Mobile Information Device Profile

for Java™ 2 Micro Edition

Version 2.0

JSR 118 Expert Group
jsr-118-comments@jcp.org

Java Community Process

Mobile Information Device Profile Specification (“Specification”)
Version: 2.0
Status: FCS
Release: November 5, 2002

Copyright 2002 Sun Microsystems, Inc. and Motorola, Inc.
All rights reserved.

NOTICE; LIMITED LICENSE GRANTS

Sun Microsystems, Inc. (“Sun”) hereby grants you a fully-paid, non-exclusive, non-transferable, worldwide, limited license (without the right to sublicense), under Sun’s applicable intellectual property rights to view, download, use and reproduce the Specification only for the purpose of internal evaluation, which shall be understood to include developing applications intended to run on an implementation of the Specification provided that such applications do not themselves implement any portion(s) of the Specification.

Sun also grants you a perpetual, non-exclusive, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights or patent rights it may have in the Specification to create and/or distribute an Independent Implementation of the Specification that: (i) fully implements the Spec(s) including all its required interfaces and functionality; (ii) does not modify, subset, superset or otherwise extend the Licensor Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the Licensor Name Space other than those required/authorized by the Specification or Specifications being implemented; and (iii) passes the TCK (including satisfying the requirements of the applicable TCK Users Guide) for such Specification. The foregoing license is expressly conditioned on your not acting outside its scope. No license is granted hereunder for any other purpose.

You need not include limitations (i)-(iii) from the previous paragraph or any other particular “pass through” requirements in any license You grant concerning the use of your Independent Implementation or products derived from it. However, except with respect to implementations of the Specification (and products derived from them) that satisfy limitations (i)-(iii) from the previous paragraph, You may neither: (a) grant or otherwise pass through to your licensees any licenses under Sun’s applicable intellectual property rights; nor (b) authorize your licensees to make any claims concerning their implementation’s compliance with the Spec in question.

For the purposes of this Agreement: “*Independent Implementation*” shall mean an implementation of the Specification that neither derives from any of Sun’s source code or binary code materials nor, except with an appropriate and separate license from Sun, includes any of Sun’s source code or binary code materials; and “*Licensor Name Space*” shall mean the public class or interface declarations whose names begin with “java”, “javax”, “com.sun” or their equivalents in any subsequent naming convention adopted by Sun through the Java Community Process, or any recognized successors or replacements thereof.

This Agreement will terminate immediately without notice from Sun if you fail to comply with any material provision of or act outside the scope of the licenses granted above.

TRADEMARKS

No right, title, or interest in or to any trademarks, service marks, or trade names of Sun or Sun’s licensors is granted hereunder. Sun, Sun Microsystems, the Sun logo, Java, J2ME, and the Java Coffee Cup logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

DISCLAIMER OF WARRANTIES

THE SPECIFICATION IS PROVIDED “AS IS”. SUN MAKES NO REPRESENTATIONS OR WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT, THAT THE CONTENTS OF THE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE OR THAT ANY PRACTICE OR IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADE SECRETS OR OTHER RIGHTS. This document does not represent any commitment to release or implement any portion of the Specification in any product.

THE SPECIFICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION THEREIN; THESE CHANGES WILL BE INCORPORATED INTO NEW VERSIONS OF THE SPECIFICATION, IF ANY. SUN MAY MAKE IMPROVEMENTS AND/OR CHANGES TO THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THE SPECIFICATION AT ANY TIME. Any use of such changes in the Specification will be governed by the then-current license for the applicable version of the Specification.

LIMITATION OF LIABILITY

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION, LOST REVENUE, PROFITS OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO ANY FURNISHING, PRACTICING, MODIFYING OR ANY USE OF THE SPECIFICATION, EVEN IF SUN AND/OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You will indemnify, hold harmless, and defend Sun and its licensors from any claims arising or resulting from: (i) your use of the Specification; (ii) the use or distribution of your Java application, applet and/or clean room implementation; and/or (iii) any claims that later versions or releases of any Specification furnished to you are incompatible with the Specification provided to you under this license.

RESTRICTED RIGHTS LEGEND

U.S. Government: If this Specification is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in the Software and accompanying documentation shall be only as set forth in this license; this is in accordance with 48 C.F.R. 227.7201 through 227.7202-4 (for Department of Defense (DoD) acquisitions) and with 48 C.F.R. 2.101 and 12.212 (for non-DoD acquisitions).

REPORT

You may wish to report any ambiguities, inconsistencies or inaccuracies you may find in connection with your use of the Specification ("Feedback"). To the extent that you provide Sun with any Feedback, you hereby: (i) agree that such Feedback is provided on a non-proprietary and non-confidential basis, and (ii) grant Sun a perpetual, non-exclusive, worldwide, fully paid-up, irrevocable license, with the right to sublicense through multiple levels of sublicensees, to incorporate, disclose, and use without limitation the Feedback for any purpose related to the Specification and future versions, implementations, and test suites thereof.

(LFI#119098/Form ID#011801)

Contents

Overview 1

Mobile Information Device Profile, v2.0 (JSR-118) 1

Over The Air User Initiated Provisioning Specification 11

Security for MIDP Applications 23

Trusted MIDlet Suites using X.509 PKI 29

java.lang 35

IllegalStateException 37

java.util 39

Timer 40

TimerTask 46

javax.microedition.io 49

CommConnection 55

Connector 60

HttpConnection 65

HttpsConnection 85

PushRegistry 89

SecureConnection 100

SecurityInfo 103

ServerSocketConnection 105

SocketConnection 108

UDPDatagramConnection 113

javax.microedition.lcdui 115

Alert 128

AlertType 136

Canvas 139

Choice 155

ChoiceGroup 166

Command 175

CommandListener 183

CustomItem 184

DateField 201

Display 205

Displayable 218

Font 223

Form 231

Gauge 240

Graphics 247

Image 270

ImageItem 281

Item 287

ItemCommandListener 300

ItemStateListener 301

List 303

Screen ... 315
Spacer ... 316
StringItem ... 319
TextBox ... 323
TextField ... 330
Ticker ... 345

javax.microedition.lcdui.game ... 347

GameCanvas ... 349
Layer ... 356
LayerManager ... 360
Sprite ... 365
TiledLayer ... 382

javax.microedition.media ... 391

Control ... 396
Controllable ... 397
Manager ... 399
MediaException ... 404
Player ... 406
PlayerListener ... 416

javax.microedition.media.control ... 421

ToneControl ... 422
VolumeControl ... 428

javax.microedition.midlet ... 431

MIDlet ... 444
MIDletStateChangeException ... 450

javax.microedition.pki ... 453

Certificate ... 455
CertificateException ... 458

javax.microedition.rms ... 463

InvalidRecordIDException ... 469
RecordComparator ... 471
RecordEnumeration ... 473
RecordFilter ... 478
RecordListener ... 479
RecordStore ... 481
RecordStoreException ... 493
RecordStoreFullException ... 495
RecordStoreNotFoundException ... 497
RecordStoreNotOpenException ... 499

The Recommended Security Policy for GSM/UMTS Compliant Devices ... 501

Almanac ... 517

Index ... 543

CHAPTER 1

Overview

Description

Mobile Information Device Profile, v2.0 (JSR-118)

JCP Public Draft Specification

Java 2 Platform, Micro Edition™

Copyright 2000,2002, Motorola, Inc. and Sun Microsystems, Inc. ALL RIGHTS RESERVED.

Preface

These documents define the *Mobile Information Device Profile (MIDP) v2.0 Specification* for the Java 2 Platform, Micro Edition (J2ME™).

A *profile* of J2ME defines device-type-specific sets of APIs for a particular vertical market or industry. Profiles are more exactly defined in the related publication, *Configurations and Profiles Architecture Specification*, Sun Microsystems, Inc.

Revision History

Date	Version	Description
1-September-2000	MIDP 1.0 Specification	Final MIDP 1.0 specification
23-October-2001	MIDP 2.0, EG Draft 2	First complete draft published to the Expert Group
6-November-2001	MIDP 2.0, EG Draft 3	Incorporated changes made during 30-31 October Expert Group meeting
20-November-2001	MIDP 2.0, EG Draft 4	Incorporated changes discussed on EG mailing lists.
18-December-2001	MIDP 2.0, EG Draft 5	Incorporated changes made during 5-6 December Expert group meeting and EG mailing lists. Published for Community Review
12-February-2002	MIDP 2.0, EG Draft 6	Incorporated changes made during Community Review and internal EG mailing lists.
12-March-2002	MIDP 2.0, EG Draft 7	Incorporated changes made during 20-21 February Expert group meeting and EG mailing lists. Published for Public Review
09-April-2002	MIDP 2.0, EG Draft 8	Incorporated changes made during 26 March Expert group meeting and EG mailing lists.

23-April-2002	MIDP 2.0, EG Draft 9	Incorporated changes discussed in the EG mailing lists.
9-May-2002	MIDP 2.0, EG Draft 10	Incorporated changes discussed in the EG mailing lists.
29-May-2002	MIDP 2.0, EG Draft 11	Incorporated changes discussed in the EG mailing lists.
11-June-2002	MIDP 2.0, EG Draft 12	Incorporated changes discussed in the EG mailing lists. This draft is considered final in terms of functionality in all areas. Further clarifications and editorial changes may be made in one more revision, but only if necessary.
15-July-2002	MIDP 2.0, EG Draft 13	Incorporated editorial changes and clarifications discussed in the EG mailing lists. This is final draft candidate 1.
02-August-2002	MIDP 2.0, EG Draft 14	Incorporated editorial changes and clarifications from the RI and TCK teams and EG. This draft is being submitted to the PMO as Proposed Final Draft.
04-September-2002	MIDP 2.0, EG Draft 15	Incorporated minor editorial changes and clarifications from the RI and TCK teams and EG.
05-November-2002	MIDP 2.0, Final Specification	Incorporated minor changes to the Security Policy Appendix, fixed an incorrect IETF URL, corrected MIDlet.platformRequest() method signature, finalized copyright co-ownership, and incorporated final license.

