli is not updated.

(4) The JAR file is downloaded and the i-αppli on a mobile phone is updated. After that, if there are any changes in the contents of the ADF items (existence or non-existence of HTTP communication, etc.) that require user confirmation, each settings screen for those items appears.

(5) When the download of the JAR file and the specification of each settings are completed, the updated i-αppli automatically restarts. When a stand-by application, which has been launched in stand-by, is updated by using this function, this restart is also Standby Launch.

Notes:

- Linked launching cannot be performed for the i-αppli update function from i-αpplis of immediate launch after download, or from inactive stand-by applications.

11.6 Calling the Call Function

When calling the phone call function, it is possible to call the dial function (dialer) of the mobile phone from an i-αppli. Furthermore, it is possible to refer to the unique identification information included in the hardware of the mobile phone.

11.6.1 Sending Phone Calls from an i-appli

To call the phone call function from an i-αppli, use the `call()` method of the `com.nttdocomo.util.Phone` class. The phone call function is called by specifying, in the argument, a character string that represents a phone number and calling this method. On FOMA mobile phones which support video phone functions, you can conduct a video phone call by specifying an argument to this function. Also, on mobile phones which support the PushTalk feature, you can also conduct PushTalk calls by manipulating the phone call function which was called.

For a character string of a phone number, numbers from `0` to `9`, '#', '*', '`,`' (pause: 1 second), '`/`' (pause: wait for key entry) can be specified. A pause cannot be specified successively in a character string of a phone number. Furthermore, if '−', '`(`', '`)`', or '`.`' appear in the phone number string they will be ignored.

Note that an i-αppli that uses the phone call function must declare, by using the `UseTelephone` key of the ADF, that it will use the phone call function (set the value "`call`"). An i-αppli that does not make this declaration cannot use the phone call function.

Notes:

- A stand-by application in inactive mode cannot launch the phone call function.
- When the `Phone.call()` method is called, the i-αppli enters a suspended state. Next, the i-αppli Runtime Environment prompts the user to approve the call. If the user agrees, the dial is operated. When the phone call ends, the i-αppli resumes and the `call()` method returns to the caller.

11.6.2 Referring to Unique Identification Information from an i-αppli

Unique identification information is a unique ID, which is assigned to each device. The unique identification information of the mobile phone includes the following two types and the same values for these types are not assigned to multiple devices.

- The identification information of the mobile phone unit (serial number) ... `Phone.TERMINAL_ID`
- The identification information of the UIM card of the mobile phone if a UIM card is installed ... `Phone.USER_ID`

The information described above cannot be used by itself for specifying an individual user (there is no method for specifying an individual user based on the information described above). However, by combining and utilizing an appropriate user management system and the information, system management such as access management for a server can be easily realized.

Use the static `getProperty()` method in the `com.nttdocomo.util.Phone` class to acquire the unique identification information. For the argument of this method, specify which unique identification information described above is to be acquired.

Note that in order for an i-αppli to refer to the unique identification information, it is necessary to declare, by using the `GetUtn` key of the ADF, that the unique identification information will be referenced. When specifying the value "`terminalid`" to the `GetUtn` key, the identification information of a mobile phone unit can be referenced. When specifying the value "`userid`" to the `GetUtn` key, the identification information of a UIM card can be referenced. When referring to the identification information of both the mobile phone unit and UIM card, specify the information delimited by a comma.

Notes:

- From an i-αppli of immediate launch after download, the unique identification information cannot be referenced.
- In a mobile phone where a UIM card is not installed, the identification information of a UIM card cannot be referenced.

11.7 Calling the Phone Book Management Function

When calling the phone book management function, it is possible to call the native phone book management function of a mobile phone from an i-αppli, add a new phone book entry (a set of phone number, name, and mail address) under a user's agreement, and add a new phone book group.

An i-áppli that uses these functions needs to use the AccessUserInfo key of the ADF to declare that it will use these functions (set the value to "yes").

11.7.1 Adding a New Phone Book Group

Use the static addEntry() method in the com.nttdocomo.system.PhoneBookGroup class to add a new phone book group. For the argument of this method, specify the name of the phone book group to be registered or null.

When a new phone book group is added, a mobile phone performs the following operations.

(1) The application program calls the PhoneBookGroup.addEntry() method. When this method is called, the i-αppli suspends.

(2) The native user interface screen is launched in order to register the phone book group. On this screen, a user can not only permit or cancel the registration of a new phone book from an i-αppli, but change the group name to be registered.

(3) If the user permits the registration of a new phone book group, the new phone book group is added with the name that is specified at this time. Then, an i-αppli resumes and the addEntry() method returns to the application program. At this time, this method returns an entry ID, which respectively corresponds to a newly added phone group name in the mobile phone. The entry ID can be used to specify a group in the phone book entry when registering a new phone book entry, which is describe later.

If a user cancels the registration of a new phone book group, no exception occurs and this method returns -1.

Notes:

- A new phone book group cannot be registered from i-αpplis of immediate launch after download, or from inactive stand-by applications.
- Depending on the manufacturer, a new phone book group may be added by not adding a group, but by changing the name of a group that has been preset in the mobile phone.

11.7.2 Adding a New Phone Book Entry

Use the static addEntry() method in the com.nttdocomo.system.PhoneBook class to add a new phone book entry. This method supports the following information to be registered in a phone book entry.

- Name
- Kana reading of the name
- Phone number (multiple entries can be specified)
- Mail address (multiple entries can be specified)

- Phone book group to which the phone book entry is registered (specify the group name or the entry ID of the group)

There are two methods for passing this information as parameters to the `addEntry()` method:

(1) Enumerate each of these items as arguments to the `addEntry()` method.

(2) Create a `com.nttdocomo.system.PhoneBookParam` object (this object encapsulates each of the settings to be specified for the phone book entries), and pass this as an argument to the `addEntry()` method.

When a new phone book entry is added, the mobile phone performs the following operations.

(1) The application program calls the `PhoneBook.addEntry()` method. When this method is called, the i-αppli suspends.

(2) The native user interface screen is launched in order to register the phone book entry. On this screen, a user can not only permit or cancel the registration of a new phone entry from an i-αppli, but change each item for the entry to be registered.

(3) If the user permits the registration of a new phone book entry, the new phone book entry is added with the name that is specified at this time. Then, an i-αppli resumes and the `addEntry()` method returns to the application program.

Even if the user cancels the registration of a new phone book entry, an exception does not specifically occur and the application program is returned.

Notes:

- A new phone book entry cannot be registered from i-αpplis of immediate launch after download, or from inactive stand-by applications.
- The number of registerable phone numbers and mail addresses for one phone book entry differs depending on the model. However, it is possible to register at least three items for phone number and mail address respectively in any models.
- Depending on the model, it is possible to divide the items of the name and kana reading by family name and first name for a phone book entry, and manage them. When the `PhoneBookParam` object is not used, the name and kana reading are both set in the family name in a model described above.

When the PhoneBookParam object is used, the name and kana reading can be set in the family name and first name respectively.

- Specifying null for the phone book group name and specifying –1 for the entry ID of the phone group means that a phone book group to which the phone book entry is to be registered is not registered. In this case, a phone book entry is registered in the standard location of a mobile phone.
- When specifying the entry ID of a phone book group and registering a phone book entry, if the phone book group that corresponds to the specified entry ID does not exist, an exception occurs. Compared to this, when specifying the name of a phone book group and registering a phone book entry, if the phone book group with the specified name does not exist, the i-αppli Runtime Environment asks a user whether to create the phone book group.

11.8 Calling the Bookmark Management Function

When calling the bookmark management function, it is possible to call the native bookmark management function of a mobile phone from an i-αppli, and add a new bookmark (URL to be connected and title) under a user's agreement.

An i-áppli that uses the bookmark management function needs to use the `AccessUserInfo` key of the ADF to declare that it will use these functions (set the value to "yes").

Use the static `addEntry()` method in the `com.nttdocomo.system.Bookmark` class to add a new bookmark. For the argument of this method, specify the title of the bookmark to be registered and the destination URL.

When adding a new bookmark, a mobile phone performs the following operations.

(1) The application program calls the `Bookmark.addEntry()` method. When this method is called, the i-αppli suspends.

(2) The native user interface screen is launched in order to register a bookmark. On this screen, a user can not only permit or cancel the registration of a new bookmark from an i-αppli, but change the URL and title to be registered.

(3) If the user permits the registration of a new bookmark, the new bookmark is added with the contents that are specified at this time. Then, an i-αppli resumes and the `addEntry()` method returns to the application program.

Even though the user cancels the registration of a new bookmark, an exception does not particularly occur and the application program is returned.

Notes:

- A new bookmark cannot be registered from i-αpplis of immediate launch after download, or from inactive stand-by applications.
- The only URL schemes that can be registered as a bookmark is HTTP or HTTPS.

11.9 Calling the Schedule Management Function

When calling the schedule management function, it is possible to call the native schedule management function (depending on the manufacturer, this could be called by another name such as Calendar) of a mobile phone from an i-αppli, and add a new schedule (description and date) with the user's agreement. Furthermore, in DoJa-5.0 Profile and later, the scheduler can be link launched from an i-appli. (Linked launching of the scheduler was part of the i-appli Optional API in DoJa-4.x and earlier.)

An i-áppli that uses the schedule management function needs to use the `AccessUserInfo` key of the ADF to declare that it will use these functions (set the value to "`yes`").

11.9.1 Registering a New Schedule

Use the static `addEntry()` method in the `com.nttdocomo.system.Schedule` class to add a new schedule. For the argument of this method, specify the description, date, and availability of the alarm sound for the schedule to be registered.

For specifying the date for the schedule, use the `ScheduleDate` class of the `com.nttdocomo.util` package. In the `ScheduleDate` class, it is possible to represent a one-time schedule as well as periodic schedules such as daily, weekly, monthly, and yearly schedules (set the date type of the schedule).

When adding a new schedule, a mobile phone performs the following operations.

(1) The application program calls the `Schedule.addEntry()` method. When this method is called, the i-αppli suspends.

(2) The native user interface screen is launched in order to register a schedule. On this screen, a user can not only permit or cancel the registration of a new schedule from an i-αppli, but change the description, etc. to be registered.

(3) If the user permits the registration of a new schedule, the new schedule is added with the contents that are specified at this time. Then, an i-αppli resumes and the `addEntry()` method returns to the application program.

Even though the user cancels the registration of a new schedule, an exception does not particularly occur and the application program is returned.

Notes:

- A new schedule cannot be registered from i-αpplis of immediate launch after download, or from inactive stand-by applications.
- For registering a schedule, which date type is available depends on the manufacturer. The date type that can be commonly specified for all models is “one time”. Also, by using the `Schedule.getSupportedTypes()` method, it is possible to check what date type can be used for a particular model.

[DoJa-5.1]

In DoJa-5.1 Profile, a variation of the `ScheduleDate` class constructor was added which allows the specification of a time zone (`java.util.TimeZone` class object).

11.9.2 Linked Launch of Scheduler

Use the `IApplication.launch()` method to launch the scheduler from an i-αppli. To launch the scheduler, set the method’s first argument (type of application to launch) to `IApplication.LAUNCH_SCHEDULER`. Since no launch parameters are used to launch the scheduler in linked mode, set the second parameter of the `launch()` method (a `String` array indicating the launch parameters) to null.

Notes:

- Inactive standby applications may not launch the scheduler.
- Calling the `IApplication.launch()` enters the i-αppli to the suspended state. Next, the i-αppli Runtime Environment prompts the user to approve the launch of the scheduler. If the user agrees, then the scheduler is launched. Scheduler-linked launching does not terminate the i-ápplí, even if the scheduler launches. When the user terminates the scheduler operation, the i-αppli resumes, and the `launch()` method returns control to the application program.

11.10 Calling the Image Data Management Function

Mobile phones which support DoJa-3.0 Profile or later have a native image management area called My Pictures. When calling the image data management function, it is possible to call the native image data management function of the mobile phone from an i-αppli, and input and output images from the My Pictures area. For calling the image data management function, use the `com.nttdocomo.system.ImageStore` class.

An i-áppli that uses these functions needs to use the `AccessUserInfo` key of the ADF to declare that it will use these functions (set the value to "`yes`"). Also, an i-αppli cannot access images which are downloaded and saved by using a browser. Images accessible from an i-αppli are limited as follows:

- Images, which are shot and saved by the camera function of the mobile phone
- Images, which are saved by an i-αppli (Images, which are saved by other i-αpplis are also accessible. Note that when an i-αppli saves an image, if it is prohibited (*1) for other i-αpplis to access the image, the image cannot be accessed.)
- Images, which are loaded outside of the mobile phone by using the native infrared communication
- Images, which have been preinstalled when the mobile phone was shipped.
- For a mobile phones which installs an external memory such as memory card, the images moved to the My Pictures area from an external memory are accessible.
- On FOMA mobile phones, among images which are downloaded and saved from a browser, some images set as redistributable by the content provider are accessible.

(*1) In DoJa-4.0 Profile and later, an API is added to prohibit other i-αpplis from accessing the image when saving an image to the native image management area from an i-αppli. See Section 11.10.1 for details.

Notes:

- Some FOMA mobile phones can save and manage image data (iMotion format data) by using the native image data management function. However, it is not possible to handle iMotion format data by using this function from an i-αppli. This function only handles static images.

11.10.1 Adding New Image Data

Use the static `addEntry()` method in the `ImageStore` class to add new image data. Specify the `MediaImage` object representing the image you wish to register as this method's argument.

When new image data is added, a mobile phone performs the following operations.

(1) The application program calls the `ImageStore.addEntry()` method. When this method is called, the i-αppli suspends.

(2) The native user interface screen is launched to register the image data. On this screen, users can allow the new image data from an i-αppli to be registered or cancel the registration. The users can also check (view) the image data that the i-αppli is attempting to register.

(3) If the user authorizes the new image data to be added, the image will be registered. Then, the i-αppli will resume and the `addEntry()` method returns to the application program. At this time, this method returns an entry ID, which respectively corresponds to a newly added image data in the mobile phone. An entry ID can be used to load image data without requiring user's operations when reading image data with specified IDs, which is described later.

Even if the user cancels the registration of new image data, no exception will be thrown and this method will return -1.

[DoJa-4.0]

In DoJa-4.0 Profile and later, an API (the `ImageStore.addEntry(MediaImage, boolean)` method) is added to prohibit other i-αpplis from accessing the image when saving an image to the native image management area from an i-αppli. When this method is used to save an image, it is prohibited to access the image from other i-αpplis, or to take the image out of the mobile phone by using the mobile phone's native function.

Also, in DoJa-4.0 Profile, a variation (`ImageStore.addEntry(MediaImage[])`) of the `ImageStore.addEntry()` method is added to save multiple images with one operation (for example, user confirmation). Note that `ImageStore.addEntry(MediaImage[])` is a method which belongs to i-αppli optional APIs and may not be supported depending on the manufacturer.

[DoJa-4.1]

In DoJa-4.1 Profile and later, depending on the manufacturer, some models support saving Flash content to the native image management area. Such a model acquires the MediaImage object by specifying the data of the Flash content in the argument of the `MediaManager.getImage()` method and calling the method, and saves the Flash content by specifying the `MediaImage` object in the argument of the `ImageStore.addEntry()` method.

Note that even with a model which supports saving Flash content to the native image management area, it is not possible to use the `MediaImage` object generated from the Flash content for the purpose other than the above (for example, to play data using `VisualPresenter`). Furthermore, it is not possible to handle the Flash content saved in the native image management area when reading the image data selection or reading image data with a specified ID(*1).

(*1) In DoJa-5.1 Profile and later, depending on the manufacturer's implementation, Flash content stored in the native image management area can be extracted by selecting it like image data or by specifying its image data ID. However, playback of extracted Flash content is still not supported.

[DoJa-5.0]

On 903i series and newer mobile phones, a Decomail pictographic character feature was added allowing the user to inline embed a small image (20x20 pixel JPEG or GIF image) like an i-mode pictographic character into a Decomail.Depending on the manufacturer, images which meet both of the conditions outlined below will be stored in the Decomail pictographic character folder if they are registered by an i-appli.

- 20x20 pixel GIF (including animated GIF) or JPEG image.
- Can be redistributed (has not been set as non-redistributable).

Notes:

- Image data cannot be registered from i-αpplis of immediate launch after download, or from inactive stand-by applications.
- The MediaImage object to be specified for the `addEntry()` method must be available (the object's `use()` method must have been called).
- Image data registered by using the `ImageStore.addEntry(MediaImage)` method can be saved outside of a mobile phone by using the mobile phone's native functions (such as infrared communication and sending an i-shot mail). For image data which is not desirable to be saved outside of a mobile phone, consider using the following methods to save it.
  - Save image data not in the My Pictures area but in ScratchPad.
  - Save image data in the My Pictures area by using the `ImageStore.addEntry(MediaImage, boolean)` method, which is newly provided in DoJa-4.0 Profile.
  - Explicitly set the `MediaImage` object that corresponds to the image data as non-redistributable by using the `MediaResource.setRedistributable()` method, which is newly provided in DoJa-4.0 Profile and save the data in the My Pictures area by using the `ImageStore.addEntry(MediaImage)` method.