Who Should Use This Specification
This document is targeted at the following audiences:

- The Java Community Process (JCP) expert group defining this profile
- Implementers of the MIDP
- Application developers targeting the MIDP
- Network operators deploying infrastructure to support MIDP devices

How This Specification Is Organized
This specification is contained in this HTML file and the following related documents:

- JavaDoc API Documentation
- OTA User Initiated Provisioning Specification
- Security for MIDlet suites
- The Recommended Security Policy for GSM/UMTS Compliant Devices

There are requirements listed both in this document and the API documentation. Where there are conflicts, the requirements listed in this document override the API documentation.

Related Literature

- *The Java Language Specification, Second Edition* by James Gosling, Bill Joy, and Guy L. Steele. Addison-Wesley, June 2000, ISBN 0-201-31008-2
- *The Java Virtual Machine Specification (Java Series), Second Edition* by Tim Lindholm and Frank Yellin. Addison-Wesley, 1999, ISBN 0-201-43294-3
- Connected, Limited Device Configuration (JSR-30), Sun Microsystems, Inc (http://jcp.org/jsr/detail/30.jsp).
- Mobile, Information Device Profile (JSR-37), Sun Microsystems, Inc (http://jcp.org/jsr/detail/37.jsp).

- Connected, Limited Device Configuration 1.1 (JSR-139), Sun Microsystems, Inc (http://jcp.org/jsr/detail/139.jsp).

Report and Contact
Your comments on this specification are welcome and appreciated. Please send your comments to:

jsr-118-comments@jcp.org

Definitions
This document uses definitions based upon those specified in RFC 2119 (http://www.ietf.org/rfc/rfc2119.txt).

Specification Terms

Term	Definition
MUST	The associated definition is an absolute requirement of this specification.
MUST NOT	The definition is an absolute prohibition of this specification.
SHOULD	Indicates a recommended practice. There may exist valid reasons in particular circumstances to ignore this recommendation, but the full implications must be understood and carefully weighed before choosing a different course.
SHOULD NOT	Indicates a non-recommended practice. There may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
MAY	Indicates that an item is truly optional.

Contributors
This specification was produced by the JSR-118 Expert Group, as a part of the Java Community Process. The following companies and individuals, listed in alphabetical order, were members of the Expert Group:

- Companies:
    - •4thpass Inc
    - •AGEA Corporation
    - •Alcatel
    - •Aplix Corporation
    - •AromaSoft Corp
    - •Baltimore Technologies
    - •CELLon France
    - •Distributed Systems Technology Centre
    - •Elata PLC
    - •Esmertec
    - •Espial Group Inc

- France Telecom / Orange
- Fujitsu Limited
- German Aerospace Center (DLR), Institute of Communications and Navigation
- Hitachi Ltd./Digital Media Group
- In Fusio
- J-PhoneEast Co. Ltd
- Logica Mobile Networks
- Mitsubishi Electric Corp
- Mobile Scope AG
- Mobilitec
- Motorola
- NEC Corporation
- Nextel Communications Inc
- Nokia
- NTT DoCoMo, Inc
- Omnitel Pronto Italia S.p.A
- One 2 One
- Openwave Systems, Inc
- Orange (UK)
- Palm
- Philips Consumer Communications
- Philips Semiconductors
- Research In Motion (RIM)
- Samsung Electronics Co., Ltd
- Sharp Labs
- Siemens AG
- Siemens ICM
- Smart Fusion
- Sony Ericsson Mobile Communications
- Sun Microsystems, Inc
- Symbian Ltd
- Telefónica Móviles España
- Vaultus, Inc
- Veloxsoft, Inc
- Vodafone Global Platform & Internet Services
- Vodafone Group Services Limited

- •Vodafone Multimedia
- •Zucotto Wireless

- Individuals:
  - •Fabio Ciucci
  - •Glen Cordrey
  - •Jon Eaves
  - •David Hook
  - •Myank Jain
  - •Neil Katin
  - •Steve Ma
  - •Ravi Reddy
  - •Wai Kit Tony Fung

Introduction

This document, produced as a result of Java Specification Request (JSR) 118, defines the Mobile Information Device Profile (MIDP) v2.0 for the Java 2 Platform, Micro Edition (J2ME™). The goal of this specification is to define an enhanced architecture and the associated APIs required to enable an open, third-party, application development environment for mobile information devices, or MIDs.

The MIDP 2.0 specification is based on the MIDP 1.0 specification and provides backward compatibility with MIDP 1.0 so that MIDlets written for MIDP 1.0 can execute in MIDP 2.0 environments.

The MIDP is designed to operate on top of the Connected, Limited Device Configuration (CLDC) which is described in *Connected, Limited Device Configuration (JSR-30)* (http://jcp.org/jsr/detail/30.jsp), Sun Microsystems, Inc. While the MIDP 2.0 specification was designed assuming only CLDC 1.0 features, it will also work on top of *CLDC 1.1 (JSR-139)* (http://jcp.org/jsr/detail/139.jsp), and presumably any newer versions. It is anticipated that most MIDP 2.0 implementations will be based on CLDC 1.1.

Scope

Mobile Information Devices (MIDs) span a potentially wide set of capabilities. Rather than try to address all such capabilities, the MIDP 1.0 (JSR-037) and MIDP 2.0 (JSR-118) expert groups agreed to limit the set of APIs specified, addressing only those functional areas that were considered absolute requirements to achieve broad portability and successful deployments. These include:

- Application delivery and billing
- Application lifecycle (i.e., defining the semantics of a MIDP application and how it is controlled)
- Application signing model and privileged domains security model
- End-to-end transactional security (https)
- MIDlet push registration (server push model)
- Networking
- Persistent storage
- Sound
- Timers
- User interface (UI) (including display and input, as well as the unique requirements for games).

The above features are discussed in more depth in the associated Javadoc.

By the same reasoning, some areas of functionality were considered to be outside the scope of the MIDP. These areas include:

- System-level APIs: The emphasis on the MIDP APIs is, again, on enabling application programmers, rather than enabling system programming. Thus, low-level APIs that specify a system interface to, for example, a MID's power management or voice CODECs are beyond the scope of this specification.
- Low-level security: The MIDP specifies no additional low-level security features other than those provided by the CLDC.

Architecture

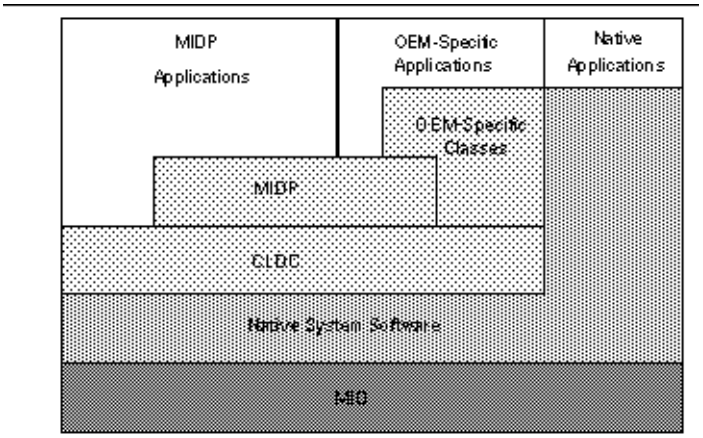
This section addresses issues that both implementers and developers will encounter when implementing and developing MIDP. While not comprehensive, this chapter does reflect the most important issues raised during deliberations of the MIDP Expert Group (MIDPEG).

As stated before, the goal of the MIDP is to create an open, third-party application development environment for MIDs. In a perfect world, this specification would only have to address functionality defined by the MIDP specification. In reality, most devices that implement the MIDP specification will be, at least initially, devices that exist on the market today. The High-Level Architecture shows a high-level view of how the MIDP fits into a device. Note that not all devices that implement the MIDP specification will have all the elements shown in this figure, nor will every device necessarily layer its software as depicted in this figure.

In the High-Level Architecture, the lowest-level block (MID) represents the Mobile Information Device hardware. On top of this hardware is the native system software. This layer includes the operating system and libraries used by the device.

Starting at the next level, from left to right, is the next layer of software, the CLDC. This block represents the Virtual Machine and associated libraries defined by the CLDC specification. This block provides the underlying Java functionality upon which higher-level Java APIs may be built.

High-Level Architecture View



Two categories of APIs are shown on top of the CLDC:

- MIDP APIs: The set of APIs defined in this specification.
- OEM-specific APIs: Given the broad diversity of devices in the MIDP space, it is not possible to fully address all device requirements. These classes may be provided by an OEM to access certain functionality specific to a given device. These applications may not be portable to other MIDs.

Note that in the figure, the CLDC is shown as the basis of the MIDP and device-specific APIs. This does not imply that these APIs cannot have native functionality (i.e., methods declared as native). Rather, the intent of the figure is to show that any native methods on a MID are actually part of the virtual machine, which maps the Java-level APIs to the underlying native implementation.

The top-most blocks in the figure above represent the application types possible on a MID. A short description of each application type is shown in the table below.

MID Application Types

Application Type	Description
MIDP	A MIDP application, or MIDlet, is one that uses only the APIs defined by the MIDP and CLDC specifications. This type of application is the focus of the MIDP specification and is expected to be the most common type of application on a MID.
OEM-Specific	An OEM-specific application depends on classes that are not part of the MIDP specification (i.e., the OEM-specific classes). These applications are not portable across MIDs.
Native	A native application is one that is not written in Java and is built on top of the MID's existing, native system software.

It is beyond the scope of this specification to address OEM-specific or native applications.

Device Requirements

The requirements listed in this chapter are additional requirements above those found in *Connected, Limited Device Configuration (JSR-30 and JSR-139)*, Sun Microsystems, Inc.