11.10.2 Reading Image Data Selection

Reading an image data selection is a function, which gives image data selected by a user from the list of images displayed in the screen every time in response to an image data request (calling a method) from the application program. Use the static `selectEntry()` method of the `ImageStore` class to read an image data selection.

When the image data selection is read, the mobile phone performs the following operations.

(1) The application program calls the `ImageStore.selectEntry()` method. When this method is called, the i-αppli suspends.

(2) The native user interface screen is launched to select an image data. The user selects image data for the i-αppli to access via this screen. The user may also cancel the operation.

(3) When the user finalizes the image data selection, the i-αppli resumes, and the `selectEntry()` method returns to the application program. At this time, this method returns the `ImageStore` object corresponding to the selected image data.

Even if the user cancels the image data selection, no exception will be thrown and this method will return null.

From the `ImageStore` object returned by the `selectEntry()` method, the selected byte image and the `MediaImage` object can be acquired.

The entry ID of the image data can also be obtained from the `ImageStore` object. This ID can then be used to read image data again without further user intervention when reading an image data with a specified ID, which is described below.

Notes:

- Image data selection cannot be read from i-αpplis of immediate launch after download, or from inactive stand-by applications.

11.10.3 Loading Image Data by Specifying its ID

An entry ID of image data is a unique integer value that is assigned by the system when an image is saved in the My Pictures area and that does not change until the image data is deleted or overwritten by another image. When an i-αppli once acquires a valid entry ID when registering a new image data or reading an image data selection, it can access the image data thereafter without any user's operations. Use the static `getEntry()` method of the `ImageStore` class to load image data by specifying its ID.

When specifying an entry ID of the image data in the argument of the `getEntry()` method and calling the method, the `ImageStore` object that corresponds to the image data is returned. From the `ImageStore` object, as well as reading an image data selection, the byte image for the image and the `MediaImage` object can be acquired.

Notes:

- It is not possible to load video data by specifying its ID from an i-áppli that launches immediately after download. Furthermore, this function can be used by a standby application in inactive mode because it requires no user intervention.

11.11 Calling the Camera Functions

Camera functions are installed in mobile phones supporting DoJa-3.0 Profile or later as standard. When calling a camera function, means are provided to access this camera function from an i-αppli. Some models supporting video filming support handling static images as well as movie images.

Class `com.nttdocomo.device.Camera` is used to call camera functions.

11.11.1 Camera Control

Camera control functions are provided by the `com.nttdocomo.device.Camera` class. Multiple camera devices are installed on some mobile phones and each device is identified by a camera ID (an integer starting from 0, assigned to each device permanently). Application programs can obtain a camera object by specifying the camera ID to the `Camera.getCamera()` method.

By using this function, i-αppli can perform following controls to camera objects.

- Executing camera device setup
- Launching native camera functions
- Obtaining camera images

(A) Executing camera device setup

There are three camera device setup items; device attribution, camera image size and frame image.

The device attribution is set by `Camera.setAttribute()` method. The defined attributes are as follows:

- Action mode (enabled only for static image shooting)
- Image quality
- Microphone volume (enabled only for movie image shooting)

The attributes that the actual camera device supports varies on each model, however, by `Camera.isAvailable()` method, application program decides the attribute the mobile phone is supporting. Also, current setup value for each attribution is available by `Camera.getAttribute()`.

`Camera.setImageSize()` method is used to setup the camera image size. The camera size is represented by the number of dots in vertical and horizontal direction. If the specified camera image size is not supported by the mobile phone, the specified size is adjusted to the supported size by the system. Also, `Camera.setFrameImage()` method is used to setup a frame image.

Some camera devices may support multiple camera image sizes, however, it is not always possible to shoot with frames in all of the sizes. Information related to camera image sizes for camera devices can be obtained by following methods:

- `Camera.getAvailablePictureSizes()`: To obtain the size of static images allowed to shoot
- `Camera.getAvailableMovieSizes()`: To obtain the size of movie images allowed to shoot
- `Camera.getAvailableFrameSizes()`: To obtain the size of static images allowed to shoot with frames

[DoJa-4.1] / [DoJa-5.1]

In DoJa-4.1 Profile (902iS) and later, optional functions for setting and retrieving focus mode settings were added in addition to the above functions for models which can switch between macro and standard focus modes by the system.

On models which support this functionality, you can use the `Camera` class' `setFocusMode()` method to set the focus mode for the camera when it launches. However, the user can still change the focus mode after the camera launches despite this setting. In that case, the details of the focus mode setting made by the user will not be provided as feedback to the `Camera` object.

You can also use the `getFocusMode()` method to check which focus mode is currently set and the `getAvailableFocusModes()` method to retrieve a list of all focus modes available on that particular model.

In DoJa-4.1 Profile, two focus modes are defined: `FOCUS_NORMAL_MODE` (normal mode) and `FOCUS_MACRO_MODE` (macro mode).In DoJa-5.1 Profile, the `FOCUS_HARDWARE_SWITCH` option was added which means that the focus mode can only be switched by the user on the hardware, and focus modes cannot be controled from an i-appli.

(B) Launching Native Camera Functions

Application programs can launch camera functions in static image shooting mode by calling the `Camera.takePicture()` method. Also models supporting movie image shooting via i-αppli can launch camera functions in movie image shooting mode by calling the `Camera.takeMovie()` method.

If a camera function is launched, i-αppli execution stops and a camera operation window is displayed on the screen. Actual shooting is performed based on the users' operations, and application programs cannot perform shutter operations.

When users exit the camera operation window, i-αppli execution is resumed. If the user saved the camera images when exiting the camera function, the image is maintained in a camera object.

Notes:

- Inactive standby applications cannot use the camera function.

(C) Obtaining Camera Images

The image taken by the camera function is maintained in a camera object until the next time the camera function is launched, or i-αppli exits, or `Camera.disposeImages()` method explicitly destroy it, and can be looked up as media image or byte stream form from application programs.

`Camera.getImage()` method is used to obtain the image saved in a camera object in media image format. Or `Camera.getInputStream()` method is used to obtain the image in byte stream (`JPEG` format for static images and i-motion format for movie images) format. For the argument to these methods, specify the index of image saved in a camera object. Usually 0 is specified for the image index, however, when continuous shooting is performed, the target image can be specified by the index.

11.11.2 Exchanging Camera Images Between Mobile Phones

Some mobile phone models supporting DoJa-3.0 Profile or later have a function to exchange camera images between mobile phones via infrared ports as a mobile phone native function.Since this function is achieved based on OBEX external connection function, image exchange is possible between i-αppli using OBEX external connection function and this native application.

When the native application exchanges images, objects compliant with IrMC vNote Version 1.1 are used. However, this function extended vNote object property to send/receive additional information necessary for image exchange.

The following shows the property details of objects used in this function.

Item No.	Property	Value	Description
1	VERSION	1.1	vNote version
2	X-DOCOMO-TYPE	JPEG or GIF	Image data format type
3	X-DOCOMO-SIZE	<width>X<height>	Image size (Horizontally x Vertically) X should be half-width alphabet upper case (ASCII code 0x58)
4	X-DOCOMO-FILESIZE	<filesize>byte	Image file size (in byte)
5	X-DOCOMO-FILENAME	Refer to the note.	File name (String of up to 24 bytes including extension in half-width alphanumerical)
6	SUMMARY	Refer to the note.	Image title (String within 18 bytes including SHIFT-JIS, QUOTED-PRINTABLE encoded)
7	DATE	Refer to the note.	Date and time of shooting or editing (UTC notation)
8	X-DOCOMO-BODY	Refer to the note.	JPEG or GIF data encoded in Base 64 format. Add <CR><LF> at the very end of this property value.

The following shows an example of describing vNote objects to exchange image data.

```
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:<Image data body encoded in Base64 format＋<CR><LF>>
END:VNOTE
```

* Set <CR><LF>（0x0d0a）at the end of each line. For X-DOCOMO-BODY, two <CR><LF> should be repeated together with <CR><LF> at the end of data.

When the i-αppli side sends image data to the native application side, the native application side acts as an OBEX server. The i-αppli side should be implemented as an OBEX client, and send image data (vNote objects) by a PUT operation.

Also when the native application side sends image data to the i-αppli side, the native application side acts as an OBEX client. i-αppli side should be implemented as an OBEX server, and receive image data (vNote objects) sent from the native application by a PUT operation.

Notes:

- When exchanging images, multiple send/receive for nVote objects cannot be performed. Only one image data can be sent for one operation.
- In the current profile, the data size that i-αppli can be sent/received using OBEX external connection function is limited to 100Kbyte at maximum (30Kbyte maximum for DoJa-2.0 Profile). when exchanging images, the overall size of above mentioned vNote objects must be under the limit. This is the same when the i-αppli side sends or receives images.
- For the `X-DOCOMO-FILESIZE` property, specify the file size before encoding to Base 64 format.
- When an i-αppli sends image data to native application side, the Name header for objects sent by PUT operation must be "`mypic.vnt`".
- When an i-αppli sends image data to native application side, the image size allowed to be sent (Vertical and horizontal size of images indicated by `X-DOCOMO-SIZE`) is limited to the camera image size supported by the mobile phone of the receiver side.

11.11.3 Code Reader Function

The code reader function provides functions for loading and recognizing data expressed as an image (a representative example is a barcode), using the camera's onboard camera device. The code recognition is provided by class `com.nttdocomo.device.CodeReader`.

The current Profile defines the following types of recognizable codes.

- `CodeReader.CODE_JAN13` (standard JAN type)
- `CodeReader.CODE_JAN8` (shortened JAN type)
- `CodeReader.CODE_OCR` (text recognition)
- `CodeReader.CODE_QR` (QR codes)
- `CodeReader.CODE_MICRO_QR` (micro QR codes)
- `CodeReader.CODE_39` (CODE-39 codes)
- `CodeReader.CODE_NW7` (NW-7 codes)

However, the recognizable codes for any model are these three: `CODE_JAN13`, `CODE_JAN8` and `CODE_QR`.

[DoJa-4.1]

`CODE_MICRO_QR`, `CODE_39`, and `CODE_NW7` were added in DoJa-4.1 Profile (902iS) and later profiles as optional code types.

The code-recognition procedure is as follows:

1) Acquire and set up a `CodeReader` object.
2) Load and recognize the code using the camera device.
3) Acquire the results of code recognition.

<Acquire and set up a `CodeReader` object>

Use the static `getCodeReader()` method of the `CodeReader` class to obtain a `CodeReader` object. For the argument to this method, specify a camera ID used in the camera control function as well (for models installed with multiple camera devices, devices having resolution available for code recognition may be limited to only one of those).

Before reading in a code, you must set the type of code to read in the `CodeReader` object. To set the code type, specify one of the code types listed above as the parameter to the `CodeReader.setCode()` method. You can use the `CodeReader.getAvailableCodes()` method to find out the types of code supported by the mobile phone.

Additionally, some manufactures support specifying `CodeReader.CODE_AUTO` as the code type; this causes the code reader to recognize the code type automatically.

[DoJa-5.0]

In DoJa-5.0 Profile and later, optional functions to set and acquire the focus mode were added to the `CodeReader` class as they were in the `Camera` class.

<Reading In and Recognizing Codes Using a Camera Device>

Use the `CodeReader.read()` method to start up the camera device, and read and recognize a code. By calling this method, the i-αppli is suspended and the camera device will be launched.

When the user photographs a code and initiates recognition processing, the photographed code is converted into data, and stored as a `CodeReader` object. If the user cancels the operation, or code recognition fails, no code data will be saved. In either case, when the user terminates the camera-device operation, the i-αppli resumes, and returns to the application program.

<Acquire the Results of Code Recognition>

If the camera device successfully reads and recognizes the code, the application program can acquire the results from the `CodeReader` object. The application program can refer to the following information as code reader-results:

- Type of code recognized (`getResultCode()` method)
- The data acquired from the code reader (`the getString()` or `getBytes()` method)
- The data type of the data acquired from the code reader (`getResultType()` method)

The data can have one of the following four types:

- Number string (`CodeReader.TYPE_NUMBER`)
- ASCII string (`CodeReader.TYPE_ASCII`)
- Non-ASCII string (`CodeReader.TYPE_STRING`)
- Binary (`CodeReader.TYPE_BINARY`)

The JAN code standard limits codes to alphanumeric characters, but a QR code can be a string including non-ASCII characters, or binary data.

The code recognition function is defined as the i-αppli optional API in the DoJa-3.0 Profile. It was later moved to the i-αppli standard API in the DoJa-3.5 Profile.

Notes:

- Inactive standby applications may not call the `CodeReader.read()` function.
- Since the code reader function uses the same camera device as the camera control function, if application programs attempt to use these functions simultaneously with the same camera ID, it could result in situation not intended by the developer. For example, if the `CodeReader.read()` method is called while the `Camera` object is storing an image, the image stored in the `Camera` object will be discarded.
- If a JAN code is read, a string containing all the data that was read, including the check digit, is passed to the application program. In other words, in the case of standard JAN format, a string of length 13 is returned to the application program, and in the case of shortened JAN format, a string of length 8 is returned.
- QR codes can handle data according to the following specification:

  - Version : 1 to 10 (whether version 11 and above is available is manufacturer dependent)
  - Model : Model 2 (Model 1 and MicroQR not supported)
  - Error correction level : L/M/Q/H

In addition, if handling Kanji data, Shift-JIS code (the default encoding of the i-αppli Runtime Environment) can be used.

- The granularity of the type of data read that the `CodeReader.getResultType()` method returns is manufacturer dependent. For example, if a number string was read, some manufacturers' mobile phones may identify this as the specific type (`CodeReader.TYPE_NUMBER`), but others may simply return `CodeReader.TYPE_ASCII` or `CodeReader.TYPE_STRING`, without inspecting the contents of the string. Use the value returned by this method as a hint for your application program to analyze the data acquired from the code reader.

Note that mobile phones supporting DoJa-3.5 Profile or later are installed with a function to link launch specific i-αpplis downloaded on the mobile phones by making the mobile phone native code recognition application read the two dimensional barcodes containing special information (Optional function for DoJa-3.0 Profile mobile phones).

Below is an example of text data(i-αppli launch information) included in the 2-dimensional barcode for linked launching of an i-αppli. When i-αppli launch information that has been converted into a two-dimensional barcode is loaded into the native code-reader application, a screen appears, prompting the user to authorize the launch of the corresponding i-αppli.

<Sample i-αppli Launch Information Text>

```
LAPL:
ADFURL:http¥://www.nttdocomo.co.jp/java/a.jam;
CMD:startup_Application_A;
PARAM:param1,i-mode;
PARAM:param2,iappli;
;
```

* For convenience, the text data is shown with word wrapping, but the text itself does not include line breaks.

Below are described the format and notation for i-αppli launch information:

```
LAPL:
ADFURL:<ADF URL(*1) of the i-αppli to launch>;
CMD:<launch command(*1)>;
PARAM:<parameter name(*2)>, <parameter value(*2)>;
    : (repeat PARAM statements as necessary)
;
```

* As with the sample code, the text data is shown with word wrapping for ease of viewing, but the text itself does not include line breaks.

(*1) For these displayed special letters, '¥', ':', ';' , add an escape letter '¥' just in front of them.

(*2) For these displayed special letters, '¥', ':', ';', ',' , add an escape letter '¥' just in front of them.

- `LAPL:`

  Identifier indicating the start of the i-αppli launch information.

- `ADFURL:<ADF URL` of the i-αppli to launch>`;`

  Specify the URL of the ADF for the i-appli you want to launch via this i-appli launch information in ASCII format (maximum 255 bytes). Must be terminated by a semicolon. The i-αppli to be launched must have already been downloaded to the mobile phone.

- `CMD:`<launch command>`;`

  The launch command, in ASCII format (maximum 250 bytes). Must be terminated by a semicolon. If the launch command specified here (excluding the terminating semicolon) does not exactly match the launch command assigned by the `AllowPushBy` key of the i-αppli 's `ADF` authorizing launch (the string in the `AllowPushBy` key following

"Code:"), the i-αppli cannot be launched.See Section 15.5.1 for more information about the ADF's AllowPushBy key.

- PARAM:<parameter name>, <parameter value>;

  Specify parameters to pass to the i-αppli to be launched (optional). One parameter has the following format:

  PARAM <parameter name>, <parameter value>;

  * Must be terminated by a semicolon.

  The parameter name is used to specify a parameter that can be obtained by the i-αppli on launch via the IApplication.getParameter() method. You can specify up to 16 parameters in one i-αppli launch information statement. Note, however, that the total length of all the names and values of all parameters is restricted to 255 bytes. Additionally, only ASCII characters may be used in parameter names or values.

- ;

  Identifier indicating the end of the i-αppli launch information.Complements the start identifier (LAPL:).