At a high level, the MIDP specification assumes that the MID is limited in its processing power, memory, connectivity, and display size.

Hardware

As mentioned before, the main goal of the MIDP is to establish an open, third-party application development environment for MIDs. To achieve this goal, the MIDPEG has defined a MID to be a device that SHOULD have the following minimum characteristics:

- Display:
- Screen-size: 96x54
- Display depth: 1-bit
- Pixel shape (aspect ratio): approximately 1:1
- Input:
- One or more of the following user-input mechanisms: one-handed keyboard, two-handed keyboard, or touch screen
- Memory:
- 256 kilobytes of non-volatile memory for the MIDP implementation, beyond what's required for CLDC.
- 8 kilobytes of non-volatile memory for application-created persistent data

- 128 kilobytes of volatile memory for the Java runtime (e.g., the Java heap)
- Networking:
- Two-way, wireless, possibly intermittent, with limited bandwidth
- Sound:
- The ability to play tones, either via dedicated hardware, or via software algorithm.

Examples of MIDs include, but are not restricted to, cellular phones, two-way pagers, and wireless-enabled personal digital assistants (PDAs).

Software

For devices with the aforementioned hardware characteristics, there is still a broad range of possible system software capabilities. Unlike the consumer desktop computer model where there are large, dominant system software architectures, the MID space is characterized by a wide variety of system software. For example, some MIDs may have a full-featured operating system that supports multi-processing and hierarchical filesystems, while other MIDs may have small, thread-based operating systems with no notion of a filesystem. Faced with such variety, the MIDP makes minimal assumptions about the MID's system software. These requirements are as follows:

- A minimal kernel to manage the underlying hardware (i.e., handling of interrupts, exceptions, and minimal scheduling). This kernel must provide at least one schedulable entity to run the Java Virtual Machine (JVM). The kernel does not need to support separate address spaces (or processes) or make any guarantees about either real-time scheduling or latency behavior.
- A mechanism to read and write from non-volatile memory to support the requirements of the Record Management System (RMS) APIs for persistent storage.
- Read and write access to the device's wireless networking to support the Networking APIs.
- A mechanism to provide a time base for use in time-stamping the records written to Persistent Storage and to provide the basis for the Timer APIs.
- A minimal capability to write to a bit-mapped graphics display.
- A mechanism to capture user input from one (or more) of the three input mechanisms previously discussed.
- A mechanism for managing the application life-cycle of the device.

Specification Requirements

This section lists some explicit requirements of this specification. Other requirements can be found in the associated Javadoc. If any requirements listed here differ from requirements listed elsewhere in the specification, the requirements here take precedence and replace the conflicting requirements.

Compliant MIDP 2.0 implementations:

- MUST support MIDP 1.0 and MIDP 2.0 MIDlets and MIDlet Suites.
- MUST include all packages, classes, and interfaces described in this specification.
- MUST implement the OTA User Initiated Provisioning specification.
- MAY incorporate zero or more supported protocols for push.
- MUST give the user a visual indication of network usage generated when using the mechanisms indicated in this specification.
- MAY provide support for accessing any available serial ports on their devices through the CommConnection interface.

- MUST provide support for accessing HTTP 1.1 servers and services either directly, or by using gateway services such as provided by WAP or i-mode.
- MUST provide support for secure HTTP connections either directly, or by using gateway services such as provided by WAP or i-mode.
- SHOULD provide support for datagram connections.
- SHOULD provide support for server socket stream connections.
- SHOULD provide support for socket stream connections.
- SHOULD provide support for secure socket stream connections.
- MUST support PNG image transparency.
- MAY include support for additional image formats.
- MUST support Tone Generation in the media package.
- MUST support 8-bit, 8 KHz, mono linear PCM wav format IF any sampled sound support is provided.
- MAY include support for additional sampled sound formats.
- MUST support Scalable Polyphony MIDI (SP-MIDI) and SP-MIDI Device 5-to-24 Note Profile IF any synthetic sound support is provided.
- MAY include support for additional MIDI formats.
- MUST implement the mechanisms needed to support “Untrusted MIDlet Suites”.
- MUST implement “Trusted MIDlet Suite Security” unless the device security policy does not permit or support trusted applications.
- MUST implement “Trusted MIDlet Suites Using X.509 PKI” to recognize signed MIDlet suites as trusted unless PKI is not used by the device for signing applications.
- MUST implement “MIDP x.509 Certificate Profile” for certificate handling of HTTPS and SecureConnections.
- MUST enforce the same security requirements for I/O access from the Media API as from the Generic Connection framework, as specified in the package documentation for javax.microedition.io.
- MUST support at least the UTF-8 (http://ietf.org/rfc/rfc2279.txt) character encoding for APIs that allow the application to define character encodings.
- MAY support other character encodings.
- SHOULD NOT allow copies to be made of any MIDlet suite unless the device implements a copy protection mechanism.

References

1. Scalable Polyphony MIDI Specification Version 1.0, MIDI Manufacturers Association, Los Angeles, CA, USA, February 2002.
2. Scalable Polyphony MIDI Device 5-to-24 Note Profile for 3GPP Version 1.0, RP-35, MIDI Manufacturers Association, Los Angeles, CA, USA, February 2002.

Package Summary

User Interface Package

javax.microedition.lcdui 115	The UI API provides a set of features for implementation of user interfaces for MIDP applications.
javax.microedition.lcdui.game 347	The Game API package provides a series of classes that enable the development of rich gaming content for wireless devices.

Application Lifecycle Package

javax.microedition.midlet 431	The MIDlet package defines Mobile Information Device Profile applications and the interactions between the application and the environment in which the application runs.

Persistence Package

javax.microedition.rms 463	The Mobile Information Device Profile provides a mechanism for MIDlets to persistently store data and later retrieve it.

Networking Package

javax.microedition.io 49	MID Profile includes networking support based on the `Generic Connection` framework from the *Connected, Limited Device Configuration*.

Public Key Package

javax.microedition.pki 453	Certificates are used to authenticate information for secure Connections.

Sound and Tone Media

javax.microedition.media 391	The MIDP 2.0 Media API is a directly compatible building block of the Mobile Media API (JSR-135) specification.
javax.microedition.media.control 421	This package defines the specific `Control` types that can be used with a `Player`.

Core Packages

java.lang 35	MID Profile Language Classes included from Java 2 Standard Edition.
java.util 39	MID Profile Utility Classes included from Java 2 Standard Edition.

CHAPTER 2

Over The Air User Initiated Provisioning Specification

for the Mobile Information Device Profile

Preface

This document, *Over The Air User Initiated Provisioning*, is for the Mobile Information Device Profile (MIDP) specification version 2.0. The original JSR and expert group details can be found at http://jcp.org/jsr/detail/118.jsp (http://jcp.org/jsr/detail/118.jsp). The terminology used herein is defined in the Definitions section of the MIDP 2.0 specification except where noted.

How This Specification Is Organized

The topics in this specification are organized in the following sections:

- *Section 1, "Over The Air User Initiated Provisioning"*, defines how MIDP applications should be distributed to wireless devices.
- *Section 2, "MIDP Provisioning and Networking in the WAP June2000 Environment"*, describes the specific requirements for deploying MIDP applications via a proxied WAP Gateway.

References

1. Connected, Limited Device Configuration (CLDC)
   http://jcp.org/jsr/detail/30.jsp (http://jcp.org/jsr/detail/30.jsp)
2. Mobile Information Device Profile (MIDP 1.0)
   http://jcp.org/jsr/detail/37.jsp (http://jcp.org/jsr/detail/37.jsp)
3. Mobile Information Device Profile 2.0 (MIDP 2.0)
   http://jcp.org/jsr/detail/118.jsp (http://jcp.org/jsr/detail/118.jsp)
4. HTTP 1.1 Specification
   http://www.ietf.org/rfc/rfc2616.txt (http://www.ietf.org/rfc/rfc2616.txt)
5. HTTP Authentication: Basic and Digest Access Authentication
   http://www.ietf.org/rfc/rfc2617.txt (http://www.ietf.org/rfc/rfc2617.txt)
6. Java(tm) Servlet 2.3 Specification
   http://jcp.org/jsr/detail/53.jsp (http://jcp.org/jsr/detail/53.jsp)

Changes since the OTA Recommended Practice

After the MIDP 1.0 specification was published, a document entitled, *Over The Air User Initiated Provisioning Recommended Practice for the Mobile Information Device Profile, was published*. This specification replaces that document, and the following changes were made for MIDP 2.0:

- Removed the Cookie support requirement. This was necessary because in some network architectures the cookie information may not be transmitted to the client. The cookies were used to maintain state information between the Application Descriptor, JAR downloads, and Install-Notify reports. An alternative approach of URL rewriting is possible, and can serve the same purpose. For example, when sending the Application Descriptor, the server can insert unique JAR, MIDlet-Install-Notify, and MIDlet-Delete-Notify URLs that associate these with this a particular download session. Other options may also be possible.

Section 1, Over The Air User Initiated Provisioning

Overview and Goals

The purpose of this document is to describe how MIDlet suites can be deployed Over-The-Air (OTA), and the requirements imposed upon the client device to support these deployments. Following these recommendations will help ensure interoperability between clients and servers from all manufacturers and provide guidance to mobile network operators deploying MIDP devices.

Devices MUST provide mechanisms that allow users to discover MIDlet suites that can be loaded into the device. In some cases, discovery will be via the device's resident browser (e.g., i-mode or WAP). In other cases, it may be a resident application written specifically to identify MIDlet suites for the user to download. Throughout this document, an application with this functionality will be referred to as the discovery application, or *DA*.

Other installation mechanisms (e.g. Bluetooth™ wireless technology, serial cable, IrDA™, etc.) MAY be supported by devices, but are outside the scope of this version of the specification.

The term Application Management Software (AMS) is a generic term used to describe the software on the device that manages the downloading and lifecycle of MIDlets. This term does not refer to any specific implementation and is used for convenience only. In some implementations, the term Java Application Manager (JAM) is used interchangeably.