See the manual for one of the two-dimensional barcode editing tools or the like for instructions on how to convert i-αppli launch-information text created according to these rules into a two-dimensional barcode.

Notes:

- The mobile phone user can configure the mobile phone to prohibit the launch of i-αppli s via two-dimensional barcodes.2If the launch of i-αppli s via two-dimensional barcodes is prohibited, then loading one will not launch an i-αppli.
- When an i-αppli launched from a two-dimensional barcode exits, the phone returns to the state it had before the i-αppli was launched (the code-reader results are displayed). Note that when a user forces the termination of an i-αppli that has been launched from a two-dimensional barcode, the status that appears after the i-αppli terminates varies, depending on the manufacturer.
- Although it is difficult for a human to directly read a two-dimensional barcode visually, they are easy for devices supporting two-dimensional barcodes (not just dedicated readers, but any mobile phone supporting two-dimensional barcodes) to read. You must take care not to set confidential or sensitive data in the command or parameters of i-αppli launch information.

11.12 Calling the Camera Video Data Management Function

Some functions are added to handle camera video data (iMotion) for FOMA mobile phones in the DoJa-4.0 profile or later versions. In DoJa-4.0 Profile, general i-αpplis cannot use the native video data management function, however, in DoJa-4.1 Profile or later, general i-αpplis can use this function to register new video data in the native storage area.

An i-ápplì that uses this function needs to use the `AccessUserInfo` key of the ADF to declare that it will use this function (set the value to "`yes`").

[DoJa-4.1]

A new video data registration by the video data management function is available in general i-αppli since DoJa-4.1 Profile. For previous profiles, this function is only available for trusted i-αpplis.

Also, the video data management function can be called from i-αppli is only new video data registration as i-αppli basic API of DoJa-4.1 Profile. Functions to obtain video data saved in the native storage area from i-αppli (such as selective read and ID specified read) are positioned as i-αppli optional API in DoJa-4.1 Profile.

Use the static `addEntry()` method of the `com.nttdocomo.system.MovieStore` class to add new camera video data to the native storage area. Specify a `MediaImage` object representing the camera video data (iMotion format data) you wish to register as this method's parameter.

When new camera image data is added, the mobile phone behaves as follows:

(1) The application program calls the `MovieStore.addEntry()` method. When this method is called, the i-αppli suspends.

(2) The native user interface screen launches to register the camera video data. On this screen, users can allow or cancel the new camera image data from an i-αppli to be registered. The users can also check (view) the video data the i-αppli is attempting to register.

(3) If the user authorizes the new movie data to be added, the movie will be registered. Then, the i-αppli will resume and the `addEntry()` method returns to the application program. Here, this method returns a unique entry ID bearing a 1-to-1 relationship with the camera video data newly added to the mobile phone. For models supporting video data acquisition from the native storage area, the entry ID can be used as a key to obtain the video data.

Even if the user cancels the registration of new camera video data, no exception will be thrown and this method will return -1.

Notes:

- New video data cannot be registered from i-αpplis of immediate launch after download, or from inactive stand-by applications.
- The `MediaImage` object to be specified for the `addEntry()` method must be available (the object's `use()` method must have been called). This `MediaImage` object should also have been been created from camera video data in iMotion format (it is not allowed to specify a `MediaImage` object created from a static image or an animated GIF as an argument for the `MovieStore.addEntry()` method).
- When storing `MediaImage` objects created from video data in the `MovieStore` class, the re-distribution enabled/disabled information set in the original video data is also stored as it is. The i-αppli may set this information by using the `MediaResource.setRedistributable()` method, which was added from DoJa-4.0 onward.

Chapter 12
Infrared Remote Control

In DoJa-3.0 Profile, an API was added to send control signals from infrared ports on mobile phones to general devices supporting infrared remote controls. Those infrared remote control supporting devices can be operated externally by constructing and sending proper control signals from i-αppli.

Notes:

- For i-αppli infrared remote control function, a primitive API is defined to construct and send physical wave form of infrared signals. On the other hand, for general infrared remote control supporting devices, each supporting device manufacturer has its own designs specifically as to which wave forms would be accepted as control signals. The scope of i-αppli specification does not include virtualization/standardization of control signals for infrared remote controls. Each developer is required to design the i-αppli to send appropriate control signals according to the specification of target devices supporting infrared remote controls.
- For the i-αppli infrared remote control function, it is expected to be able to control devices 2-3 meters away from the mobile phone. However, how far it is able to control actual devices is affected by the surrounding environment and light receiver capability of the infrared remote control supporting devices as well as signal output and directivity from the mobile phone.

12.1 Configuration of Control Signals

i-αppli infrared remote control functionality can handle control signals for infrared remote controls shown below. Most of the general home electric appliances sold within Japan can express control signals in the range described below. However, as described above, since control signals are designed by each device manufacturer in a proprietary fashion, it is not guaranteed that the following description is applicable to all infrared remote control supporting devices.

Carrier Frequency and Modulation Scheme:

The control signals for infrared remote controls are handled as pulse phase modulation signals (PPM signals: hereinafter logical wave form carried by carrier wave is referred as control signals) carried by carrier wave of carrier frequency 25kHz-50kHz (rectangular wave expressed by ON and OFF of infrared emitting). Bit values (logical 0 and logical 1) are expressed by combining the length of High zone and Low zone of control signals.

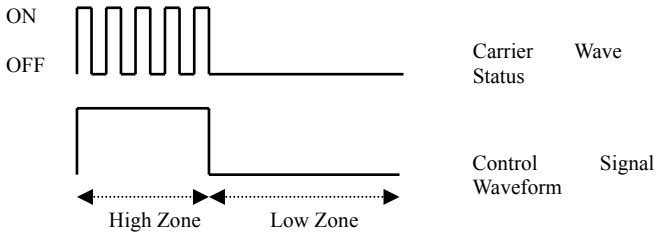
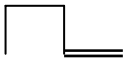
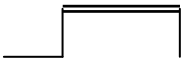


Figure 20: Configuration of Control Signals by Carrier Wave

There are following two patterns for the bit value expression method on control signal wave forms, and the decision to which method is used depends on the design of infrared remote control supporting devices. For i-αppli infrared remote control function, either one of the following methods can be specified to be used (The logical 0 and logical 1 are defined by the length of the double line in the waveform chart).

	An example of wave form for logical 0	An example of wave form for logical 1
High precedence pattern Express logical 0 and logical 1 by the length of Low zone.		
Low precedence pattern Express logical 0 and logical 1 by the length of High zone.		

In the above description, the following items depend on the design of the infrared remote control supporting device, and application programs can set those to appropriate values for the appropriate device.

- The length of High zone and Low zone per waveform of carrier wave (This can decide the carrier frequency of carrier wave)
- Bit value expression method (High precedence pattern or Low precedence pattern)
- High zone length and Low zone length for logical 0 on control signals
- High zone length and Low zone length for logical 1 on control signals

Configuration of Data Frame:

Data sent from infrared remote controls are created by adding a start part (combination of control signals for each one of High zone and Low zone which have a certain length) and a stop part (a control signal consists of one High zone which has a certain length) to the control signal for bit data modulated by the method described above

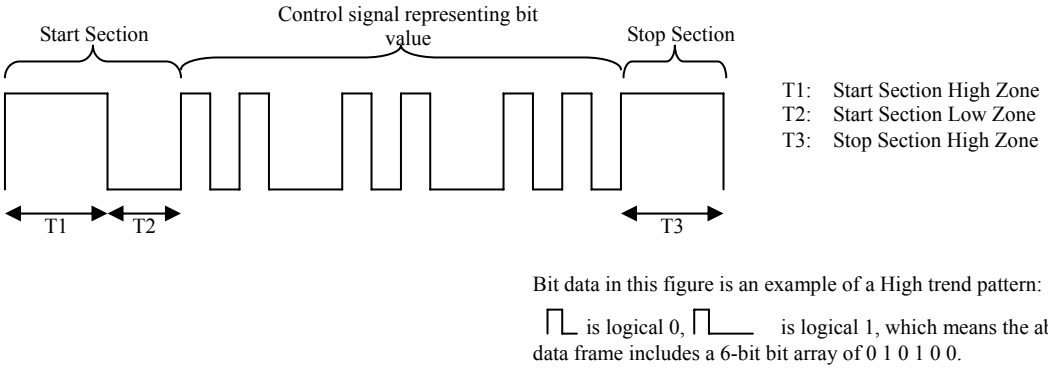


Figure 21: Configuration of Data Frame

The length of High zone and Low zone in the start part, and High zone in the stop part depends on the design of the infrared remote control supporting device, and application programs can set those to appropriate values for the device. Also, the number of bits for bit data that can be included in one data frame has limitation for each version of DoJa Profile. Data frame is expressed by the class `com.nttdocomo.device.IrRemoteControlFrame` discussed in the section 12.2.2. Please refer to the section 12.2.2 for limitations on the number of bits in a data frame as well.

Overall Data Configuration Sent From Infrared Remote Control:

Data sent from infrared remote controls are expressed as a group of data frames. There are some cases as totally different data frames are continuously sent or a certain data frames are repeatedly sent.

In addition, infrared remote controls typically repeat sending from the leading data frame again after finished sending the defined data frame once. This matches that for example, by holding down the volume button on remote controls for TVs, control signals are kept sending for that period.

Data frames are repeatedly sent until the time (timeout time) or repeat count specified when sending data is passed, or an application program explicitly stops sending data.

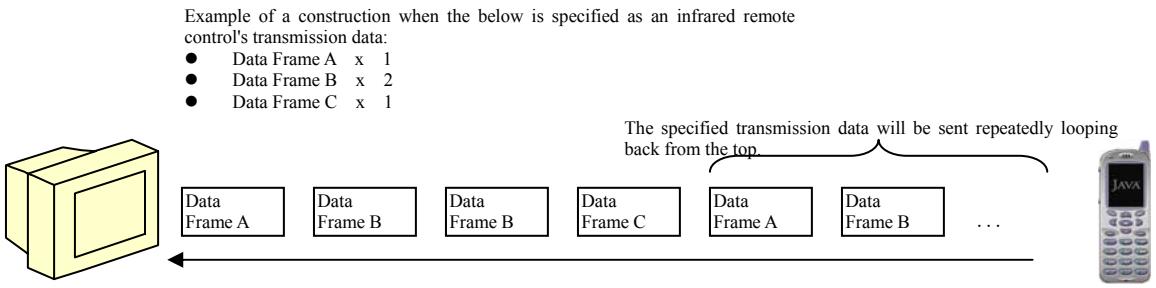


Figure 22: An Example of Data Frame Sent From Mobile Phone

Application programs can setup sending order of created data frame information, sending interval, time out time or repeat count until data is completely sent to appropriate values for target devices. Data sent from infrared remote controls are sent by using class `com.nttdocomo.device.IrRemoteControl` described in the section 12.2.1.

12.2 Infrared Remote Control API

The following classes for i-αpplis are used for the infrared remote control functions described in the previous section.

- `com.nttdocomo.device.IrRemoteControl`
- `com.nttdocomo.device.IrRemoteControlFrame`

Class `IrRemoteControl` has a role as a sender to send control signals from infrared ports. Also, class `IrRemoteControlFrame` express data frame consisting data to be sent. Application programs create a set of data frames in the form of `IrRemoteControlFrame` objects, and send the set from infrared ports using an `IrRemoteControl` object.

12.2.1 IrRemoteControl

Class `IrRemoteControl` has a function to send data frames specified in the form of `IrRemoteControlFrame`. Also, it has a method to perform configuration across the remote control function.

Application programs use class `IrRemoteControl` in the following manner.

(1) Obtain an `IrRemoteControl` object by using the `IrRemoteControl.getIrRemoteControl()` method. `IrRemoteControl` object corresponds one-on-one to the infrared remote control port on a mobile phone and there cannot be multiple objects in an application program.

(2) Performs physical configuration for infrared remote controls. Infrared remote control setup items are as follows:

- Setup for High zone length for carrier wave and Low zone length (carrier frequency)(`setCarrier()` method)
- Expression pattern for logical 0 waveform on control signals, High zone length and Low zone length (`setCode0()` method)
- Expression pattern for logical 1 waveform on control signals, High zone length and Low zone length (`setCode1()` method)

For expression patterns for logical 0 waveform and logical 1 waveform, specify a constant `PATTERN_HL` (High precedence) or `PATTERN_LH` (Low precedence) of class `IrRemoteControl`.

(3) Generate a set of `IrRemoteControlFrame` objects to represent data frames to send, and send it by using `send()` method. In `send()` method, it is allowed to specify the time (timeout time) to auto-send repeatedly a set of data frames. The default timeout time is 10 seconds.
`send()` method returns to the application program immediately after accepting the send request for data frames, and the data frames are sent asynchronously to the application program subsequently. The data frames are sent repeatedly until any one of the following occurs.
- Timeout period is exceeded
- Explicit instruction to stop sending from an application program (`stop()` method)
- Interruption due to events such as incoming phone call
- HTTP communication execution from an application program
- Application program exit (including abnormal exit)

[DoJa-4.1]

In DoJa-4.1 Profile, for the `send()` method, variation is added to specify the repeat count when sending a set of data frame as well as the timeout period to stop sending data. By using this method, even though it is within the timeout period, sending can be stopped with the specified repeat count.

Notes:

- In the following cases, application programs are unable to call the send() method.
  - When stand-by applications are inactive.
  - When mobile phones are set in self mode.
  - When i-αppli uses the infrared port (infrared remote controls, OBEX external connection)
  - During packet communication.
- For infrared remote control configuration, the unit of configured values or the range for configurable value varies on the method. For the unit of configured values for each method, please refer to the API Reference.
- Depending on the characteristics of infrared devices employed by each manufacturer, the value of waveform parameter specified in API (High zone length or Low zone length) might be adjusted to values (units) which the device is capable of handling.

12.2.2 IrRemoteControlFrame

The `IrRemoteControlFrame` class expresses each data frame. Application programs generates objects by using the constructor of this class, and configures data frames by setting the start part, the bit data part, and the stop part.

Data frame setup items are as follows:

- Data frame sending interval (`setFrameDuration()` method)
  Specifies the interval from the point to start sending the data frame to the point to start sending the next data frame.
- Data frame repeat sending count (`setRepeatCount()` method)
  Specifies the count when sending same data frame continuously in one data frame set. Specify 1 when not sending continuously the same data frame.
- Start part format (`setStartHighDuration()` and `setStartLowDuration()` method)
  Specifies the length of High zone and Low zone in the start part in the data frame.
- Stop part format (`setStopHighDuration()`)
  Specifies the length of High zone in the stop part in the data frame.

- Bit data value (`setFrameData()` method)
  Specify a value sent as bit data in the form of byte array or long value.

  [DoJa-3.0] / [DoJa-4.0] / [DoJa-4.1]

  In DoJa-3.0 Profile, the bit data length which can be sent on any models is 48-bit per frame.This value is also expanded to 512 bits per frame in DoJa-4.0 Profile and 1024 bits per frame in DoJa-4.1 Profile.

Notes:

- For data frame configuration, the unit of configured values or the range for configurable value varies on the method. For the unit of configured values for each method, please refer to the API Reference.
- Depending on the characteristics of infrared devices employed by each manufacturer, the parameter values related to the start part, stop part and data frame sending interval specified in API (High zone length or Low zone length) might be adjusted to values (units) which the device is capable of handling.

Chapter 13
3D Graphics/3D Sound

For DoJa-4.0 Profile or later, to enhance i-αppli rendering capability, an API has been added to control 3D graphics and 3D sound controlled by i-αpplis.

Traditionally, 3D graphics rendering functions were provided as an i-αppli extension API; however, in this profile, this function was enhanced to be defined as a new i-αppli basic API. Also, to utilize fully the stereo speakers installed on mobile phones, 3D sound control functions are defined.

This chapter presents an overview of the 3D graphics rendering functions and 3D sound control functions. As well, for mobile phones supporting this profile, HI Corporation's Mascot Capsule Engine Micro 3D Edition Version 4 has been installed as a standard 3D graphics engine. For i-αppli programming details utilizing this engine (including the collision detection features added in DoJa-5.0 Profile), please refer to "Micro3D Programming for i-αpplis" provided by HI Corporation.

13.1 3D Graphics Rendering Function

13.1.1 3D Graphics Rendering

3D graphics rendering functions in this profile are based on the high level 3D graphics rendering function defined as i-αppli extension API in previous profiles, and by enhancing the functions and deleting some of the functions, class configuration and method configuration changes have been added accordingly. Since it is based on the high level 3D graphics rendering functions, 3D model rendering can be performed by relatively simple API calls.

3D graphics rendering function is provided in `com.nttdocomo.ui.graphics3d` package. In addition, utility functions provided in `com.nttdocomo.ui.util3d` package are used for numeric value calculation relating to 3D.