This document describes the general functional requirements on the device and the functions supporting the MIDlet suite lifecycle. The lifecycle of a MIDlet suite consists of discovery, installation, update, invocation and removal. Descriptions are included for additional Application Descriptor attributes and mechanisms that identify the device type and characteristics to servers providing MIDlet suites.

Functional Requirements

A MIDP-compliant device MUST be capable of:

- Browsing, or otherwise locating MIDlet suite Application Descriptors in the network.
- Transferring a MIDlet suite and its associated Application Descriptor to the device from a server using HTTP 1.1 or a session protocol that implements the HTTP 1.1 functionality (including the header and entity fields) as required in this document.
- Responding to a 401 (*Unauthorized*) or 407 (*Proxy Authentication Required*) response to an HTTP request by asking the user for a user name and password and re-sending the HTTP request with the credentials supplied. The device MUST be able to support at least the RFC2617 Basic Authentication Scheme.
- Installing the MIDlet suite on the device
- Invoking MIDlets
- Allowing the user to delete MIDlet suites stored on the device. Single MIDlets cannot be deleted since the

MIDlet suite is the unit of transfer and installation.

MIDlet Suite Discovery
Application discovery is the process by which a user locates a MIDlet suite using the device. User-initiated discovery and installation of MIDlet suites MUST be supported in the following high-level manner:

- While using the DA, the user is presented with a link to a MIDlet suite or Application Descriptor.
- The user selects the link to begin the installation process
- If available, the Application Descriptor is transferred to the device first. This descriptor contains information about the MIDlet suite and can be used by the device's AMS to start installation
- If the Application Descriptor is not available, or after the AMS has downloaded the Application Descriptor and determined that installation should continue, the MIDlet suite JAR file download begins.

Using the DA, the user SHOULD be able to access a network location and see a description of the MIDlet suite along with a link that, when selected, initiates the installation of the MIDlet suite. If the link refers to a JAR file as described in the MIDP specification, the JAR file and its URL are passed to the AMS on the device to start the installation process. If the link refers to an Application Descriptor, as described in the MIDP specification:

1. Once the link has been selected, the server MUST indicate in the response that the data being transferred (i.e., the Application Descriptor) has a MIME type of "*text/vnd.sun.j2me.app-descriptor*".
2. After completing this transfer, the application descriptor and its URL are passed to the AMS on the device to start the installation process. The Application Descriptor is used by the AMS to determine if the associated MIDlet suite can be successfully installed and executed on the device. If not, the user MUST be notified of the conditions that prevent its installation. The user SHOULD be informed of unusual conditions as early as possible to minimize wasted time and network bandwidth. The request-header attributes described in Device Identification and Request Headers SHOULD be used when retrieving the Application Descriptor.
3. The Application Descriptor MUST be converted from its transport format to the Unicode-encoding that is specified by the MIDP specification before it can be used. The default character set specified for the MIME type "*text/vnd.sun.j2me.app-descriptor*" is "*UTF-8*". If the device supports other character sets, the appropriate *Accept-Charset* header SHOULD be included in the request, and the content SHOULD be converted based on the *charset* attribute returned on the *Content-Type* header. If *charset* is undefined, the encoding defaults to "*UTF-8*", and it SHOULD be converted accordingly. The attributes in the descriptor MUST be formatted according to the syntax in the MIDP specification and all of the attributes required by the MIDP specification MUST be present in the descriptor. If this is not the case, then the client MUST return *Status Code 906* in the status report.
4. Using the information in the Application Descriptor including the vendor, name, version, and size attributes, the user SHOULD be given a chance to confirm that they want to install the MIDlet suite. Situations such as trying to install an older version, or installing the same version, SHOULD be brought to the user's attention. Conditions that can prevent the successful installation and execution of the MIDlet suite SHOULD be identified, and the user notified. For example, if it is known that insufficient memory is available, the software SHOULD aid the user in reviewing memory usage and freeing sufficient memory for installation of the new MIDlet suite.

MIDlet Suite Installation
Application installation is the process by which a MIDlet suite is downloaded onto the device and made available to the user. Application installation MUST be supported. The network supporting the devices, as well any proxies and origin servers that are used during provisioning, MUST be able to support this requirement. The

user retains control of the resources used by MIDlet suites on the device and MUST be allowed to delete or install MIDlet suites.

The device MUST make the MIDlet(s) in the MIDlet suite available for execution by the user. When multiple MIDlets are contained in a MIDlet suite, the user MAY need to be aware that there is more than one. The device MAY run a MIDlet from the MIDlet suite immediately at the user's option.

During installation, the user SHOULD be informed of progress and MUST be given an opportunity to cancel the process. Interrupting installation MUST leave the device in the state it was in before installation began.

If the MIDlet suite is already installed on the device, it SHOULD be treated as an update. See MIDlet Suite Update for additional information on how to handle an update.

To install a MIDlet suite, the AMS performs the following series of steps and checks and provides the user with feedback about the progress:

1. The device initiates the download of the MIDlet suite via HTTP. If an Application Descriptor was first downloaded as described in the MIDlet suite Discovery section, the request for the MIDlet suite MUST be for exactly the URL specified in the descriptor; additional headers are unnecessary.
2. If the server or proxy responds to the request for the MIDlet suite with a 401 (*Unauthorized*) or 407 (*Proxy Authentication Required*), the device SHOULD re-send the request with the user-supplied credentials in an *Authorization* or *Proxy-Authorization* header field as specified in RFC2617. The credentials SHOULD be provided by the user—-for example, a common mechanism would be to present a dialog to the user to enter a user name and password. The device MUST be able to support at least the Basic Authentication Scheme as described in RFC2617.
3. The MIDlet suite and the headers that are received MUST be checked to verify that the retrieved MIDlet suite is valid and can be installed on the device. The user MUST be alerted to at least the following problems that prevent installation:

- If there is insufficient memory to store the MIDlet suite on the device, the device MUST return *Status Code 901* in the Status Report.
- If the JAR is not available at the *MIDlet-Jar-URL* attribute in the descriptor, the device MUST return *Status Code 907* in the Status Report.
- If the received JAR file size does not match the size specified in the Application Descriptor, the device MUST return *Status Code 904* in the Status Report.
- If the manifest or any other file can not be extracted from the JAR, the device MUST return *Status Code 907* in the Status Report.
- If the JAR manifest is not in the correct syntax, or if any of the required attributes are missing in the JAR manifest, the device MUST return *Status Code 907* in the Status Report.
- If the mandatory attributes in the descriptor "*MIDlet-Name*", "*MIDlet-Version*", and "*MIDlet-Vendor*" do not match those in the JAR manifest, the device MUST return *Status Code 905* in the Status Report.
- If the MIDlet suite is trusted, then the values in the application descriptor for MIDlet-* attributes MUST be identical to the corresponding attribute values in the Manifest. If not, the device MUST return *Status Code 905* in the Status Report.
- If the application failed to be authenticated, the device MUST return *Status Code 909* in the Status Report.
- If the application is an unsigned version of an installed signed version of the same application, the device MUST return *Status Code 910* in the Status Report.
- If the application is not authorized for a permission listed in the *MIDlet-Permissions* attribute, the device MUST return *Status Code 910* in the Status Report.

- If a static push registration fails for a reason other than not being authorized, the device MUST return *Status Code 911* in the Status Report.
- If the network service is lost during installation, *Status Code 903* SHOULD be used in a Status Report if possible (it may be impossible to deliver the status report due to the network-service outage).

1. Provided there are no problems that prevent installation, the MIDlets contained in the MIDlet suite MUST be installed and made available for execution by the user via the device's MIDlet selection mechanism.
2. Installation is complete when the MIDlet suite has been made available on the device, or an unrecoverable failure has occurred. In either case, the status MUST be reported as described in Installation Status Reports..

MIDlet Suite Update

A MIDlet suite update is defined as the operation of installing a specific MIDlet suite when that same MIDlet suite (either the same version or a different version) is already installed on the device. Devices MUST support the updating of MIDlet suites. In order to be meaningful to the user, the device MUST allow the user to obtain information about the MIDlet suite(s) on the device and determine which versions of software are installed. See Device Identification and Request Headers. for the attributes that apply to updates.

When a MIDlet suite update is started, the device MUST notify the user if the MIDlet suite is a newer, older, or the same version of an existing MIDlet suite and MUST get confirmation from the user before proceeding.

The RMS record stores of a MIDlet suite being updated MUST be managed as follows:

- If the cryptographic signer of the new MIDlet suite and the original MIDlet suite are identical, then the RMS record stores MUST be retained and made available to the new MIDlet suite.
- If the scheme, host, and path of the URL that the new Application Descriptor is downloaded from is identical to the scheme, host, and path of the URL the original Application Descriptor was downloaded from, then the RMS MUST be retained and made available to the new MIDlet suite.
- If the scheme, host, and path of the URL that the new MIDlet suite is downloaded from is identical to the scheme, host, and path of the URL the original MIDlet suite was downloaded from, then the RMS MUST be retained and made available to the new MIDlet suite.
- If the above statements are false, then the device MUST ask the user whether the data from the original MIDlet suite should be retained and made available to the new MIDlet suite.

In all cases, an unsigned MIDlet MUST NOT be allowed to update a signed MIDlet suite. The format, contents and versioning of the record stores is the responsibility of the MIDlet suite. The user-granted permissions given to the original MIDlet suite SHOULD also be given to the new MIDlet suite, if they are in the security domain of the new MIDlet suite.

MIDlet Suite Execution

When the user selects a MIDlet to be run, the device MUST invoke the MIDlet with the CLDC and MIDP classes required by the MIDP specification. If multiple MIDlets are present, the user interface MUST allow the user to select each one for execution.

MIDlet Suite Removal

Devices MUST allow users to remove MIDlet suites. When a MIDlet suite is to be removed from the device, the user SHOULD be prompted to confirm that the MIDlet suite may be removed. The device SHOULD warn the user of any special circumstances that arise during the deletion of the MIDlet suite. For example, the MIDlet suite MAY contain multiple MIDlets, and the user SHOULD be made aware that all of the MIDlets and associated RMS record stores are being removed.

If the Application Descriptor includes the attribute MIDlet-Delete-Confirm, its value SHOULD be included in the prompt. This will allow the MIDlet suite provider to highlight any specific conditions that might arise if the MIDlet suite were to be removed.

Installation/Deletion Status Reports

The success or failure of the installation, upgrade, or deletion of a MIDlet suite is of interest to the service providing the MIDlet suite. The service MAY specify URLs in the Application Descriptor that MUST be used to report installation and deletion status. See Additional Descriptor Attributes for more information. If the device cannot send the installation status report, the requested action MUST still be completed. For example, if the device cannot send the installation status report to the MIDlet-Install-Notify URL, the MIDlet suite MUST still be enabled, and the user MUST be allowed to use it. Likewise if the device cannot send the deletion status report to the MIDlet-Delete-Notify URL, the MIDlet suite MUST still be deleted.

The operation status is reported by means of an HTTP POST to the URL specified in the *MIDlet-Install-Notify* attribute for installations, or the *MIDlet-Delete-Notify* attribute for deletions. The only protocol that MUST be supported is "*http://*". Other protocols MAY be ignored by the device.

The content of the body of the POST request MUST include a status code and status message on the first line. See Status Codes and Message for list of valid codes and status messages.

In the case of a deletion status report, the notification is sent only when the MIDlet is deleted; *Status Code 912* MUST be sent, notifying that the deletion occurred.

In response to a status report, the server MUST reply with a "*200 OK*" response. No content SHOULD be returned to the device and, if any is sent, it MUST be ignored. If a response is received the request SHOULD NOT be retried. Contrary to the MIDP 1.0 OTA Recommended Practice, the server MUST NOT include a *Set-Cookie* header with the attribute *Max-Age=0* to request that the *cookie* be discarded. If such an attribute is received, the device MUST ignore it. As an example, please see Example: Install Status via HTTP Post Request.

For installations, if the status report cannot be sent, or if the server reply is not received, the installation status report MAY be sent again (as described above) each time a MIDlet in this suite is executed and the device has data network connectivity. This will improve the likelihood of the status report being successfully sent. The number of retries attempted SHOULD be kept small since each one may result in a charge to the user's bill. The MIDlet suite MUST be made available for use, whether or not the installation status report has been successfully sent and the acknowledgement have been received.

For deletions, an attempt to send the status report MUST be made the next time either an OTA installation is performed or an installation status report is being sent. This will improve the likelihood of the status report being successfully sent and will minimize confusion by the user when they see network activity. If the status report cannot be sent, or if the server reply is not received, the deletion status report MAY be sent again (as described above) each time an OTA installation installation is performed or an installation status report is being sent. The number of retries attempted SHOULD be kept small since each one may result in a charge to the user's bill. The MIDlet suite MUST be removed from memory, whether or not the installation status report has been successfully sent and the acknowledgement have been received.

Install Status Codes and Message

Status Code	Status Message
900	Success
901	Insufficient Memory

902	User Cancelled
903	Loss of Service
904	JAR size mismatch
905	Attribute Mismatch
906	Invalid Descriptor
907	Invalid JAR
908	Incompatible Configuration or Profile
909	Application authentication failure
910	Application authorization failure
911	Push registration failure
912	Deletion Notification

Additional Descriptor Attributes

The following additional attributes are defined in the Application Descriptor. Each may appear only once in the descriptor.

MIDlet Attributes

Attribute Name	Attribute Description
MIDlet-Install-Notify	The URL to which a POST request is sent to report the installation status (whether a new installation or MIDlet suite update) of this MIDlet suite. The device MUST use this URL unmodified. The URL MUST be no longer than 256 UTF-8 encoded characters. If the device receives a URL longer than 256 UTF-8 encoded characters it MUST reject the installation and return *Status Code 906* in the status report.
MIDlet-Delete-Notify	The URL to which a POST request is sent to report the deletion of this MIDlet suite. The device MUST use this URL unmodified. The URL MUST be no longer than 256 UTF-8 encoded characters. If the device receives a URL longer than 256 UTF-8 encoded characters it MUST reject the installation and return *Status Code 906* in the status report.
MIDlet-Delete-Confirm	A text message to be provided to the user when prompted to confirm deletion of this MIDlet suite

Device Identification and Request Headers

The process of discovering a MIDlet suite via the DA can be customized by the device sending information about itself to the server. The DA MUST provide the network server with information (e.g. the manufacturer and device model number) so that the server can determine the device's capabilities. In many cases, a DA will already have identified the device type to the server by means consistent with its network connection and markup language.

During the download of a MIDlet suite, a device SHOULD identify its characteristics and the type of the content being requested as completely as possible to the server. The HTTP request-headers used to fetch the

content MUST include, *User-Agent, Accept-Language,* and *Accept*. Servers SHOULD use this additional information to select the appropriate Application Descriptor for the device.

User-Agent Product Tokens
The MIDP specification identifies HTTP *User-Agent* request headers to identify the client to the server. RFC2616 specifies a format for product tokens such as:

"User-Agent" ":" 1(product | comment)*

The product tokens used to identify the device as supporting CLDC and MIDP are specified the Networking portion of the MIDP specification. As in RFC2616, the comment field is optional.

In addition, the device SHOULD further identify itself by adding a device-specific product token to the *User-Agent* header as defined by RFC2616. The device-identifying token SHOULD be the first token. The product-token and product-version values are specific to each device and are outside of the scope of this specification.

Accept-Language Header
The device MAY supply the *Accept-Language* request-header as specified in RFC2616 to indicate the language that is in use on the device.

Accept Header
The *Accept* HTTP header is used to indicate the type of content being requested. When requesting MIDlet suites, this header SHOULD include *application/java-archive*. For retrieving application descriptors, this header SHOULD include *text/vnd.sun.j2me.app-descriptor.*

Example: HTTP Request for Application Descriptor
When requesting the download of an Application Descriptor, the request headers might look as follows:

```
GET http://host.foo.bar/app-dir/game.jad HTTP/1.1
 Host: host.foo.bar
 Accept: text/vnd.sun.j2me.app-descriptor
 User-Agent: CoolPhone/1.4 Profile/MIDP-2.0 Configuration/CLDC-1.0
 Accept-Language: en-US, fi, fr
 Accept-Charset: utf-8
```

The response headers from the server might look as follows:

```
HTTP/1.1 200 OK
 Server: CoolServer/1.3.12
 Content-Length: 2345
 Content-Type: text/vnd.sun.j2me.app-descriptor; charset=utf-8
```

Example: HTTP Request to Install/Update a MIDlet suite
When requesting the download of a MIDlet suite JAR file, the request headers might look as follows:

```
GET http://host.foo.bar/app-dir/game.jar HTTP/1.1
 Host: host.foo.bar
 Accept: application/java, application/java-archive
```

The response headers from the server might look as follows:

```
HTTP/1.1 200 OK
 Server: CoolServer/1.3.12
 Content-Length: 25432
 Content-Type: application/java-archive
```

Example: Install Status via HTTP Post Request
For example, installing a MIDlet suite with an application descriptor given below:

```
...
 MIDlet-Install-Notify: http://foo.bar.com/status
 ...
```

After a successful install of the MIDlet suite, the following would be posted:

```
POST http://foo.bar.com/status HTTP/1.1
 Host: foo.bar.com
 Content-Length: 13
 900 Success
```

The response from the server might be:

```
HTTP/1.1 200 OK
 Server: CoolServer/1.3.12
```

Section 2, MIDP Provisioning and Networking in the WAP June2000 Environment

Purpose of This Section
The purpose of this section is to complement the OTA and MIDP specifications by providing requirements and recommendations specific to MIDP Over The Air Provisioning and MIDlet networking in the WAP June2000 environment. Future WAP developments will be addressed in future versions of the MIDP. Following these recommendations will help ensure interoperability between different WAP elements from all manufacturers. It will also provide guidance to network operators in deploying MIDP services when provisioning is performed via a browser using the WAP protocol stack, as well as to MIDlet developers in creating MIDlets that function optimally when the transport is WSP.

Overview
MIDlet suites are downloaded using HTTP from a provisioning server (possibly via a gateway in between). Also, the MIDP library MUST support network access in the form of the HTTP/1.1 protocol.

Depending on the end-user device and the wireless network, the communication MAY occur between the end-user device and provisioning server with the HTTP protocol end-to-end, or the end-user device MAY use another protocol, and have a gateway convert this protocol to HTTP. The provisioning server needs to only support HTTP in any case (unless there are other reasons for the same service provider to operate the protocol gateway as well). In WAP June2000 environments, there is always a WAP gateway between the terminal and the provisioning server to translate between the WSP protocol used to communicate with the device and TCP/IP used to communicate with the server.

There are essentially two basic interfaces that need to be considered:

- the interface from the end-user device to the network
- the interface from the provisioning server to the network

The latter of these interfaces will always be HTTP carried as usual over TCP/IP.

For the former interface, this document describes one of the two basic cases:

- the end-user device uses a browser using the WAP protocol stack and
- the WSP protocol is used for communication between the terminal and the WAP gateway.

When the end-user device uses a browser using the WAP protocol stack and has the WAP transport protocols, WSP MAY be used instead of HTTP in the end-user device. Only connection-oriented WSP and only the following WAP protocol stack configurations and bearers are supported:

- WAP/UDP/IPv4/PPP/CSD
- WAP/UDP/IPv4/GPRS

Where WAP can be either:

- WSP/WTP/WTLS, or
- WSP/WTP

(The other bearers in [WAP_WDPS.], such as SMS- or USSD -based bearers are not supported). These restrictions are made in order to achieve maximal interoperability in MIDP provisioning in WAP environments.