Classes consisting of 3D Graphics Rendering are shown below. These class configuration, methods in the classes and field configuration are based on the (i-αppli extension API) high level 3D graphics rendering function included in the option package, however certain changes have been made and they are not compatible with the previous APIs including their name and usage. To operate easily existing applications using the previous high level 3D graphics rendering function, it is recommended that the high level 3D graphics rendering functions included in the optional package should be used in this profile as well.

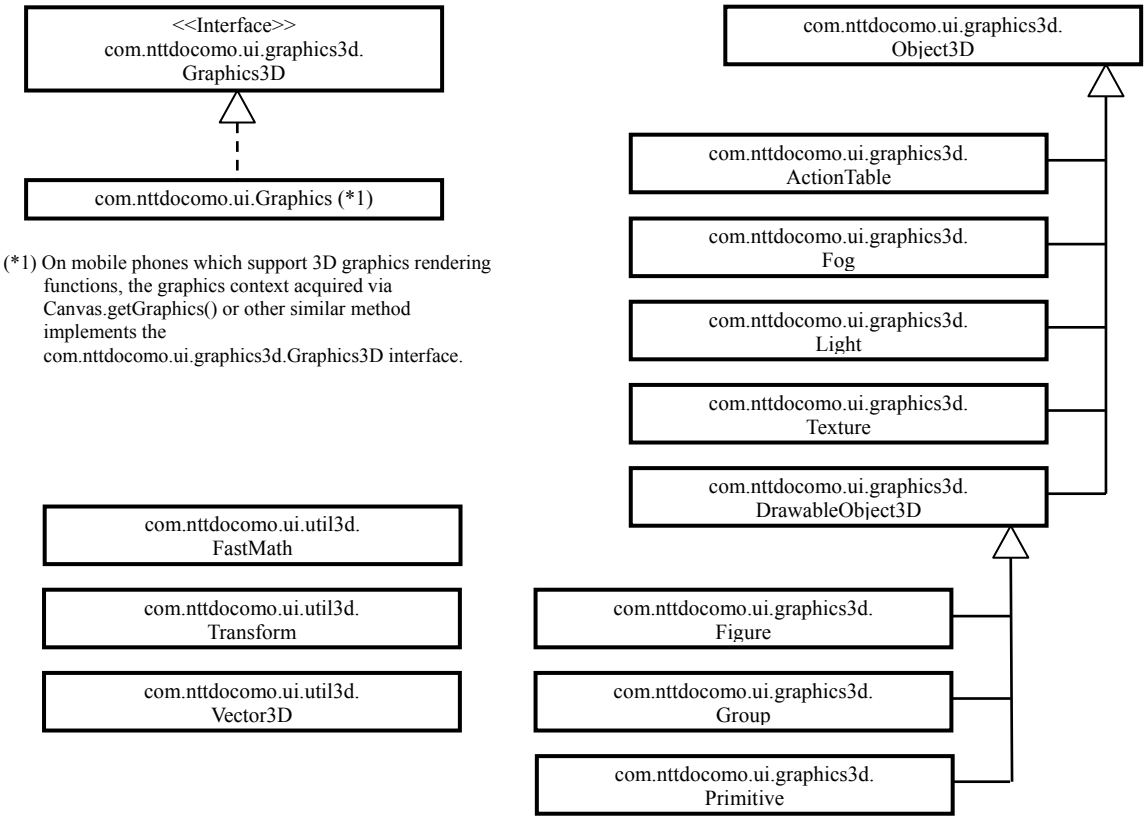


Figure 23: Class Configuration for 3D Graphics Rendering Function

The following provides a functional overview of each class and interface.

Class Interface	Function Overview
`Graphics3D`	This is an interface to define methods which must provide graphics contexts to support 3D Graphics Rendering functions. Whether the graphics context corresponding `Image` supports this interface or not is dependent on each manufacturer.
`Object3D`	This is the base class for all 3D objects.A 3D object indicates all objects which affect the rendering results of 3D graphics.3D objects are roughly divided into two: objects such as Figure or Primitive, which can be directly targeted for rendering and objects such as Light or Fog, which affect the rendering results. In this class, `createInstance()` is defined as a method to generate 3D objects, however, depending on the type of 3D objects, objects are generated by using the constructor.
`ActionTable`	This is a class to maintain an action (animation data) of a 3D model. The animation data is created using commercially-available 3D authoring tools and the like. The pose to be adopted by a 3D model in a specific defined period of time is referred to as a “frame”, and action is composed of a sequence of frames. It is possible to store a number of actions in `ActionTable`. The frame corresponds to “current time” maintained internally by all 3D objects, and by setting the current time in Figure objects described later, it is possible to make 3D models pose.
`Fog`	This is a class to maintain data to add a fog effect. This supports a linear fog mode and an index fog mode.
`Light`	This is a class to maintain light source data. Maximum number of light sources to be supported varies by each manufacturer (can be confirmed by the `getMaxLights()` method). It supports an environment light source, a parallel light source, a point light source, and a spot light source.

`Texture`	This class stores texture data. This can handle both a texture used for model mapping and a texture used for environment mapping. As well, how to use the texture for environment mapping in this function has been changed from the way used in the high level 3D graphics rendering function of i-αppli extension API. In this function, texture for environment mapping is used by setting in each texture object for model mapping instead of 3D graphics context (`Graphics3D` object).
`DrawableObject3D`	This is the base class for a rendering-enabled 3D object. Each object of the `Figure` class, the `Group` class, the `Primitive` class deriving this class can be targeted for rendering by `Graphics3D.renderObject3D()` method. In addition, this has some methods such as a method to easily recognize collisions of two objects in the 3D space, a blend function, and a distortion correction function for textures.
`Figure`	This is a class to maintain 3D model data. This object can render and be specified to the method `Graphics3D.renderObject3D()` as a rendering target.
`Group`	This is a class to represent a group that is a 3D object aggregate.This object can render and be specified to the method `Graphics3D.renderObject3D()` as a rendering target. It can assign multiple `Figure` objects or `Primitive` objects to it and render at once. It can also assign Fog objects or `Light` objects to it to affect rendering in a group.
`Primitive`	This is a class to maintain an array of primitives (geometric shapes, such as points, lines, or planes) that stores vertex data for use in rendering primitives. This object can render and be specified to the method `Graphics3D.renderObject3D()` as a rendering target.
`FastMath`	A high speed math utility with the premise of using with 3D-related functions.Since mobile phones supporting this profile are installed with CLDC-1.1, a floating point number operation can be used, but the floating point number operation is internally replaced with an integer operation in this class. Therefore, this might cause a larger error than a normal floating point number operation, but it performs faster calculations.
`Transform`	Handles the matrixes for conversion of 3D affines. The objects from this class are used whenever rotating a 3D model or moving its view point.
`Vector3D`	This is a class to represent three-dimensional vectors. The inner product, outer product, and normalization utility methods are provided.

Notes:

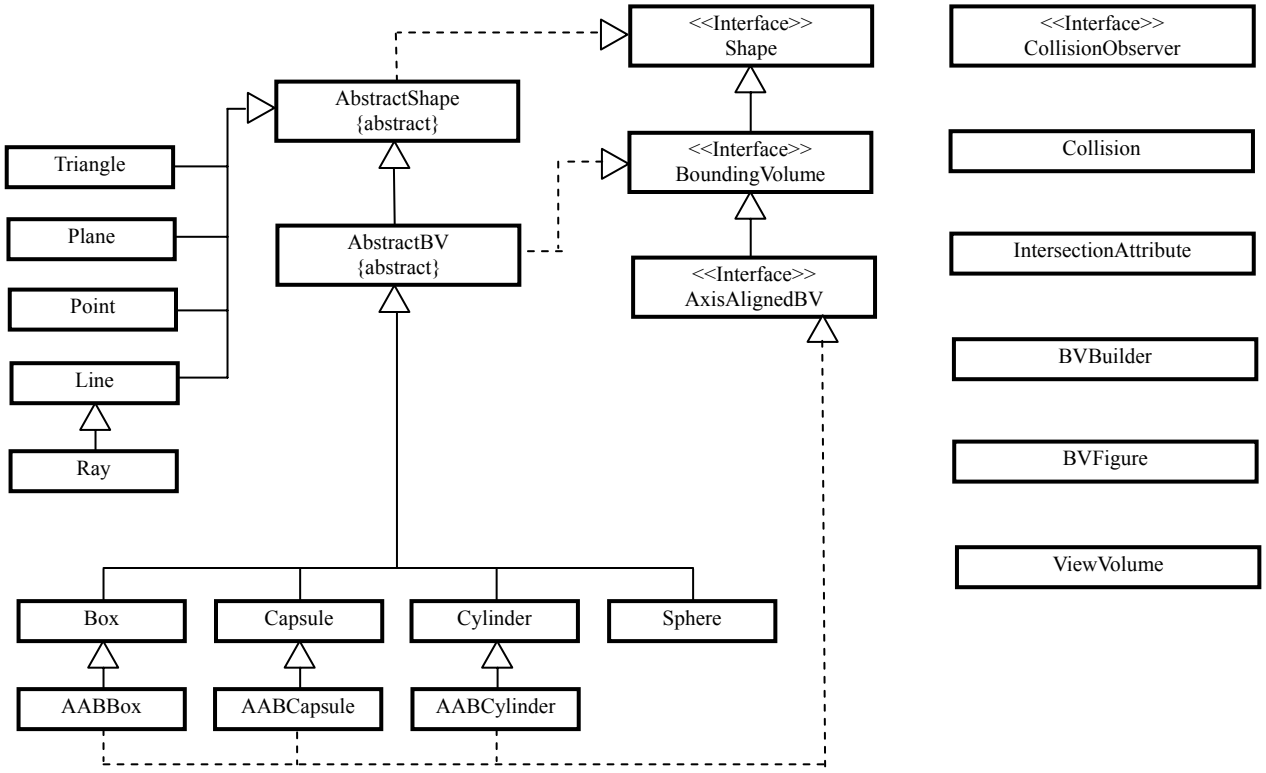
- For mobile phones supporting this profile, HI Corporation Mascot Capsule Engine Micro 3D Edition Version 4 has been installed as a standard 3D graphics engine. For i-αppli programming details utilizing this engine, please refer to “Micro3D Programming for i-αppli” provided by HI Corporation.
- The high level 3D graphics rendering function in i-αppli extension API (`com.nttdocomo.opt.ui.j3d` package) and this function cannot be used concurrently in one i-αppli. If an attempt is made to use one while using the other one, an exception will be thrown.
- This function has been enhanced compared to the previous high level 3D graphics rendering functions, such as function enhancement related to light sources (supporting multiple light sources and point/spot light sources), fog effect and newly introduced object grouping function. On the other hand, the command list and tone shading functions in the high level 3D graphics rendering function of i-αppli extension function are not supported, so they cannot be used.
- In the previous high level 3D graphics rendering functions, to define a 3D function API on CLDC-1.0 where floating point number format does not exist, a unique unit system was used to handle values after the decimal point using integer format. In contrast, in this function, it was changed to an interface using float point number format of CLDC-1.1 (platform for this profile). However, for system internal calculations, the calculation uses integer formats, same as before, to maintain the functionality.
- Some mobile phones are installed with designated hardware to speed up the process related to 3D graphics. For those models, there are some cases where calculation is inferior to other models and emulators in accuracy due to the limitation of hardware implementation. Especially, when this problem affects position relationship calculation in Z direction (depth buffer process), objects which have very similar front and back relationship may not be rendered in front and back relationship as intended.

13.1.2 Collision Detection

In DoJa-5.0 Profile, the `com.nttdocomo.ui.graphics3d.collision` package was newly added to perform collision detection on objects in 3D space together with the 3D graphics rendering functions discussed in the previous section.

A collision detection function (the `isCross()` method) is already provided with the `DrawableObject3D` class discussed in the previous section; however, that function only performs collision precisely on a per polygon basis, even for complex 3D objects made up of many polygons. This makes this function unsuitable for performing high speed collision detection operations on complex 3D objects. The collision detection functions discussed in this section perform collision detection using simplified shapes instead of the actual 3D objects themselves. This means that they are less accurate than the `DrawableObject3D.isCross()` method, but can be performed much faster. For content where processing speed is an important factor, we recommend using the collision detection functions presented in this section rather than the `DrawableObject3D.isCross()` method.

The classes and interfaces in the `com.nttdocomo.ui.graphics3d.collision` package which make up the collision detection functions are shown below.



The following provides a functional overview of each class and interface.

Class/Interface	Function Overview
`AABBox`	This class represents a `Box` which is parallel to world coordinate axes.
`AABCapsule`	This class represents a `Capsule` whose central axis is parallel to the world coordinate Y-axis.
`AABCylinder`	This class represents a `Cylinder` whose central axis is parallel to the world coordinate Y-axis.

AbstractBV	An abstract class that represents a three dimensional shape. This class implements the BoundingVolume interface. Three dimensional shapes are provided as concrete classes by inhereting this class.
AbstractShape	An abstract class which becomes the basis for all shapes; implements the Shape interface. Non three dimensional shapes are provided as concrete classes by inhereting this class. Three dimensional shapes are provided as concrete classes by inheriting the AbstractBV class, which in turn inherits this class.
AxisAlignedBV	Among three dimensional shapes used for comparison decisions, this interface is implemented by all three dimensional shapes that are placed parallel to world coordinate axes.
BoundingVolume	Among the shapes used for comparison decisions, this interface implements three dimensional shapes. This interface inherits the Shape interface.This package supports rectangular solids, cylinders, spheres, and capsule shapes.
Box	This class represents a rectangular solid and is one of the solid shapes used for comparison decisions.
BVBuilder	This class is used to create a BVFigure object or a BoundingVolume object (any of the three dimensional objects which implement the BoundingVolume interface) which can be attached to a BVFigure object from a Figure.
BVFigure	A Figure which has a bone structure and is made up of multiple BoundingVolume objects. Normal Figures (Figure class) make a shape by attaching polygons to them, but a BVFigure makes a shape by attaching BoundingVolume objects to it.
Capsule	This class represents a capsule shape and is one of the solid shapes used for comparison decisions.
Collision	This class provides functions related to collision detection. Collision detection, intersection detection, distance calculation, and other such functions are provided by this class.
CollisionObserver	This interface should be implemented by any observer class designed to receive notification about collisions and information about those collisions when a collision is detected in the Collision class. Applications which want to receive such notifications must create an observer class that implements this interface, create an object from that class, and assign it to a created Collision object.
Cylinder	This class represents a cylinder and is one of the solid shapes used for comparison decisions.
IntersectionAttribute	This class represents intersection information that is to be passed to the CollisionObserver's notification method when an intersection is detected between a Figure and a Ray.
Line	This class represents a line and is one of the non-solid shapes used for comparison decisions.
Plane	This class represents an infinite plane and is one of the non-solid shapes used for comparison decisions.
Point	This class represents a point and is one of the non-solid shapes used for comparison decisions.
Ray	This class represents a ray and is one of the non-solid shapes used for comparison decisions.
Shape	This interface is implemented by all shapes used in collision detection. The collision detection methods supplied by this package do not directly determine when Figure collide with each other. Rather, it determines when simple shapes placed at nearly the same location of these Figure collide with each other instead. The Shape interface is implemented by all shapes used for these comparison decisions.
Sphere	This class represents a sphere and is one of the solid shapes used for comparison decisions.
Triangle	This class represents a triangle and is one of the non-solid shapes used for comparison decisions.
ViewVolume	This class provides functionality to determine which BoundingVolume objects can be seen in the current view frustum (the field of view from the camera's perspective). By using this class to determine which objects are not visible within the camera's current field of view, you can find out only which objects actually require rendering on the screen at that time.

Notes:

- For i-αppli programming details utilizing these collision detection functions, please refer to “Micro3D Programming for i-αppli” provided by HI Corporation.

13.2 3D Sound Control Function

This profile defines 3D sound control function to give some effect in which sound is three-dimensionally audible from various directions, not only from right and left but also from up and back, utilizing stereo speakers installed on mobile phones. Using this function makes users feel that they are hearing sound from a certain direction by specifying the position relationship (referred as location) of the listener and the virtual sound source, or it makes users feel that the virtual sound is moving while making sound by specifying the location while the time is passed.

This section describes about how to use the 3D sound control API.

13.2.1 Class Configuration for 3D Sound Control Function

3D sound control function is provided in the com.nttdocomo.ui package and the `com.nttdocomo.ui.sound3d` package. In addition, same as 3D graphics rendering function, utility functions provided in `com.nttdocomo.ui.util3d` package are used for numeric value calculation relating to 3D.

The classes consisting of or relating to 3D Sound control functions are shown below.

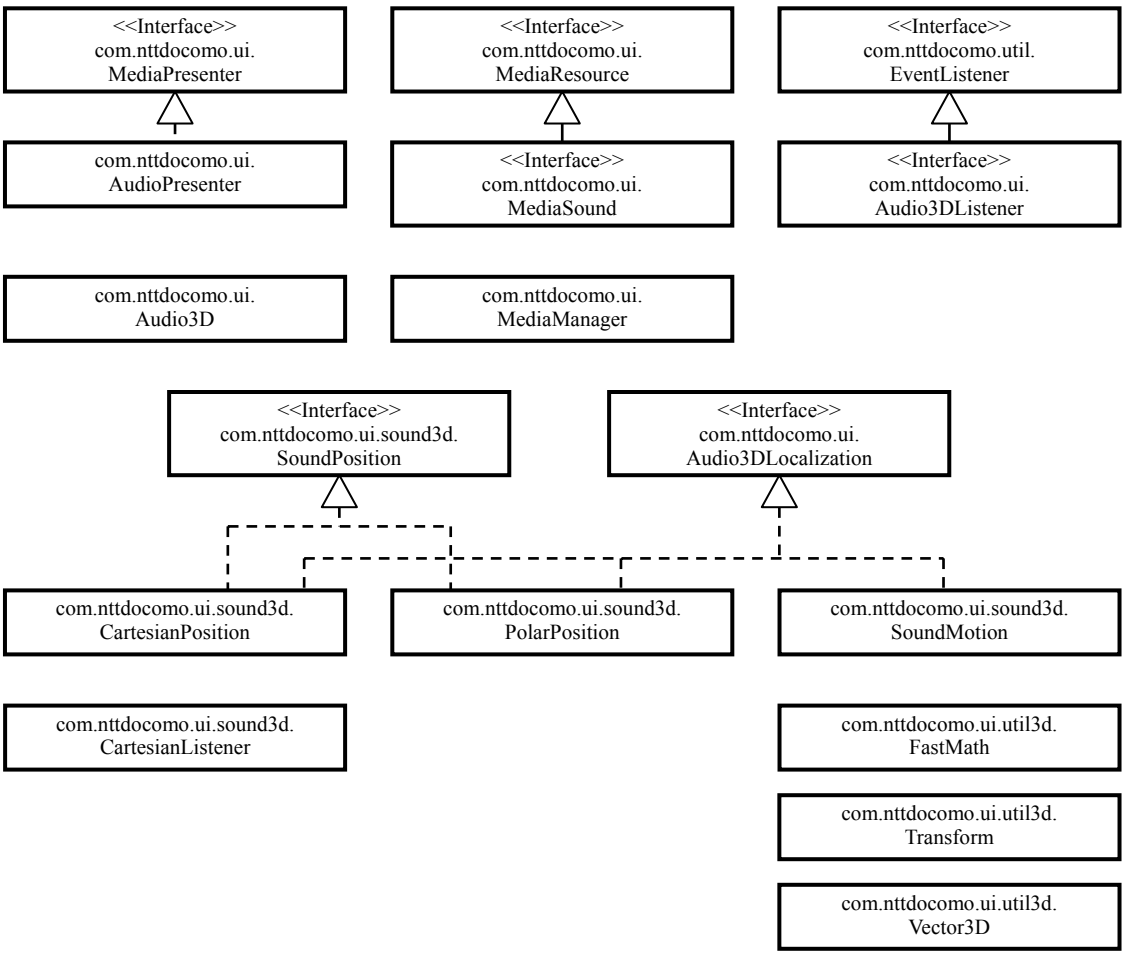


Figure 24: Class Configuration for 3D Sound Control Functions

An overview of each class and interface function which makes up the 3D sound control functions is shown below.

Class Interface	Function Overview
`AudioPresenter`	An audio presenter. In 3D Sound Control, an audio presenter is equivalent to a virtual sound source and the location (the relationship between the listener and the virtual sound source) is controlled by the controller (`Audio3D` object) corresponding to each audio presenter.In this profile, `getAudio3D()` method is added to take out an `Audio3D` object corresponding to the presenter in class `AudioPresenter`.
`Audio3D`	This is a controller to control the 3D sound for each AudioPresenter. It enables/disables the 3D sound control or controls the location by using the `Audio3D` object. The number of available virtual sound sources in the i-αppli (the number of 3D sound control resources) depends on the manufacturer, but the number of 3D sound control resources available in any model is four.i-αpplis can use `Audio3D` objects within the scope of this limitation. [DoJa-5.0] The number of 3D sound control resources available for use on all mobile phone models was increased with DoJa-5.0 Profile. In DoJa-4.x Profile, the maximum number of 3D sound resources available for use on all mobile phone models is two.

MediaSound	Media data to represent sound data. For 3D Sound Control, 3D control can be automatically performed by embedding the location information into the sound data in advance and using it when playing back the sound, other than a method in which 3D control is performed by specifying the location from i-αppli. When embedding the location information into the sound data, multiple 3D sound control resource can be assigned to one sound.In class MediaSound in this profile, a function is added to obtain how many 3D sound control resources are assigned to the location information embedded in the corresponding sound data.
Audio3DLocalization	This is an interface to represent location.There are some methods (classes) to represent location in the 3D sound control function, and any classes implement this interface. However, methods and fields are not defined for this interface.
SoundPosition	An interface to represent fixed location represented by the position of a listener and the position of a virtual sound source. A class implementing this interface represents a relative position of both by specifying a position of the listener and the position of a virtual sound source. However, methods and fields are not defined for this interface.
PolarPosition	This is a class to implement SoundPosition interface and represents the position relationship between a listener and a virtual sound source using a polar coordinate system. The polar coordinate system assumes that a listener faces a certain direction at the origin and represents a position of the virtual sound source by specifying the azimuth, elevation, and the distance from there.
CartesianPosition	This is a class to implement SoundPosition interface and represents a relative position (location) between a listener and a virtual sound source by using the Cartesian coordinate system consisting of three axes: x axis, y axis and z axis. While PolarPosition can be considered a local coordinate system where a listener is located at the origin, CartesianPosition can be considered a so-called world coordinate system containing a listener who is located at any position and a virtual sound source. However, even in CartesianPosition, the position relationship of the virtual sound source viewed from the listener is calculated ultimately, and used.
CartesianListener	A class to represent a listener located on the Cartesian coordinate system when representing the location by the CartesianPosition class. Setting the direction of the listener required to represent location on the Cartesian coordinate system will be set by using this class.
SoundMotion	A class SoundMotion is a class to represent "moving location". It enables a user to feel that a virtual sound source moves along a route by setting an elapsed time since beginning of moving and the location at the time as route information.
Audio3DListener	An interface to define an event listener to notify an event related to the 3D sound control to an application program. A motion complete event is defined in SoundMotion (moving location) in this profile.
FastMath	A high speed math utility with the premise of using with 3D-related functions.Mobile phones supporting this profile are installed with CLDC-1.1, and floating point number operations supported by KVM can be used in this version, but the floating point number operations are internally replaced with an integer operation in this class. Therefore, this might cause a larger error than a normal floating point number operation, but it performs faster calculations.
Transform	Handles the matrixes for conversion of 3D affines. When giving the direction of listener to CartesianListener, transformation matrix to the view point coordinate can be set and used.
Vector3D	This is a class to represent three-dimensional vectors. The inner product, outer product, and normalization utility methods are provided. This is used to represent a position or a direction on a coordination system.

13.2.2 Audio3D

Class Audio3D has a role as a controller to perform 3D sound control. An Audio3D object is assigned to each AudioPresenter object, and can be obtained by using AudioPresenter.getAudio3D() method.

To switch on and off of 3D sound control, Audio3D.enable() method and Audio3D.disable() method are used.For 3D Sound Control, 3D control can be automatically performed by embedding the

location information into the sound data in advance and using the information when playing back the sound, other than a method in which 3D control is performed by specifying the location from i-αppli. Whether 3D control is performed by using the location information embedded in sound data or enabling the location instruction from i-αppli by ignoring the location information embedded in sound data is specified in the argument for `enable()` method.

If 3D sound control is enabled by using `Audio3D.enable()`, the system consumes resources to 3D control sound (referred to as 3D sound control resources).The total number of these resources depends on the manufacturer, but as of DoJa-5.0 Profile at least four are available on all mobile phone models. (In DoJa-4.x Profile the minimum number is only two for all models.) 3D sound control resources are resources to represent a virtual sound source, i.e. a speaker moving virtually, and when i-αppli controls by specifying the location, one resource is consumed per `AudioPresenter`. However, when controlling by using the location information embedded in sound data, depending on the content of the sound data, one `AudioPresenter` may use multiple resources. How many 3D sound control resources are used for certain sound data can be known by specifying an argument of `MediaSound.AUDIO_3D_RESOURCES` in the `MediaSound.getProperty()` method.

After enabling 3D control, `Audio3D.setLocalization()` method is used to setup specifically the position relationship between a listener and a virtual sound source by specifying the location. When controlling the location by the moving location (`SoundMotion`), an event listener can be used to receive a notice as an event of the completion of location move by calling `Audio3D.setListener()`.

How to specify the location is discussed in the next section.

Notes:

- Without using the moving location (SoundMotion), it is possible to make users feel like the location is moving by repeating the setup change for the location in a certain interval via i-αppli. In this case, the interval to change the location should be roughly more than or equal to the returned time (msec) from Audio3D.setTimeResolution(). Since the location change requires certain loading, if the location is changed in smaller interval than the returned interval from this method, the system is loaded excessively so that operation as intended may not be achieved.

13.2.3 Specifying Location

There are three ways to represent the location expressing the position relationship between a listener and a virtual sound source as follows;

Use `CartesianPosition` / `CartesianListener`

Specify the location of a listener and a virtual sound source using Cartesian coordinate system consisting of 3 axes: x axis, y axis and z axis. Although both a listener and a virtual sound source can exist any position in the coordinate system, the position of sound source viewed from the listener is calculated ultimately to be reflected to the actual effect. `CartesianListener` is a class to represent a listener located in the coordinate. The position relationship between both is represented by setting the position and direction of the listener in `CartesianListener` object, and by setting the object and the location of virtual sound source to `CartesianPosition` object.

Use `PolarPosition`

The position of virtual sound source is specified by locating a listener facing to the positive direction of Z axis at the center of coordinate, and by specifying the azimuth, elevation and distance from the listener. Since a listener always exists at the origin of coordinate, the position of virtual sound source specified represents the position relationship with the listener.

Use `SoundMotion`

In contrary to that `CartesianPosition` and `PolarPosition` represent the fixed location which does not move automatically unless the setup for parameters is not changed by application programs, `SoundMotion` represents location which automatically moves tracing specified routes. The route information is specified by giving the lapsed time since the move is started and the location (`CartesianPosition` or `PolarPosition`) at the time as a set. By giving multiple route information, it is possible to make it trace complicated routes. When using `SoundMotion`, a notice of completion of the location move can be received as an event by using `Audio3D.setListener()` method and by setting a 3D sound control listener.

These classes are implemented with `Audio3DLocalization` interface to represent the location and can be set to an argument of `Audio3D.setLocalization()`.

`SoundMotion` is used when the route is already decided before moving the location. In contrast, when the route during the location move is decided dynamically via users' ad-hoc operation, instead of using `SoundMotion`, `CartesianPosition` and `PolarPosition` are used so that application program can change the location setup dynamically.

Chapter 14
Utility API

This chapter describes utility functions not included in the function categories discussed earlier.

14.1 Digital Signature API

This section describes the digital signature API introduced in the DoJa-4.1 Profile.

In FOMA mobile phones, except for the first model, a client certificate, which has been issued for each client, can be stored in a UIM card and FirstPass, which provides SSL client authentication functionality during Internet access, can be used. Digital signature API provides a function to perform digital signature on any data (PKCS#7 Conversion to Signed Data Format) using this FirstPass client certificate. Use of this API enables authentication of the signer and confirmation of the data integrity.

Notes:

- When signing in Digital Signature API, the client certificate issued in FirstPass is used. Therefore, for mobile phones performing the digital signature, the client certificate of FirstPass needs to be downloaded on the UIM card in advance. Also, to receive FirstPass service, it is required that a UIM card equal to or later version 2 be used (Version 1 UIM cards cannot receive the FirstPass service). The version information for the UIM card currently inserted into mobile phones can be checked by specifying `Phone.UIM_VERSION` as an argument for the static method `getProperty()` of class `com.nttdocomo.util.Phone`.
- To verify the signature performed in the digital signature API, it is required that the certificate of authentication station in FirstPass be used (DOCOMO root CA certificate). On mobile phones DOCOMO root CA certificate is integrated as the factory default setting. Or to use the DOCOMO root CA certificate on the server side, it is necessary to subscribe to the service separately with NTT DOCOMO.
- For FirstPass details and to apply to use Docomo root CA certificate on the server-side, refer to the NTT DOCOMO site below.

  http://www.nttdocomo.co.jp/service/anshin/firstpass/index.html

[DoJa-4.1]

Digital signature API and `Phone.UIM_VERSION` are newly created in DoJa-4.1 Profile.

14.1.1 Digital Signature API Class Configuration

Classes to configure Digital Signature API are provided in `com.nttdocomo.security` package. Class configuration for this function is shown below.

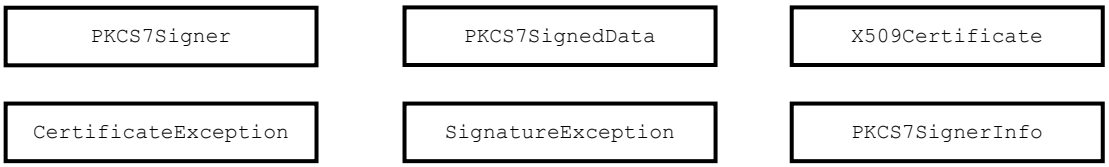


Figure 25: Digital Signature API Class Configuration

Functional overviews for each class are as follows;

Class	Function Overview
`PKCS7Signer`	Class that provides a function to generate data with digital signature. Digital signature is performed by giving data to be signed to an object in this class. Data with digital signature are represented by the class `PKCS7SignedData` described below.
`PKCS7SignedData`	Represents data with digital signature. Binary data expression for data with digital signature can be derived or contents included in data with digital signature (data before signature) can be derived from objects in this class.
`X509Certificate`	Represents a X509 certificate. Data with digital signature include the signer's certificate information. In Digital Signature API, certificate information contained in data with digital signatures (`PKCS7SignedData` object) can be obtained as objects in this class.
`PKCS7SignerInfo`	Class that represents information about the signer included in a digital signature. This class' object can be retrieved using the `getSignerInfos()` method from a `PKCS7SignedData` object.
`CertificateException`	Defines exceptions regarding to certificates. An exception occurs when the terminal does not have root certificates corresponding to the signature for verifying, or when they become invalid due to the expiration even if the root certificate exists.
`SignatureException`	Defines exceptions of process regarding to signatures. An exception occurs when PIN input prompted for users are cancelled or failed when performing digital signature, or when verification result for digital signature is failure.

14.1.2 Use of Digital Signature API

Overview of how to use Digital Signature API is described below. There are mainly two cases when using Digital Signature API.

- Performing digital signature onto data to be signed (use of class `PKCS7Signer`)
- Handling data with digital signature (use of class `PKCS7SignedData`)

Performing digital signature onto data to be signed

The procedure to perform digital signature onto data to be signed is shown below.

1) Generate `PKCS7Signer` object using the constructor.

2) Perform setup for `PKCS7Signer` object. There are two setup items, digest algorithm selection and content type setup for data to be signed. The digest algorithm is set by specifying an argument "`SHA-1`" or "`MD5`" for `setDigestAlgorithm()` method. The default digest algorithm is SHA-1. The content type is also set by specifying a constant `DATA` or `SIGNED_DATA` defined in `PKCS7Signer` as an argument for `setContentType()` method. `DATA` is specified when signing on ordinary data, while `SIGNED_DATA` is specified if the data to be signed are data with signature (i.e. duplicate digital signature).

3) Setup data to be signed for `PKCS7Signer` object using `update()` method. Calling `update()` method more than once enables to setup data to be signed in more than one batches or using `reset()` method enables to destroy data to be signed which have been setup.

4) Call `sign()` method after setting up data to be signed. This method performs digital signature on data to be signed, and returns `PKCS7SignedData` object as a result. In the digital signature process, digest algorithm can select SHA-1 or MD5 phase 2), however, encoding method is RSA fixed.

`PKCS7SignedData` object obtained as above can be handled in accordance with the handling method for data with digital signature shown below.

Handling Data with Digital Signature

Following operations are allowed for data with digital signature (PKCS7SignedData object).

- Verifying Data with Digital Signature

  Data with digital signature are allowed to be verified by using `verify()` method. This verification enables authentication of the signer and confirmation of the data integrity.

- Obtaining binary data expression for data with digital signature

  Data with digital signature can be obtained as binary data in ASN.1 format encoded with DER using `getEncoded()` method. By using this method, data with digital signature can be obtained as a byte array which can be sent externally by means such as HTTP communication.

  `PKCS7SignedData` object can be also generated by calling the constructor giving this binary data as an argument. By utilizing this, it is possible to send signed data to another mobile phone and perform operations such as verification after objectifying them by i-αppli on destination.

- Obtaining contents (data before signature) contained in data with digital signature

  Content data before signature can be obtained from data with signature by using `getContent()` method.