Depending on the wireless network and the capabilities of the end-user device, different mechanisms for obtaining the IP connection are used. These mechanisms and their required configurations are outside the scope of this document.

Terminal Requirements and Recommendations
This section lists the requirements and recommendations related to the WAP terminals. In the case of common requirements to both terminals and gateways, the requirements and recommendations are listed in both terminal and gateway sections.

WAP terminals used MUST be WAP June2000 conformant.

Specifically, the following issues are critical:

- JAD and JAR MIME-types, as described in previous sections, MUST be supported.
- HTTP authentication (server responses 401 and 407) MUST be supported.
- POST-messages from the terminal to provisioning server MUST be supported.

Requirements and recommendations in addition to those in the WAP Specifications
In the case where the HTTP connections are implemented over WSP, the system implementation of HTTP MUST add the request header "*Accept: */*" to GET and POST requests when a MIDlet creates an HTTP-request, but the MIDlet does not include a non-empty *Accept* header in the request. This ensures that the WAP Gateway will always have an explicit set of types and will pass the requested data. This is conceptually the same as leaving out the *Accept* header from an HTTP-request on other transports. If the MIDlet sets a non-empty *Accept* header for its HTTP-request, no change is made (the MIDlet's own *Accept* field is the only one sent).

Gateway Requirements and Recommendations
This section lists the requirements and recommendations related to the WAP gateways. The purpose of presenting these issues here is to make sure that they are taken into consideration when WAP-based MIDlet provisioning is considered.

WAP gateways used MUST be WAP June2000 conformant.

Specifically, the following issues are critical:

- JAD and JAR MIME-types MUST be supported. WAP Gateway must follow the rules for HTTP proxies (RFC2616) these MIME types.
- HTTP authentication (server responses 401 and 407) MUST be supported.
- Data of any kind MUST be passed to the terminal, if the terminal's request has included " *Accept: */*" header.
- POST-messages from the terminal to provisioning server MUST be supported.

MIDlet/MIDlet Suite Recommendations

MIDlets SHOULD function correctly even with long connection setup delays and long breaks in connection. Long connection setup delays affect circuit-switched data connections, and long breaks affect GPRS connections.

References

1. OTA
   Over The Air User Initiated Provisioning for Mobile Information Device Profile
2. MIDP
   Mobile Information Device Profile Specification 1.0, http://jcp.org/jsr/detail/37.jsp (http://jcp.org/jsr/detail/37.jsp)
3. MIDP 2.0
   Mobile Information Device Profile Specification 2.0, http://jcp.org/jsr/detail/118.jsp (http://jcp.org/jsr/detail/118.jsp)
4. WAP_JUNE2000
   WAP June2000 Conformance Release, http://www.wapforum.org/what/technical.htm (http://www.wapforum.org/what/technical.htm)
5. WAP_WDPS
   WAP Wireless Datagram Protocol Specification, http://www.wapforum.org/what/technical.htm (http://www.wapforum.org/what/technical.htm)

Terms

- CSD = Circuit Switched Data
- GPRS = General Packet Radio Service
- PPP = Point-to-Point Protocol
- SMS = Short Message Service
- USSD = Unstructured Supplementary Services Data
- WAP = Wireless Application Protocol
- WSP = Wireless Session Protocol

CHAPTER 3

Security for MIDP Applications

The MIDP 1.0 specification constrained each MIDlet suite to operate in a sandbox wherein all of the APIs available to the MIDlets would prevent access to sensitive APIs or functions of the device. That sandbox concept is used in this specification and all untrusted MIDlet suites are subject to its limitations. Every implementation of this specification MUST support running untrusted MIDlet suites.

MIDP 2.0 introduces the concept of trusted applications that may be permitted to use APIs that are considered sensitive and are restricted. If and when a device determines that a MIDlet suite can be trusted then access is allowed as indicated by the domain policy. The Trusted MIDlet Suite Security section below describes the concepts. Any MIDlet suite that is not trusted by the device MUST be run as untrusted. If errors occur in the process of verifying that a MIDlet suite is trusted then the MIDlet suite MUST be rejected.

Untrusted MIDlet Suites

An untrusted MIDlet suite is a MIDlet suite for which the origin and the integrity of the JAR file can NOT be trusted by the device. Untrusted MIDlet suites MUST execute in the untrusted domain using a restricted environment where access to protected APIs or functions either is not allowed or is allowed with explicit user permission. Any MIDP 1.0 compliant MIDlet suite MUST be able to run in an implementation of this specification as untrusted. Any APIs or functions of this specification which are not security sensitive, having no permissions defined for them, are implicitly accessible by both trusted and untrusted MIDlet suites. Untrusted MIDlet suites do not request permissions explicitly in the JAR manifest or application descriptor.

The untrusted domain for untrusted MIDlet suites MUST allow, without explicit confirmation by the user, access to:

API	Description
`javax.microedition.rms`	RMS APIs
`javax.microedition.midlet`	MIDlet Lifecycle APIs
`javax.microedition.lcdui`	User Interface APIs
`javax.microedition.lcdui.game`	The Game APIs
`javax.microedition.media` `javax.microedition.media.control`	The multi-media APIs for playback of sound

The untrusted domain for untrusted MIDlet suites MUST allow, with explicit confirmation by the user, access to protected APIs or functions:

API	Protocol
`javax.microedition.io.HttpConnection`	`http`
`javax.microedition.io.HttpsConnection`	`https`

Trusted MIDlet Suite Security

Security for Trusted MIDlet suites is based on protection domains. Each protection domain defines the permissions that may be granted to a MIDlet suite in that domain. The protection domain owner specifies how the device identifies and verifies that it can trust a MIDlet suite and bind it to a protection domain with the permissions that authorize access to protected APIs or functions. The mechanisms the device uses to identify and trust MIDlet suites are defined separately to allow them to be selected appropriately to the device, network, and business case.

The Trusted MIDlet Suites Using X.509 PKI describes a mechanism for identifying trusted MIDlet suites though signing and verification. If an implementation of this specification will recognize MIDlet suites signed using PKI as trusted MIDlet suites they must be signed and verified according to the formats and processes specified in Trusted MIDlet Using X.509 PKI.

Definition of Terms

Term	Definition
Protection Domain	A set of *Allowed* and *User* permissions that may be granted to a MIDlet suite
Permission	A named permission defined by an API or function to prevent it from being used without authorization
Trusted MIDlet Suite	A MIDlet suite for which the authentication and the integrity of JAR file can be trusted by the device and bound to a protection domain

Authorization Model

The basic authorization of a MIDlet suite is established by the relationships between the following elements:

- A protection domain consisting of a set of *Allowed* and *User* permissions
- A set of permissions requested by the MIDlet suite in `MIDlet-Permissions` and `MIDlet-Permissions-Opt` attributes
- A set of permissions for each protected API or function on the device which is a union of all permissions defined by every API on the device for protected functions
- The user who may be asked to grant permissions

Assumptions

- MIDlets do not need to be aware of the security policy except for security exceptions that may occur when using APIs.
- A MIDlet suite is subject to a single protection domain and its permissible actions.
- The internal representation of protection domains and permissions is implementation specific.

- The details of how authentication results and configuration settings are presented to the user in the user interface are implementation dependent and are outside the scope of this specification.
- The device must protect the security policy and protection domain information stored in the device from viewing or modification except by authorized parties.
- If the security policy for a device is static and disallows use of some functions of the security framework then the implementation of unused and inaccessible security functions may be removed.
- Security policy allows an implementation to restrict access but MUST NOT be used to avoid implementing functionality. For example, unimplemented protocols under the Generic Connection framework MUST throw `ConnectionNotFoundException`.

Permissions

Permissions are the means to protect access to APIs or functions which require explicit authorization before being invoked. Permissions described in this section only refer to those APIs and functions which need security protection and do not refer to other APIs which can be accessed by both trusted and untrusted MIDlet suites and do not need explicit permission. Permissions are checked by the implementation prior to the invocation of the protected function.

The names of permissions have a hierarchical organization similar to Java package names. The names of permissions are case sensitive. All of the permissions for an API MUST use the prefix that is the same as the package name of the API. If the permission is for a function of a specific class in the package then the permission MUST include the package and classname. The set of valid characters for permissions is the same as that for package and class names. The conventions for use of capitalization in package names SHOULD be used for permission names. For example, `javax.microedition.io`. Following the permission name, whether by package or class, additional modifiers may be appended with a separator of "." (Unicode U+002E).

Each API in this specification that provides access to a protected function will define the permissions. For APIs defined outside of MIDP 2.0 there must be a single document that specifies any necessary permissions and the behavior of the API when it is implemented on MIDP 2.0.

Permissions for Protected Functions

Each function (or entire API) which was identified as protected must have its permission name defined in the class or package documentation for the API.

Refer to the documentation of the `javax.microedition.io` package for permissions on all Generic Connection schemes defined in this specification. All APIs and functions within this specification that do not explicitly define permissions MUST be made available to all trusted and untrusted MIDlet suites.

Requesting Permissions for a MIDlet Suite

A MIDlet suite that requires access to protected APIs or functions must request the corresponding permissions. Permissions requested can be required by listing the permissions in the attribute `MIDlet-Permissions`. These permissions are critical to the function of the MIDlet suite and it will not operate correctly without them.

If the MIDlet suite can function correctly with or without particular permission(s) it should request them using the `MIDlet-Permissions-Opt` attribute. The MIDlet suite is able to run with reduced functionality (for example, as a single player game instead of a net game) without these non-critical permissions and MUST be installed and run.

The `MIDlet-Permissions` and `MIDlet-Permissions-Opt` attributes contain a list of one or more permissions. Multiple permissions are separated by a comma (Unicode U+002C). Leading and trailing whitespace (Unicode U+0020) and tabs (Unicode U+0009) are ignored.

Permissions on the Device

Each device that implements this specification and any other Java APIs will have a total set of permissions referring to protected APIs and functions. It is the union of all permissions defined by every protected function or API on the device.

Protection Domain

A protection domain defines a set of permissions and related interaction modes. A protection domain consists of:

- a set of permissions that should be allowed (*Allowed*)
- a set of permissions that the user may authorize (*User*); each with its user interaction mode

Within a protection domain each permission may be either *allowed* or *user* but not both.

The *Allowed* permissions are any permissions which explicitly allow access to a given protected API or function on the basis of MIDlet suite being associated with the protection domain. *Allowed* permissions do not require any user interaction.

The *User* permissions are any permissions for a protected API or function on the basis of MIDlet suite being bound to the protection domain and will allow access to protected API or function after the prompt given to the user and explicit user permission being granted.

User Permission Interaction Modes

A User Permission is defined to allow the user to deny permission or to grant permission to a specific API with one of the following interaction modes:

- "`blanket`" is valid for every invocation of an API by a MIDlet suite until it is uninstalled or the permission is changed by the user.
- "`session`" is valid from the invocation of a MIDlet suite until it terminates. "`session`" mode MUST prompt the user on or before the first invocation of the API or function which is protected. When the user re-invokes the MIDlet suite the prompt MUST be repeated.
- "`oneshot`" MUST prompt the user on each invocation of the API or function which is protected.

The choice of user permission interaction modes is driven by the security policy and the device implementation. Each user permission has a default interaction mode and a set of other available interaction modes. The user SHOULD be presented with a choice of interaction modes. The default interaction mode may be offered if it is supplied. The user MUST always be able to deny permission.

If and when prompted, the user SHOULD be provided with a user friendly description of the requested permissions sufficient to make a well-informed choice.

The range of blanket to oneshot action permission modes represents a tradeoff between usability and user notification and should behave smoothly and consistently with the human interface of the device.

Granting Permissions to Trusted MIDlet Suites

Authorization of trusted MIDlet suites uses protection domain information, permissions on the device, and permissions requested in the MIDlet suite. Permissions in the domain are *Allowed* or *User.* Permissions requested by the application are either critical or non-critical.

To establish the permissions granted to a trusted MIDlet suite when it is to be invoked all of the following MUST be true:

- The MIDlet suite must have been bound to a protection domain.
- The requested critical permissions are retrieved from the attributes `MIDlet-Permissions` and non-critical permissions from `MIDlet-Permissions-Opt`. If these attributes appear in the application

descriptor they MUST be identical to corresponding attributes in the manifest. If they are not identical, the MIDlet suite MUST NOT be installed or invoked.

- If any of the requested permissions are unknown to the device and are not marked as critical then they are removed from the requested permissions.
- If any of the requested permissions are unknown to the device and marked as critical, the MIDlet suite MUST NOT be installed or invoked.
- If any of the requested permissions are not present in the protection domain (*Allowed* or *User*) permission sets and the requested permission was marked as critical then the MIDlet suite does not have sufficient authorization and MUST NOT be installed or invoked.
- If any of the requested permissions are not present in the protection domain (*Allowed* or *User*) permission sets, and the requested permissions are not marked as critical, the application MUST still be installed and MUST be able to be invoked by the user.
- If any of the requested permissions match the *User* permissions of the protection domain then the user MUST explicitly provide authorization to grant those permissions to the MIDlet suite. The implementation is responsible for making the request to the user and getting the response to allow or deny the request.
- The permissions granted to the MIDlet suite are the intersection of the requested permissions with the union of the allowed and user granted permissions.
- During execution, any protected APIs MUST check for the appropriate permissions and throw a `SecurityException` if the permission has not been granted.

The successful result of authorization is that the MIDlet suite is granted access to protected APIs or functions for which it requested permissions.

Example External Domain Policy Format

An external representation for protection domains allows clear communication between developers, operators and manufacturers. This format is provided only as an example. There is no requirement for an implementation of this specification to use this format. The policy file character set is UTF-8 encoding of Unicode to support any language. The policy file syntax is based on the JAR manifest format.

A policy consists of the definitions of domains and aliases. Each domain consists of the definition of granted permissions and user permissions. Aliases permit groups of named permissions to be reused in more than one domain and helps keep the policy compact. Aliases may only be defined and used within a single policy file. References to an alias MUST follow the definition of the alias in the policy file.

A domain is defined with a domain identifier and a sequence of permissions. The domain identifier is implementation specific. Each permission line begins with "`allow`" or user permissions "`blanket`", "`session`", or "`oneshot`" to indicate the interaction level for the list of permissions that follow. User permissions may also include a default mode. Multiple permission lines are allowed. The permissions are processed in order and if a permission occurs multiple times within a domain only the last definition of the permission is used. It is not recommended that permissions appear more than once.

BNF Syntax:

```
policy_file = 1*(directive)
directive = (domain_def | alias_def) [newlines]
domain_def = “domain:” *WS domain_id *WS newlines
             1*permission
domain_id = 1*<any Unicode char and continuation, but not newline>
permission = permision_level “:” api_names newlines
api_names: *WS alias_or_name *(*WS “,” *WS alias_or_name) *WS
alias_or_name = alias_ref | api_name
alias_ref = <alias_name from a previous alias_def in the same policy_file>
permission_level = allow | user_permission_levels
user_permision_levels = highest_level [“(” default_level “)”]
highest_level = user_permission_level
default_level = user_permision_level ; cannot be greater the highest_level
user_permission_level = blanket | session | oneshot
allow = “allow” ; allow access without asking the user.
blanket = “blanket” ; Allow access, do not ask again.
                    ; Include session and oneshot when asking.
session = “session” ; Allow access, ask again at next MIDlet suite startup.
                    ; Include oneshot when asking.
oneshot = “oneshot” ; Allow access, ask again at next use.
                      ; If no default provided, default is to deny access.
alias_def = “alias:” *WS alias_name 1*WS alias_api_names
alias_api_names =  api_name
                    *(*WS “,” *WS api_name) *WS newlines
alias_name = java_name
api_name = java_class_name
WS = continuation | SP | HT
continuation = newline SP
newlines = 1*newline  ; allow blank lines to be ignored
newline = CR LF | LF | CR <not followed by LF>
CR = <Unicode carriage return (U+000D)>
LF = <Unicode linefeed (U+000A)>
SP = <Unicode space (U+0020)>
HT = <Unicode horizontal-tab (U+0009)>
java_name = 1*<characters allowed in a java_class_name except for “.”>
java_class_name = 1*<characters allowed in a Java class name>
```

Example policy file:

```
domain: O=“MIDlet Underwriters, Inc.”, C=US
allow: javax.microedition.io.HttpConnection
oneshot(oneshot): javax.microedition.io.CommConnection
alias: client_connections javax.microedition.io.SocketConnection,
  javax.microedition.io.SecureConnection,
  javax.microedition.io.HttpConnection,
  javax.microedition.io.HttpsConnection
domain: O=Acme Wireless, OU=Software Assurance
allow: client_connections
allow: javax.microedition.io.ServerSocketConnection,
˝ javax.microedition.io.UDPDatagramConnection
oneshot(oneshot): javax.microedition.io.CommConnection
domain: allnet
blanket(session): client_connections
oneshot: javax.microedition.io.CommConnection
```

CHAPTER 4

Trusted MIDlet Suites using X.509 PKI

Signed MIDlet suites may become trusted by authenticating the signer of the MIDlet suite and binding it to a protection domain that will authorize the MIDlet suite to perform protected functions by granting permissions allowed in the protection domain. The mechanisms defined here allow signing and authentication of MIDlet suites based on X.509 Public Key Infrastructure so the device can verify the signer and trust the MIDlet suite.

If an implementation of this specification will recognize MIDlet suites signed using PKI as trusted MIDlet suites they MUST be signed and verified according to the formats and processes below.

The MIDlet suite is protected by signing the JAR. The signature and certificates are added to the application descriptor as attributes. The device uses them to verify the signature. The device completes the authentication using a root certificate bound to a protection domain on the device. The details of the processes and formats follow.

References

MIDP 2.0 devices are expected to operate using standard Internet and wireless protocols and techniques for transport and security. The current mechanisms for securing Internet content is based on existing Internet standards for public key cryptography:

- [RFC2437] - PKCS #1 RSA Encryption Version 2.0 (http://www.ietf.org/rfc/rfc2437)
- [RFC2459] - Internet X.509 Public Key Infrastructure (http://www.ietf.org/rfc/rfc2459)
- [RFC2560] - Online Certificate Status Protocol (http://www.ietf.org/rfc/rfc2560)
- [WAPCERT] - WAP-211-WAPCert-20010522-a - WAP Certificate Profile Specification (http://www.wapforum.org/what/technical.htm)

Definition of Terms

The terms *Trusted MIDlet suite*, *Permission*, and *Protection Domain* are defined by Security for MIDP Applications.

The following additional term is defined:

Term	Definition

Protection Domain Root Certificate	A certificate associated with a protection domain that the device implicitly trusts to verify and authorize downloaded MIDlet suites

Signing a MIDlet Suite

The security model involves the MIDlet suite, a signer, and public key certificates. As with any public key system authentication is based on a set of root certificates which are used to verify other certificates. Zero or more root certificates will need to be on the device. Additionally, root certificates may be present in removable media such as SIM(WIM) card/USIM module. Implementations MUST support X.509 Certificates and corresponding algorithms. Devices MAY support additional signing mechanisms and certificate formats.

The signer of the MIDlet suite may be the developer or some entity that is responsible for distributing, supporting, and perhaps billing for its use. The signer will need to have a public key certificate that can be validated to one of the protection domain root certificates on the device. The public key is used to verify the signature on the MIDlet suite. The public key is provided as a RSA X.509 certificate included in the application descriptor.