- Obtaining certificate information contained in data with digital signature

  Certificate information contained in data with digital signature can be obtained by using `getCertificates()` method. Certificate information is represented as an object in class `X509Certificate`. If multiple certificates are contained in data with digital signature, certificate information can be obtained for all of them. From `X509Certificate` object, information such as information about the certificate issuer and a person to be authorized (general name, country name and organization name) and certificate expiration data.

- Obtaining signer information contained in data with digital signature

  Signer information contained in data with digital signature can be obtained by using `getSignerInfos()` method. Signer information is represented by a `PKCS7SignerInfo` class object. The `getSignerInfos()` method returns all signer information contained in digitally signed data. From a `PKCS7SignerInfo` object you can obtain the signer information's issuer identifier name values (general name, organization name, department name) and serial number information.

The figure below shows the relationship of data handled in Digital Signature API.

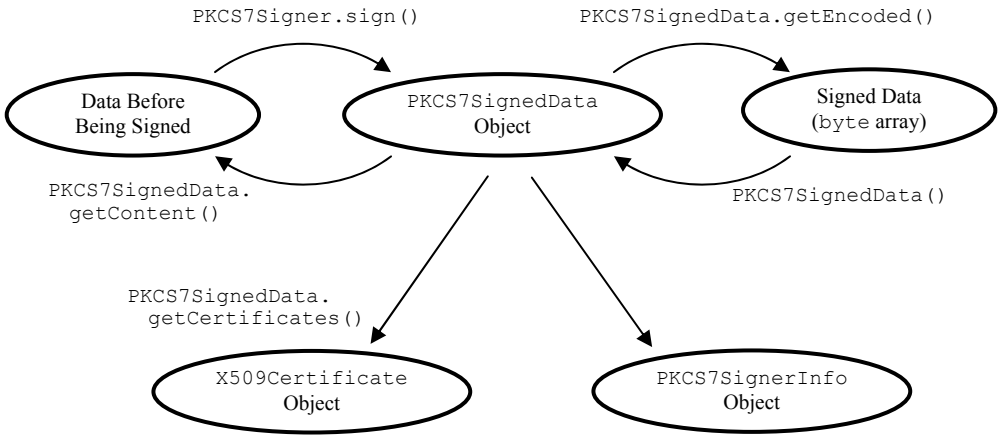


Figure 26: Relationship of Data Handled in Digital Signature API

[DoJa-5.0]

Among DoJa-5.0 Profile or later mobile phones, some models have functionality which allows them to, in addition to FirstPass provided client certificates, obtain a client server distributed by a content provider through an i-mode site (i.e., the i-mode browser on the mobile phone) and make that client certificate usable on the user's side. In DoJa-5.0 Profile, the following APIs are defined as i-appli APIs to allow client certificates downloaded in this way to be used for digital signing.

`CertificateStore` Class (`com.nttdocomo.system` Package):

A class used to manipulate the native storage area for client certificates. By calling this class' static method `selectEntryId()`, the developer can have the user select any of the client certificates stored on the mobile phone and notify the i-appli of that client certificate's entry ID.

`PKCS7Signer.sign(int)` method:

By specifying the client certificate's entry ID acquired via `CertificateStore.selectEntryId()`, that client certificate can be used for digital signing.

For FOMA 904i series and newer DoJa-5.0 Profile compatible mobile phones, the `getSerialNumber()` method was added to obtain the serial number of a `X509Certificate` class certificate. This method does not exist on FOMA903i series or earlier DoJa-5.0 compatible mobile phones; therefore, this function is part of the i-appli Optional API within the DoJa-5.0 Profile. As a pre-condition for running i-applis on FOMA903i series or earlier mobile phones, do not use the `X509Certificate.getSerialNumber()` method.

Furthermore, in DoJa-5.0 Profile, the following two exception states were added to those defined in `SignatureException`.

- `ENCRYPTED_DIGEST_ERROR`
- `SECURITY_CODE_REJECTED`

[DoJa-5.1]

The `PKCS7SignerInfo` class and `PKCS7SignedData.getSignerInfos()` method were added in DoJa-5.1 Profile. Furthermore, the `INVALID` state was added to the `CertificateException` state constants to represent when a certificate included with digitally signed data has expired.

Chapter 15 Building, Testing, and Packaging of i-αppli

This section describes the i-αppli development cycle. Though NTT DOCOMO provides the development environment for the i-αppli development on PC (appli development environment, hereafter), in this chapter, individual details information of each additional tools and i-αppli development environment. For more details regarding the i-áppli development environment, refer to the User Guide for i-áppli Development Environment.

It is regarded that the developers have used Java-2 SDK, Standard Edition (on or over version 1.3.1), they have knowledge on tools and documents contained in J2SE SDK.

To install a developed i-αppli to the cell phone for testing, it is necessary to setup the web site to distribute i-αppli (internet connection is necessary). See Chapter 15 regarding the distribution of i-αppli.

15.1 Downloading and Installing the i-αppli Development Environment

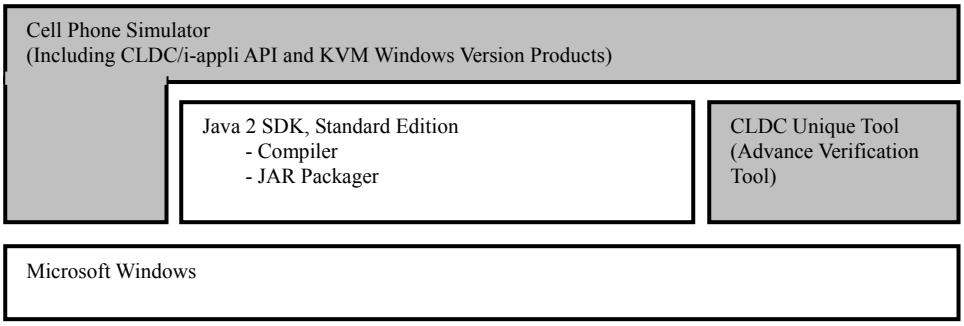
To use the i-αppli development environment, get the i-αppli development environment archive files distributed by NTT DOCOMO. For more details regarding the i-áppli development environment, refer to the User's Guide for i-áppli Development Environment.

To install i-αppli development environment, it is required to prepare J2SE SDK besides i-αppli development environment. After installing J2SE SDK, install i-αppli development environment.

15.2 Components of the i-αppli Development Environment

i-αppli development environment is a simple development environment to develop i-αpplis on a PC. Using these components, a build and simulation of i-αppli execution is possible on PC.

Components structure of i-αppli development environment is shown below.



NOTE: Some hatching components are included in the i-appli Development Environment.
Furthermore, the operating environment supported by the i-appli Development Environment (Microsoft Windows, Java 2 SDK version, etc.) depends on the version of the i-appli Development Environment (i.e., the supported profile). For further details, see the i-appli Development Environment User's Guide.

Figure 27: i-αppli Development Environment

The following shows an overview of each component included in i-αppli development environment.

- Cell Phone Simulator
  Main component of the i-αppli Development Environment. Using the mobile phone simulator, the developer can build (compile, pre-validate, JAR packaging), test and execute. i-αppli build can be done from each build's command line.
  In this document, the cell phone simulator is sometimes referred as emulator.

- CLDC Unique Tool
  CLDC unique tool contains a pre-verify tool. Pre-verification refers to a part of byte code verification operation by standard Java virtual machine beforehand (at application build time).By adopting the pre-verification method in CLDC, the cost will be less than verification during application execution. In the i-appli Development Environment, classes that have not been pre-verified cannot be used.
  The pre-verification tool can be used not only from i-αppli functions of the cell phone simulator, but also from the command line of independent tools.

[DoJa-2.0]

The i-αppli development environment provided after DoJa-2.0 profile has a few changes in the component structure and i-αppli development environment for DoJa-1.0 profile.

15.3 Overview of Development Cycle

The following figure illustrates the development cycle:

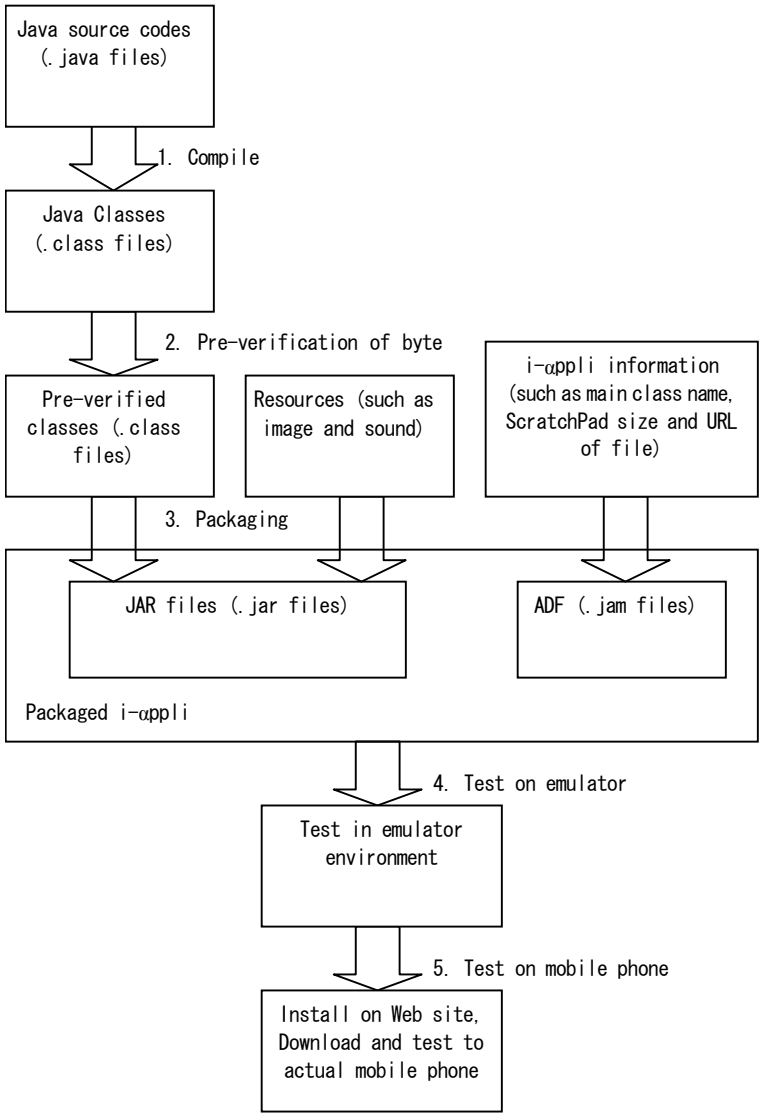


Fig. 28 Development Cycle

Compile - To compile a class file from source code files, J2E SDK Java compiler (javac) is used. The compiler also verifies that the application is not using any packages, classes, or methods not belonging to i-αppli execution environment.

Byte Code Pre-Verification - Java class is pre-verified by the byte code pre-verification tool of emulator. When the pre-verification tool is executed, additional attributes are written into the class file, which makes the byte codes verification of CLDC virtual machine execution efficient. Also, the pre-verification tool checks to see if CLDC's restricted functions as finalized method are not used in the application program.

Packaging - Use J2SE SDK archive tool (jar) to package all pre-verified class files and resource files as the JAR file. The usage of jar command is same as the standard usage of jar command in J2SE SDK. In order to minimize the jar file size stored in cell phone, set the compressing option off.

Test in Emulator - It can verify if i-αppli will function as intended by using the emulator environment.

Testing i-αppli using appropriate mobile phone models - Place i-αppli (JAR file, ADF) and download navigation HTML contents on a web site and download it to actual mobile phone. By executing i-αppli on cell phone, verify if all functions as designed. Test i-αppli on all targeted manufacture's cell phones, because depending on cell phones, usable memory size and hardware resource size may be different.

The consecutive pattern of compilation, pre-verification and packaging can be done as an individual tool, but also it can be batched together by using i-αppli build function that the emulator has. For i-αppli build functionality of emulator, refer to i-αppli development environment User's Guide.

15.4 Creating a "Communication To" Application

Client/Server type i-αppli needs to have the server side application for i-αppli to communicate to. Client/Server type i-αppli needs to have the server side application for i-αppli to communicate to.

Also, it is necessary to prepare OBEX application serving as receiving communication, when i-αppli uses OBEX outside connection. In the case of utilizing OBEX outside connection between i-αppli supported cell phones, both OBEX client and OBEX server developments are possible using i-αppli API. . When connecting cell phones to other devices and development of receiving transmission application, refer to OBEX application development environment documents used by receiving side platform.

15.5 Packaging of an i-αppli

As shown in Figure 24, an i-αppli consists of two files, an ADF and a JAR file (containing a Java class file and a resource file).

15.5.1 Creation of ADF

ADF is a text file written in SJIS encoding. This file consists of lines containing a key/value pair separated by a "=". Any text editor can be used to edit the fie. In this file, the following key entries (lines) are specified:

`AppName:` Required Key

This is the name of the application (16 bytes maximum). When i-αppli is displayed as a list in a cell phone, the name indicated here is shown. SJIS can be used to define Japanese names.

`AppVer:` Optional Key

i-αppli version (Maximum 10 bytes). Using this key, the developer can administrate i-αppli version management. Only ASCII characters can be used.

`PackageURL:` Required Key

URL of JAR file's i-αppli (255 bytes maximum). When a relative path is defined, the base directory shows the location of ADF. Only ASCII characters can be used. The host name specified in the URL here must be symbolic (e.g., "www.nttdocomo.co.jp"). You cannot use numerical IP addresses (e.g., "127.0.0.1").

`AppSize:` Required Key

Size of JAR File. Specify integer in bytes. The size of a JAR file is restricted as discussed in Section 3.2.

`AppClass:` Required Key

The main class name used for i-αppli execution (Full package name, Maximum 255 bytes). Only ASCII characters can be used.

`AppParam:` Optional Key

Launch parameter of main class. Parameters are to be separated by spaces (Maximum 255 bytes). Use the `com.nttdocomo.ui.IApplication.getArgs()` method to get these parameters from application program.Only ASCII characters can be used. By default, there is no launch parameter.

`ConfigurationVer(KvmVer):` Optional Key

J2ME Configuration Version. When this key is not specified, it indicates that i-αppli can be operated all versions of CLDC virtual machine. `ConfigurationVer` and `KvmVer` mean the same but both cannot be specified at the same time in one ADF.KvmVer is provided for compatability with DoJa-1.0 Profile. The setting value of this key should follow accordingly.

`CLDC-x.x` (x is any number 0 to 9. This version supports "1.0" or "1.1").

`ProfileVer:` Optional Key (new in DoJa-2.0 profile)

i-αppli runtime environment profile version. When this key is not specified, it indicates that i-αppli can be operated on all versions of i-αppli runtime environments. The setting value of this key should follow accordingly.

`DoJa-x.x` (x is any number 0 to 9)

This version supports the following:

`DoJa-1.0` (503i Series Support)

`DoJa-2.0` (504i Series Support)

`DoJa-2.1` (FOMA02 Model Support)

`DoJa-2.2` (FOMA03 Initial Model Support)

`DoJa-3.0` (505i Series Support)

`DoJa-3.5` (FOMA900i Series Support)

`DoJa-4.0` (FOMA901i Series Support)

`DoJa-4.1` (FOMA902i Series Support)

`DoJa-5.0` (FOMA903i Series, FOMA904i Series Support)

`DoJa-5.1` (FOMA905i Series, FOMA906i Series Support)

`SPsize:` Optional Key

Size of ScratchPad. Specify integer in bytes. Usable ScratchPad size of i-αppli's ScratchPad is subject to the limitations discussed in Section 3.2.When omitted, a ScratchPad is not used.

In DoJa-3.0 Profile, up to 16 ScratchPads can be separated and managed internally. When partitioning a ScratchPad, each byte number is listed separated by comma as the key value of `SPsize` (Do not include a space in front and/or back of comma). To access the ScratchPad, specify the ScratchPad number in URL (0 to 15 at maximum), and the number corresponds to the size list of this key.

(If "`Spsize = 1024,2048`" is written, the ScratchPad size of 0 equals 1024 bytes and the 1st number is 2048 bytes.)

When ScratchPad is partitioned and managed, the total size of each partition should be in the range of size restriction.

`LastModified:` Required Key

i-αppli 's last modified time. Update of i-αppli is done based on this key's contents. Formats are as shown below.

`Dow, DD Mon YYYY HH:MM:SS TimeZone`

`Dow:` day of the week `DD:` day `Mon:` month `YYYY:` year

`HH:` hour `MM:` minutes `SS:` seconds `TimeZone:` time zone

The time zone can be omitted. Because DoJa-1.0 profile does not support the time zone setup, when ADF is shared between DoJa-1.0 Profile compatible mobile phones and DoJa-2.0 profile compatible mobile phones, omit the time zone (ADF created by emulator does not set the time zone).

Time Zone supports "GMT" and "JST". Also, it is possible to indicate the time zone using the time differences like "+0900". When the time zone is not set, the default is "JST".

Example: `Fri, 25 Feb 2000 12:04:25 JST`

UseNetwork: Optional Key

When the network function is used, it specifies communication method. In i-αppli runtime environment, "http" is the only effective value (When using HTTPS, set "http"). If i-αppli does not have the option, network connection can not be demanded.

TargetDevice: Optional Key

It specifies the device name of a target cell phone (maximum 128 bytes). When all types of devices are target, the entry itself is not specified. On this key, some formatted value is set up.

Individual specification of targeted device name. A name identical to that included in the user agent is set for device-name. When there are many multiple targeted devices, they are separated by comma.

Example: `X901i,Y901i,Z901i`

Collective specification of targeted series device names. Like "all901", decide a series name after "all" keyword. When there are many multiple targeted devices, they are separated by comma. Please note that this method cannot be interpreted by cell phones that support only DoJa-1.0 Profile.

LaunchAt: Optional Key

When i-αppli is designed to do automatic booting, specify the launch timing. The first character is the form specifier. In the i-αppli runtime environment, it supports "I" representing intervals as the form specifier.

Time Interval Specification Format (Format Specifier: I)
Format: `I h`

(h: Hour: effective digits are 2, need a space between "I" and number)

Example: `I 6` (Launch every six hours)

The beginning of time interval is when the i-αppli is downloaded to cell phone.

If other applications are running or during conversation at the auto-launch timing, the i-αppli is not auto-launched. On such occasion, the cell phone notifies the user that no i-αppli was auto-launched in the cell phone's specific way.

Depending on the manufacturer, as a function of the cell phone, it may have user registration set up to auto-launch i-αppli at a specific time. When auto-launched with such a function, i-αppli's launching condition is not timer launch (`IApplication.LAUNCHED_FRON_TIMER`) but (`IApplication.Launched_FROM_MENU`).

MyConcierge: Optional Key (new in DoJa-2.0 profile)

Creating a Stand-by Application. When "Yes" is set as a value, the i-αppli is usable as a stand-by application. An i-αppli for which this key is not set cannot be registered in a mobile phone as a stand-by application.

UseTelephone: Optional Key (new in DoJa-2.0 profile)

Preparation for i-αpplis that use telephone call functions. When "call" is set as a value, the i-αppli can not be used for vocal communications (`com.nttdocom.uti.Phone.call()` method). If i-αppli does not have the option, telephone calls can not be used.

UseBrowser: Optional Key (new in DoJa-2.0 profile)

It declared that i-αppli uses browser functions. When "launch" is set as a value, the i-αppli can launch a brower using `com.nttdocom.ui.IApplication.launch()` method. If i-αppli does not have the option, the browser is not launched. Then, when i-αppli is running as stand-by, a browser cannot be launched irrespective of whether this key is set or not.

`LaunchByMail:` Optional Key (new in DoJa-2.0 profile)

It declares that this i-αppli has a permission to be launched. For its value, enter only one sending mail address to permit the launch with ASCII (50 bytes maximum). When the sending address contains as the results of reverse match comparison, it permits the launch of i-αpplis via email. When "`any`" is set in this key, there is no restriction on the sending mail address.

`LaunchByBrowser:` Optional Key (new in DoJa-2.0 profile)

It declares that this i-αppli has a permission to be launched from a browser. For its value, enter only URL of a web page with ASCII (Maximum 255 bytes). When the value contains all URLs of browser as the results of forward match comparison, it permits to launch i-αppli via email. When "`any`" is set in this key, there is no restriction on URL.

When launching an i-appli from the ToruCa Viewer, the `LaunchByToruCa` key should be used instead of the `LaunchByBrowser` key. The target for forward comparison will not be the Web page's URL, but will instead be the URL to retrieve the ToruCa (Details). Besides this, the usage method for the `LaunchByToruCa` key is the same as the `LaunchByBrowser` key.

`AllowPushBy:` Optional Key (new in DoJa-2.0 profile)

It declares that this i-áppli has a permission to be launched from outside interface. There are 2 kinds of launching from outside interface, launching by receiving vTrigger object from the infrared port and by 2 dimensional bar codes upon code recognitions function.

<Launching from vTrigger Object Receiving via Infrared Port>

One of the following style parameter by ASCII for `AllowPushBy` key value (255 bytes maximum including the prefix, "`Irda:`").

`Irda:`<*command*>

In <*command*>, launching commands specified by vTigger that gives permission for i-αppli to be launched. When the key corresponds to the vTrigger object launching commands completely that requested launching against i-αppli, a permission is given to launch i-αppli by the vTrigger object.

A space or tab is not allowed in <*command*>.

<Launching from 2 dimensional barcode via code recognition function>

One of the following style parameter by ASCII for AllowPushBy key value (255 bytes maximum including the prefix, "`Code:`").

`Code:`<*command*>

In <*command*>, launching commands specified by the two-dimensional barcode that gives permission for i-αppli to be launched. When the key corresponds to the 2 dimensional barcode launching commands completely that requested launching against i-αppli, a permission is given to launch i-αppli by the 2 dimensional barcode.

A space or tab is not allowed in <*command*>.

`AppTrace:` Optional Key (new in DoJa-2.0 profile)

This key provides simple debug function on cell phone during i-αppli development phase. When the key is set to "on", the following functions become effective on cell phone.

When uncaught exception happens in i-αppli, it shows what type of exception caused the termination of i-αppli.

Logging output characters from i-αppli standard output or standard error output and can be displayed after i-αppli is terminated. However, the length of the character string that can be logged is limited (to 512 bytes maximum), and display within the range limit is carried out sequentially from the item most recently written. Note that results calling `java.lang.Throwable.printStackTrace()` will not be recorded in this log.

i-αppli information display function becomes detailed mode of JAM.

Upon completion of i-αppli development, delete this key from ADF.

`DrawArea:` Optional Key (new in DoJa-2.0 profile)

It defines this i-αppli's default screen size (i-αppli drawing area size). When cell phone has a large drawing area is executed with i-αppli's default small cell phone drawing area, i-αppli is executed within the specified square range with centered display. No drawing with i-αppli is allowed in the outside part of rectangular (the part's color differs depending on manufactures).

Key values are specified as shown below.

<*width*>x<*height*> (<*width*> is the width, <*height*> is the height, both in pixels; "x" must be lowercase only)

Example: `DrawArea = 120x120`

The minimum settable value for <*width*> is 96 and maximum value is 72 for <*height*>. Also, the area side specified by this key should be smaller than the drawing area of cell phone the i-αppli is to be downloaded (both horizontal and vertical value should be lower than the size of cellular phone for download). When inappropriate values are set, the i-αppli cannot be downloaded.

When this key is not specified, i-αppli is executed using all the drawing area of the cell phone.

[DoJa-5.0]

For DoJa-5.0 Profile and later, depending on the manufacturer, by using the `DrawArea` key of ADF, a display area may be used wider than the default display area (i.e., when not using the `DrawArea` key). The maximum display area size usable by each model will be announced separately by NTT DOCOMO. For i-applis which use a display area larger than that of the default display area it is recommended to lay out screens using only `Canvas` objects, without any `Panels`. This function was implemented for the purpose of using it with a `Canvas` in mind. When used with a `Panel` the screen display and component operations may not operate as the developer intended.

`GetSysInfo:` Optional Key (new in DoJa-2.0 profile)

It declares that this i-αppli refers to the icon information of the cell phone. When i-αppli is downloaded with the key set to "`yes`", the cell phone confirms with the user to permit the i-αppli to grant a permission of icon information. When the i-αppli could not get permission from the user or the key is not set, the icon information is not referenced.

Use `PhoneSystem.getAttribute()` method to obtain icon information.

[DoJa-2.0] / [DoJa-3.0] / [DoJa-5.1]

In DoJa-2.0 profile, referable icon information is the mail message receiving condition. Also, after DoJa-3.0 profile, besides mail receiving condition, it can refer the electric strength, remaining battery level and silent mode setup status.

In DoJa-5.1 Profile, the signal strength resource was replaced by the area information resource. Refer to Chapter 8 for more details.

`LaunchApp:` Optional Key (new in DoJa-3.0 profile)

It declares that other i-αppli can be launched (with launcher mode and link mode) by application linking. When "yes" is set as a value, the i-αppli can launch other applications using `com.nttdocom.ui.IApplication.launch()` method. If i-αppli does not have the option, other i-αppli can not be launched.

`LaunchByApp:` Optional Key (new in DoJa-3.0 profile)

It controls actions when other i-αppli is to be launched via application cooperation. If the value is set to "`deny`", the i-αppli refuses to be launched by other i-αppli in launcher mode. Launching i-αppli by application cooperation, it does not become stand-by launch and the stand-by. i-αppli is regarded as normal launch. Stand-by applications with no assumption of normal launching can prohibit the cooperation launch by using this key.

`AccessUserInfo:` Optional Key (new in DoJa-3.0 profile)

It declares that it is an i-αppli accessible to cell phone's user data area via application cooperation. When "`yes`" is set as a value, the i-αppli can execute following user data operations.

- Address Book entries, Address Book group registrations
- Bookmarks registrations
- Schedule data
- Registration, selection and getting images

GetUtn: Optional Key (new in DoJa-2.1 profile)

This key declares that the i-αppli will reference the unique identification information of the mobile phone. The identification number within the mobile phone itself can be referenced if this key is specified as "terminalid" and the identification number within the UIM card can be referenced if this key is specified as "userid". To reference both the identification information of the mobile phone and the identification information of the UIM card, specify both of these values separated with a comma (no space before or after the comma).

When an i-αppli with this key specified is downloaded, the mobile phone asks the user to confirm permission for the i-αppli to access the unique identification information. If this key is not specified in the i-αppli or the user does not grant the i-αppli permission, unique identification information cannot be referenced. Also, UIM identification information cannot be retrieved from mobile phones that are not equipped with a UIM card.

IletPreserve: Optional Key (new in DoJa-3.0 profile)

It declares that soon downloaded and launched i-αppli doesn't permit to store itself to cell phone. When "deny" is specified, user cannot save the i-αppli to his/her cell phone.

When this key is not set, when launch after download i-αppli is finished, the cell phone prompts the user to confirm if saved as regular i-αppli on his/her cell phone.

For details regarding i-αppli that launch immediately after download, see chapter 16.

AppIcon: Optional Key (Newly added in DoJa-4.1 Profile for IC applications, applies to non-IC applications as well from DoJa-5.0 Profile)

Depending on the manufacturer, some mobile phone models may have a feature to display an i-appli's own unique icon in addition to its name in the phone's native i-appli list screen (including the IC Card menu). The AppIcon key is used to specify the image to use as this icon on models which support this feature. (For i-applis that do not have an icon, the default system icon will be displayed.).

The image file for the icon must be placed in the root directory of the application's Jar file. In the AppIcon key specify this image file and its filename in the following format:

```
AppIcon = <filename1> [,<filename2>]
```

Specify the filename of a 48x48 pixel size image file for filename1, and if applicable specify the filename of a 96x96 pixel size image file for filename2. (Do not put any spaces before or after commas.) Depending on the manufacturer, the supported image size for icons is divided into the two sizes noted above. But, if two file names are specified in the AppIcon key representing these two different sized icon images, the size that is appropriate for the model that i-appli was downloaded on will automatically be selected.

You can use GIF87a or GIF89a (excluding animated GIFs and interlaced GIFs) or JPEG image data as icon data. Only ASCII characters may be used in the file names. Path separators such as "¥" or "/" cannot be used.

[DoJa-5.1]

DoJa-5.1 Profile brought support for using JPEG format data as icons. In DoJa-5.0 Profile or earlier, only GIF format image data can be used for icons.

AppMainTitle: Optional Key (new in DoJa-5.1 profile)

In DoJa-5.1 Profile (FOMA905i series) and later, a graphical display function was added in addition to the traditional list display format for the display style of the mobile phone's i-appli list display screen. On the graphical display style i-appli list screen, a larger icon than that in the AppIcon key (160x160 pixels) can be displayed as the main title image when the i-appli is selected from the i-appli list via user operation. (i-applis which do not have a main title image will display a default image provided by the system instead.)

The image file for the main title image must be placed in the root directory of the application's Jar file. In the AppMainTitle key specify this image file and its filename in the following format:

```
AppMainTitle = <filename>
```

Specify the filename of a 160x160 pixel image file for the <filename>.

You can use GIF87a or GIF89a (excluding animated GIFs and interlaced GIFs) or JPEG image data as main title image data. Only ASCII characters may be used in the file names. Path separators such as "¥" or "/" cannot be used.

`DeniedMultiApp:` Optional Key (new in DoJa-5.1 profile)

Some FOMA mobile phones can launch multiple device applications (i.e., the Browser, the phone call function) at the same time. Some models support the ability to launch both an i-appli and the music player or both an i-appli and digital television so that the user can listen to music or the television while using an i-appli. However, the music player and digital television functions take considerable CPU power to run alone. If an i-appli is being used together with these functions, sufficient CPU power may not be able to be provided to the i-appli. (In this situation the user may notice that a problematic i-appli may be running slower than normal or that the operation speed looks unstable or jerky.)

To solve this problem, DoJa-5.1 Profile has added the `DeniedMultiApp` key to block other functions such as the music player or digital TV from being run simultaneously with an i-appli that requires full use of the CPU's power.The `DeniedMultiApp` key is defined in the following format:

```
DeniedMultiApp = <application_name> [,<application_name> ... ]
```

Specify the name or names of the applications you want to block from running simultaneously with your i-apply in <application_name>. Under the current profile, two application names are defined: `music` (music player), and `dtv` (digital television). You can also specify "`all`" to block both the music player and digital TV functions.

It is not recommended to use this key in download and immediate launch i-applis which use the `IletPreserve` key to block saving to the mobile phone. If this key is used in a download and immediate launch i-appli, when the i-appli is downloaded it will not be able to launch if the functions it blocked from running simultaneously via the DeniedMultiApp key are already running at the time of download.

Examples of ADF are shown below: As shown below, one key entry should be written in one line. The end of each line should be CR LF (0x0d0a) including the last line.

```
AppName = Banking Demo
ConfigurationVer = CLDC-1.0
ProfileVer = DoJa-3.0
AppClass = banking.BankingDemo
AppVer = 1.0
PackageURL = http://aserver.com:8080/BankingDemo.jar
AppSize = 17556
LastModified = Fri, 25 Feb 2000 12:04:25
```

15.5.2 Creation of JAR File

JAR is the standard Java archive function to compressed resource files and class files structuring Java application. In JAR file, the total size reduces 40%-50% compared to the size before compression (the compression rate may vary depending on the contents of JAR file).

i-αppli is distributed to cell phone in the packaging form to JAR file. The advantage is not only the reduction of necessary i-αppli storage size in cell phone, but also the download time can be shorter. Packaged each file in JAR file is unpacked as needed by JAM.

15.6 Testing i-αpplis

i-αppli test can be conducted by emulator and using actual i-αppli support cell phones.

At the beginning of development phase, it is more efficient to do i-αppli test and debug using the emulator and cost effective. The emulator enables to debug interface, different kind of input/output and application logic and the results of corrected application program can be confirmed immediately on PC.

However, a complete emulation of hardware operation cannot be done by the emulator such as execution speed and network bandwidth of the program. Also, actual hardware has specific models' restriction items like memory storage. Therefore, the final testing should be done on an actual i-αppli supporting cell phone to confirm the adequacy of i-αppli's actions.

For i-αppli emulator, refer to i-αppli development environment User's Guide.

There are some restrictions regarding i-αppli testing in the emulator environment. In the emulator, though the execution environment is provided for i-αppli, such i-αppli management function (function JAM has) that is equipped in actual cell phone can not be emulated. For example, one of the behaviors which cannot be executed is update of i-αppli. This is because JAM's i-αppli management function is not implemented in the emulator. The following shows other behaviors that cannot be tested by the emulator.

- Regarding `LastModified`, `TargetDevice`, `LaunchAt`, there is no function to emulate actual behaviors.
- By the emulator, the consistency between CLDC virtual machine version and `ConfigurationVer` key (`KvmVer` key) cannot be checked.
- Because the emulator executes i-αppli JAR file after reading the development computer hard disk (in the project directory of emulator), the values specified by `PackageURL` is not utilized. When actually distributing i-αpplis to cell phones, it is necessary to set the correct URL in `PackageURL` when downloading.

When installing i-αpplis on cell phones for testing using physical cell phones, it is necessary to place the download page HTML, i-αppli, ADF, and JAR file on a web server connected to internet. Refer to Chapter 16 for more details.

Notes:

- In the i-αppli development environment supporting DoJa-1.0 profile, it is necessary to use MIDI format file for media sound data. Therefore, when developing i-αppli using media sound aupporting Doa-1.0 profile, when conducting a testing using actual cell phone, it is necessary to convert MIDI format file to MFi format file. In some cases, it may necessary to change resource URL (extension) specified in i-αppli source codes.

  In the i-αppli development environment using DoJa-2.0 or above profile, it is possible to use MIDI format file and MFi format file as media sound data.

- If i-αppli has a client/server structure, it is necessary for the server side application to be appropriately deployed when testing is done using the Web server.

Chapter 16
Distributing i-applis

This chapter describes how to setup your web server in order to distribute i-αpplis.

16.1 Web Server Structure for Distribution of i-αpplis

As described in the previous chapter, i-αppli consists of ADF (.jam file) and JAR file (.jar file). When downloading ADF from the web server, the file must be referenced by `OBJECT` tag of i-mode HTML. The following describes a descriptive example of a tag in a HTML file for launching an i-αppli.

```
<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-appli”>
</OBJECT>
To download:
Click <A ista=”#application.declaration” href=”notapplicable.html”>here</A>.
```

The following describes a descriptive example of a tag in an HTML file for downloading an i-αppli.

- The `A` tag and the `ijam` attribute

  The `A` tag is used to refer to the `OBJECT` tag that corresponds to the i-αppli to be downloaded. When a user selects this A tag, an i-αppli's download process is initiated.

  For the `ijam` attribute of the `A` tag, specify the name that is specified in the id attribute of the `OBJECT` tag. `ijam` attribute is a new attribute created for i-αppli supported mobile phones; therefore, regular i-mode mobile phone cannot interpret it.By adding `href` atribute to the `A` tag, the regular i-mode models can interpret href attribute and i-αppli supported models can interpret `ijam` attribute. Note that the ista attribute to launch i-αppli from a browser cannot be effective with ijam attribute simultaneously. If `ijam` and `ista` attributes are set simultaneously, DoJa-2.0 profile supporting mobile phones always interpret ista attribute and `ijam` attribute is ignored (DoJa-1.0 profile supporting mobile phones cannot interpret `ista` attribute; therefore, `ijam` is interpreted instead).

  Note that for the `A` tag that has the `ijam` attribute, set the `href` attribute as a pair.

  [DoJa-2.0]

  Furthermore, launching i-αppli and `ista` attribute of `A` tag were added in DoJa-2.0 profile. Refer to Chapter 11 for more details.

- The `OBJECT` tag

  The `OBJECT` tag is used to refer to an ADF that supports an i-αppli.For the id attribute of the `OBJECT` tag, specify the name (a unique name in an HTML file), which is referenced in the `ijam` attribute of the `A` tag. Furthermore, for the `data` attribute, specify an URL that indicates the location of the ADF. For the `type` attribute, specify the content type of data (in this case, ADF) indicated by the `data` attribute. The content type of the ADF is "`application/x-jam`".

JAR file of i-αppli is referenced by the PackageURL key specified within ADF.

For the inside of the OBJECT tag, the following PARAM tag can be specified.

- The PARAM tag

The PARAM tag is used to specify a parameter that can be obtained by the Iapplication.getParameter() method when launching an i-αppli for the first time after it is downloaded(*1).

A maximum of 16 PARAM tags can be included in the OBJECT tag. Also, the total length of the values of name and value attributes is limited up to 255 bytes. For the values of name and value attributes, SJIS Japanese text can be specified respectively.

(*1) If the first time the i-appli is launched after being downloaded is via a linked launch (i.e., launched from the browser, mailer, an external device, or another i-appli), the contents of the PARAM tag specified here will not be passed to the i-appli. If the initial execution is by application linking, the parameter passed from the launching application becomes effective and passed to launched i-αppli. Refer to Chapter 11 for more details about application linking.

[DoJa-2.0]

The parameter setup by PARAM tag and IApplication.getParameter() method is added to DoJa-2.0 profile.

i-αppli supported mobile phones confirm ADF file contents type via type attribute of OBJECT tag. Therefore, on the side of Web server, it is not do any configuration of contents type supporting the ADF extension.

[DoJa-2.0]

There is no restriction regarding ADF extension in DoJa-2.0 profile. On the other hand, ADF extension needs to be ".jam" on DoJa-1.0 (When the URL entity of ADF is CGI, the extension of the CGI should be ".jam").

[DoJa-5.0]

For DoJa-5.0 Profile and later, JAR files over 100KB can be downloaded due to the definition described in Section 3.2. However, in order for the mobile phone to be able to download a JAR file that exceeds 100 KB in size, the Web server used to distribute the i-appli must be able to support partial GET range requests in bytes. Note that JAR files that exceed 100 KB in size cannot be downloaded from Web servers which give the partial GET response "206 Partial content" in response to a range request.

Be particularly aware of this fact if CGI is being used in any way to download the JAR file.

In DoJa-5.0 Profile or later on mobile phones which support ToruCa version 2.0 format, you can use the tags discussed here in this chapter in the HTML content section of ToruCa (Details) to launch an i-appli from the ToruCa Viewer. In this case you should not specify a relative URL for the ADF URL in the data attribute of the OBJECT tag.

16.2 Web Server Structure for Server Side Execution of i-αpplis

When the client/server type i-αppli is executed, it is necessary to do specific configurations the web server depending on the adoptive technology on the server side application. For example, when Java servlet is used on the server side application, it is necessary to configure and install Java servlet engine on the Web server.

Refer to the documents applied to server side application and Web server documents to be used, and apply adequate configurations on the Web server.

16.3 Downloading to i-αppli Compatible Mobile Phones of a Specific Manufacturer

If i-αppli is developed using i-application functions, i-αppli extension functions or strict screen size requirements, it works only on manufacture specific mobile phones as the developer intended. It is expected that this type of i-αppli will be installed appropriately only for the targeted models.

In order to properly control i-appli downloads across different models, the following features can be used on i-appli compatible mobile phones.

The HTTP request from a mobile phone contains `User-Agent` request header to distinguish different mobile phone models. From the Web server side, when displaying the downloading HTML of i-αppli, it can navigate to show only appropriate i-αppli based on the `User-Agent` contents.

From the mobile phone side, it judges if the i-αppli can be downloaded for the model by referencing the ADF `TargetDevice` key. For the i-αppli for specific models, inappropriate download to non-matching mobile phone models can be prevented by setting the ADF `TargetDevice` key.

16.4 Download and Immediate Launch i-αpplis

In DoJa-3.0 or later profile, launch functionality immediately after download is supported so that downloading and launching an i-αppli can be performed with a single click from the user. Though i-αppli launch immediately after download requires no user confirmation, there are some restrictions on some functions thatconfirmation is required.

The download and launch i-αppli cannot call APIs listed below. When such an API is called, an exception is thrown.

- Permission function evaluated by ADF `GetSysInfo` key (refer to Icon information)
- Permission function evaluated by ADF `GetUTN` key (refer to device ID)
- Permission function evaluated by ADF `AccessUserInfo` key (access to user data area)
- Permission function evaluated by ADF `LaunchApp` key (i-αppli cooperation launch)
- Permission function evaluated by ADF `GetUTN` key (refer to device ID)
- Using ScratchPad
- Linked launch i-αppli update function (JAM) from i-αppli

When distributing i-αppli as download and immediate launch i-αppli, set the `ilet` attribute instead of `ijam` attribute specific used for downloading i-αppli application download link (A tag). As the `ijam` attribute, `ilet` attribute should be used with `href` attribute.

The following shows a descriptive example of a tag in a HTML file for downlod and launch i-αppli for distribution.

```
<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-appli">
</OBJECT>
```
Click `<A ista="#application.declaration" href="notapplicable.html">here</A>`
to do the download and immediate launch i-αppli at once.

The parameters using `PARAM` tags can be received even in the download and immediate launch i-αppli.

[DoJa-5.0]

In DoJa-5.0 Profile, the maximum size of an i-appli was increased to a total of 1024 KB for both the JAR file and the ScratchPad size together. However, depending on the manufacturer the JAR file size may be limited to 100 KB and the ScratchPad size limited to 400 KB for download and immediate launch i-appli, which is the same as the older profiles.

Notes:

- The launch from the download and immediate launch mode equals to the i-αppli launch from a browser via application cooperation. That is why the download and immediate launch i-αppli must have appropriate `LaunchByBrowser` of ADF setup so that the i-αppli can do the coopration launch from the i-αppli download HTML.See Section 15.5.1 for more information about the ADF's `LaunchByBrowser` key.
- There are some restrictions on usable functions of user confirmation and prohibited settings the download and immediate launch i-αppli, which functions could be used in regular i-αppli download; however, HTTP(S) transmission can be used as exception. The HTTP(S) transmission permission confirmation is done upon initial HTTP(S) transmission in the download and immediate launch i-αppli, not when i-αppli is downloaded.
  During the permission confirmation is executed, the execution of download and immediate launch i-αppli is interrupted.
- αppli launched by download and immediate launch mode is normal launch even if it is a stand-by application.
- When download and immediate launch mode execution is terminated normally, a confirmation is displayed if user wants to save the i-αppli to the mobile phone. When "save" is selected, the download and immediate launch i-αppli is saved as regular i-αppli, then user confirmation such as HTTP(S) communication permission setup is conducted according to ADF contents (at the time, the usage limitation of functions described in this section are released).
  Using ADF `IletPreserve` key, the developer can set that the download and immediate launch i-αppli mobile phone should not be saved.See Section 15.5.1 for more information about the ADF's `IletPreserve` key.
  If the download and immediate launch i-αppli is terminated by making a browser cooperation launch, the process for i-αppli saving is cancelled and the i-αppli is not saved.
- On the download and immediate launch i-αppli, the launching type obtained by `IApplication.getLaunchType()` method becomes `IApplication.LAUNCHED_AS_ILET`. Also, when the user saves the download and immediate launch i-αppli after termination, if the initial launch is a regular launch (launch by user operation via i-αppli list), the launch type becomes `IApplication.LAUNCHED_AFTER_DOWNLOAD`.
- Trusted i-αppli cannot be downloaded as the download and immediate launch i-αppli.

Appendix A
DoJa-5.x and DoJa-5.x LE

FOMA mobile phone of DoJa-5.x profile generation is prepared with additional model series lineups for entry, besides the 90x series (standard model) line up. FOMA mobile phone for entry level is loaded with i-αppli runtime environment supporting DoJa-5.x LE (Limited Edition) profile which is the summarized (a part of function is omitted) version of DoJa-5.x.

The main theme of DoJa-5.x LE profile is to make relative contents of PDC mobile phone usable for entry level FOMA mobile phone. and may have some restrictions on DoJa-4.x profile functions and functions added with DoJa-5.x profile.

Regarding DoJa profile loaded on each type including DoJa-5.x profile and DoJa-5.x LE will be announced by NTT DOCOMO as needed.

This section describes the difference between the DoJa-5.x Profile and the DoJa-5.x LE Profile.

3D Graphics Rendering

The i-appli standard API's 3D graphics rendering functions (the `com.nttdocomo.ui.graphics3d` package and its sub-packages) are optional in DoJa-5.x LE Profile compatible mobile phones, with some models not including these functions at all. But the high level 3D graphics rendering function regulated i-αppli extension API (`com.nttdocomo.opt.ui.j3d` package) is also specified by DoJa-5.xLE profile.

3D Sound Control Functions

The 3D sound control functions are optional in DoJa-5.x Profile compatible mobile phones, with some models not including these functions at all.

JAR file size, Size limitation of ScratchPad

On DoJa-5.x Profile FOMA mobile phones, the maximum size of an i-appli was increased to a total of 1024 KB for both the JAR file and the ScratchPad size together. However, for DoJa-5.x LE Profile this is limited to an i-appli size of 30 KB for the JAR file and 200 KB for the ScratchPad size for all models in this profile. (Some models may support a larger size than this within the range of the non-LE profile size restrictions.)

Appendix B
How the 2-in-1 Function Effects i-applis

On 904i series or later mobile phones, one mobile phone can be assigned two phone numbers and two i-mode e-mail addresses which can be switched between freely depending on how the user uses the device. This feature is called the 2-in-1 function and is available on some 904i series phones. (The 2-in-1 function is also sometimes called the HyperMulti Number function or the HyperMulti function.)

To use the 2-in-1 function, a user must sign a contract with NTT DOCOMO to use the 2-in-1 function in addition to their standard phone contract. When this contract is made the user can use another phone number/i-mode e-mail address pair (Number B/E-Mail B) in addition to their standard phone number and i-mode e-mail address (Number A/E-Mail A).

The 2-in-1 function is one that is heavily integrated with the mobile phone's calling/mail functions and operations, phone book/call history, and other user data management. Therefore, on mobile phones which have the 2-in-1 function set to be enabled, some portions of the i-appli API may be affected. This appendix section will discuss the effects the 2-in-1 function has on i-applis.

Hereinafter, the state in which the 2-in-1 function is set to enabled on the mobile phone will be referred to as the active state, while the state where the function is set to disabled will be referred to as the inactive state. In the active state, different modes have been implemented to control the viewing of user data. There are three different active state modes:

A Mode	:	Mode which allows user data related to Number A/E-Mail Address A to be viewed
B Mode	:	Mode which allows user data related to Number B/E-Mail Address B to be viewed
Dual Mode	:	Mode which allows user data related to both A and B to be viewed (also called "Common Mode")

In 2-in-1 Active A Mode, call history from B Mode, mail addressed to Address B, or any phonebook data saved with an attribute denoting it for B Mode use cannot be viewed. In 2-in-1 Active B Mode, call history from A Mode, mail addressed to Address A, or any phonebook data saved with an attribute denoting it for A Mode use cannot be viewed.

Currently when using the 2-in-1 service, mail addressed to Address B will not be sent to the mobile phone in normal i-mode mail format. Instead, this mail can be viewed in web mail format. The user will be notified that a mail has arrived for Address B by way of a notification e-mail (i-mode mail) to the mobile phone addressed to Address B.

Furthermore, in 2-in-1 active B Mode, no i-mode mail or SMS can be sent from the mobile phone. In 2-in-1 active B Mode, the user must send mail from a web-based interface just like when they view their mail.

<How the 2-in-1 Function Affects i-applis>

- Stand-by Applications

  In 2-in-1 inactive mode or 2-in-1 active A Mode, standby launch of standby applications is supported; however, this is not supported in 2-in-1 active B Mode or 2-in-1 active Dual Mode.

  If a standby application is running and the state or mode of the 2-in-1 function is changed, that standby application will temporarily close. (If standby applications are supported in the mode or state changed to, the standby application will be launched again upon the recovery of standby mode.)

- User Data Viewing Restrictions Based on the State or Mode

  The data which can be viewed by calling the My Profile management function (`OwnerProfile` class: i-appli API) changes depending on the current state or mode whlie the i-appli is running. In 2-in-1 inactive state, 2-in-1 active Dual Mode, or 2-in-1 active A Mode all My Profile information related to Number A/Address A can be viewed. In 2-in-1 active B Mode all My Profile information related to Number B/Address B can be viewed.

- User Data Registration Restrictions Based on the State or Mode

  When using the `PhoneBook.addEntry()` method from an i-appli to add an entry to the phonebook, in 2-in-1 inactive mode, 2-in-1 active Dual Mode, or 2-in-1 active A Mode, that phonebook entry will be saved as phonebook data for use in A Mode only (i.e., that data cannot be viewed in B Mode as phonebook data). In 2-in-1 active B Mode, that phonebook entry will be saved as phonebook data for use in B Mode only.

  Phonebook entries saved as described above can be modified by the user via the mobile phone's native configuration functions change the mode they can be viewed in or even so that they can be viewed in any mode at all.

- Retrieving the Reason for Suspension via the `IApplication.getSuspendInfo()` Method

  By using the `IApplication.getSuspendInfo()` method, you can retrieve the reason why the last application to suspend did so as well as information regarding what happened while it was suspended. Receive events such as "call received" or "mail received" events are included in the reason and/or events surrounding a suspension. However, in 2-in-1 active A Mode, receive events for Number B/Address B cannot be retrieved through this method. Likewise, in 2-in-1 active B Mode, receive events for Number A/Address A cannot be retrieved through this method either.

Glossary

API	Application Programming Interface
Applet	A small program sent to a user with a Web page. An i-αppli is not an applet.
bps	bits per second. The number of bits that can be transferred per second.
CGI	Common Gateway Interface
CHTML	Compact HTML
CLDC	Connected Limited Device Configuration
HTTP	Hypertext Transfer Protocol
HTTPS	Secure Hypertext Transfer Protocol
IDE	Integrated Development Environment.
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JAM	Java Application Manager
JAR	An abbreviation for Java Archive
JDK	Java Development Kit
JNI	Java Native Interface
JVMS	Java Virtual Machine Specification
MIME	Multi-Purpose Internet Mail Extensions. Extending Internet email protocols and using this protocol allows various data files to be sent on the Internet.
Servlet	A servlet. A mechanism for developing a server-side Java application in a module format for dynamically generating Web contents.
SSL	Secure Sockets Layer
UI	User Interface
Unicode	A 16-bit code to represent characters.
URL	Uniform Resource Locator. An address to a file (resource) that is accessible on the Internet.