Attributes defined within the manifest of the JAR are protected by the signature. Attributes defined within the application descriptor are not secured. When an attribute appears in the manifest it *MUST NOT* be overridden by a different value from the application descriptor. For trusted MIDlet suites the value in the application descriptor must be equal to the value of the corresponding attribute in the manifest. If not, the MIDlet suite MUST NOT be installed. The `MIDlet.getAppProperty` method must return the attribute value from the manifest if one is defined. If not, the value from the application descriptor (if any) is returned.

Note that the requirement that attributes values be equal differs from MIDP 1.0 and must be used for applications that are signed and verified by these procedures. For untrusted application descriptors, the MIDP 1.0 rule giving priority to application descriptor attributes over manifest attributes must be followed.

Creating the Signing Certificate

1. The signer will need to be aware of the authorization policy for the device and contact the appropriate certificate authority. For example, the signer may need to send its distinguished name (DN) and public key (normally, packaged in a certificate request) to a certificate authority.
2. The CA creates a RSA X.509 (version 3) certificate and returns it to the signer.
3. If multiple CA's are used then all the signer certificates in the application descriptor MUST contain the same public key.

Insert Certificates into the application descriptor

1. The certificate path includes the signer certificate and any necessary certificates but omitting the root certificate. The root certificate will be found on the device.
2. Each certificate in the path is encoded (using base64 but without line breaks) and inserted into the application descriptor as:

```
MIDlet-Certificate-<n>-<m>: <base64 encoding of a certificate>
```

<n>:= a number equal to 1 for first certification path in the descriptor or 1 greater than the previous number for additional certification paths. This defines the sequence in which the certificates are tested to see if the corresponding root certificate is on the device. See the Authenticating a MIDlet suite section below.
<m>:= a number equal to 1 for the signer's certificate in a certification path or 1 greater than the previous number for any subsequent intermediate certificates.

Creating the RSA SHA-1 signature of the JAR

1. The signature of the JAR is created with the signers private key according to the EMSA-PKCS1-v1_5 encoding method of PKCS #1 version 2.0 standard[RFC2437].

2. The signature is base64 encoded, formatted as a single MIDlet-Jar-RSA-SHA1 attribute without line breaks and inserted in the application descriptor.

```
MIDlet-Jar-RSA-SHA1: <base64 encoding of Jar signature>
```

It should be noted that the signer of the MIDlet suite is responsible to its protection domain root certificate owner for protecting the protection domain stake holder's assets and capabilities and, as such, must exercise due-diligence in checking the MIDlet suite before signing it. In the case where there is a trusted relationship (possibly bound by legal agreements), a protection domain root certificate owner may delegate signing MIDlet suites to a third-party and in some circumstances, the author of the MIDlet.

Authenticating a MIDlet Suite

When an MIDlet suite is downloaded, the device MUST check if authentication is required. If the attribute `MIDlet-Jar-RSA-SHA1` is present in the application descriptor then the JAR MUST be authenticated by verifying the signer certificates and JAR signature as below.

Application descriptors without the `MIDlet-Jar-RSA-SHA1` attribute are not authenticated but are installed and invoked as untrusted MIDlet suites.

Verify Signer Certificate

The certification path consists of the signer certificate from the application descriptor and other certificates as needed up to but not including the root certificate.

1. Get the certification path for the signer certificate from the application descriptor attributes `MIDlet-Certificate-1-<m>` where <m> starts at 1 and is incremented by 1 until there is no attribute with the given name. The value of each attribute is a base64 encoded certificate that will need to be decoded and parsed.
2. Validate the certification path using the basic path validation processes described in RFC2459 using the protection domains as the authoritative source of protection domain root certificates. Bind the MIDlet suite to the protection domain that contains the protection domain root certificate that validates the first chain from signer to root and proceed with installation.
3. If attributes `MIDlet-Certificate-<n>-<m>` with `<n>` greater than 1 are present and full certification path could not be established after verifying MIDlet-Certificate-<1>-<m> certificates, repeatedly perform steps 1 and 2 for the value <n> greater by 1 than the previous value. The results of certificate verification are gathered into the Table 1.

Table 1. Actions upon completion of signer certificate verification.

Result	Action
Attempted to validate <n> paths. No public keys of the issuer for the certificate can be found or none of the certification paths can be validated	Authentication fails, JAR Installation is not allowed.
More than one full certificate path established and validated	Implementation proceeds with the signature verification using the first successfully verified certificate path is used for authentication and authorization.
Only one full certificate path established and validated	Implementation proceeds with the signature verification

Verify the MIDlet Suite JAR

1. Get the public key from the verified signer certificate (above).
2. Get the `MIDlet-Jar-RSA-SHA1` attribute from the application descriptor.
3. Decode the attribute value from base64 yielding a PKCS #1 signature [RFC2437].
4. Use the signer's public key, signature, and SHA-1 digest of the JAR, to verify the signature. If the signature verification fails, reject the application descriptor and MIDlet suite. The implementation MUST NOT install the JAR on failure or allow MIDlets from the MIDlet suite to be invoked.

Once the steps of verifying the certificate, verifying the signature and verifying the JAR all succeed then the MIDlet suite contents are known to be intact and the identity of the signer is known. This process must be performed during installation.

Summary of MIDlet suite source verification results

It is essential that the steps performed to verify the digital signature as described above lead to the proof of the identity of the MIDlet suite signer. The results of the verification have a direct impact on authorization. The following, Table 2, summarizes the states to which the signature verification led and which are further used for authorization at install time.

Table 2. Summary of MIDlet suite source verification

Initial state	Verification result
JAD not present, JAR downloaded	Authentication can not be performed, may install JAR. MIDlet suite is treated as untrusted
JAD present but is JAR is unsigned	Authentication can not be performed, may install JAR. MIDlet suite is treated as untrusted
JAR signed but no root certificate present in the keystore to validate the certificate chain	Authentication can not be performed, JAR installation is not allowed
JAR signed, a certificate on the path is expired	Authentication can not be completed, JAR installation is not allowed
JAR signed, a certificate rejected for reasons other than expiration	JAD rejected, JAR installation is not allowed
JAR signed, certificate path validated but signature verification fails	JAD rejected, JAR installation is not allowed
JAR signed, certificate path validated, signature verified	JAR installation is allowed

Caching of Authentication and Authorization Results

The implementation of the authentication and authorization process may store and transfer the results for subsequent use and MUST ensure that the cached information practically can not be tampered with or otherwise compromised between the time it is computed from the JAR, application descriptor, and authentication information and the authorization information is used.

It is essential that the MIDlet suite and security information used to authenticate and authorize a MIDlet suite is not compromised, for example, by use of removable media or other access to MIDlet suite storage that might be corrupted.

Security in Split-VM Implementations

In environments that make use of a split VM (CLDC 5.4.6), it is possible to implement the security mechanism using JARs but this relies on converting the JAR to the device format when the JAR enters the network while faithfully preserving the semantics of the MIDlet. Once the conversion has happened, the device format of the application must be secured against tampering and retain its authorized permissions. This network security is often based on similar digital signature techniques to MIDlet security and it may be the case that this network security infrastructure is already present and active. If and only if this kind of network security infrastructure already exists and it can support all forms of protection required by this specification (and any future JSRs based on this specification), a permissible implementation of MIDlet Suite Security can be based on authenticating and authorizing the device format of the MIDlet since these implementation formats are the actual executable content that will be used on the device. The details of authenticating and authorizing a device format version of a MIDlet suite are implementation specific and thus not covered by this specification.

MIDP X.509 Certificate Profile for Trusted MIDlet Suites

Secured trusted MIDlet suites utilize the same base certificate profile as does HTTPS. The profile is based on the WAP Certificate Profile, WAP-211-WAPCert-20010522-a [WAPCert] which is based on RFC2459 Internet X.509 Public Key Infrastructure Certificate and CRL Profile [RFC2459]. Refer to the package documentation for `javax.microedition.pki` for details.

Certificate Processing for OTA

Devices MUST recognize the key usage extension and when present verify that the extension has the `digitalSignature` bit set. Devices MUST recognize the critical extended key usage extension and when present verify that the extension contains the `id-kp-codeSigning` object identifier (see RFC2459 sec. 4.2.1.13).

The application descriptor SHOULD NOT include a self-issued root certificate in a descriptor certificate chain. However MIDP devices SHOULD treat the certificate as any other in a chain and NOT explicitly reject a chain with a X.509v3 self-issued CA certificate in its chain.

Certificate Expiration and Revocation

Expiration and revocation of certificates supplied in the application descriptor is checked during the authorization procedure, specifically during certificate path validation. Certificate expiration is checked locally on the device as such information is retrievable from the certificate itself. Certificate expiration verification is an intrinsic and mandatory part of certificate path validation.

Certificate revocation is a more complex check as it requires sending a request to a server and the decision is made based on the received response. Certificate revocation can be performed if the appropriate mechanism is implemented on the device. Such mechanisms are not part of MIDP implementation and hence do not form a part of MIDP 2.0 security framework.

If certificate revocation is implemented in the device, it SHOULD support Online Certificate Status protocol (OCSP) [RFC2560]. If other certificate revocation protocols are supported, support for these other protocols may indicate that a certificate has been revoked; in this case, it is permissible to consider the certificate as revoked regardless of the result returned by the OCSP protocol.

Examples of MIDlet Suite Signing

There are many ways to structure protection domain root certificates and their associated signing policies. These examples are provided to illustrate some of the concepts in this specification and are not meant to limit the ways MIDlet PKI signing can be used. The examples allow MIDlets to be revoked (provided the device supports certificate revocation) but at differing granularities.

Example 1 - Developer Owns Signing Certificate
This encodes the origin of the MIDlet suite into the JAD (via the identity of the signer). If the certificate is revoked, all of the developer's signed MIDlets on every device for every user will have their execution permissions revoked.

1. Developer creates MIDlet network application
2. Developer encodes permissions into JAR manifest and creates final MIDlet JAR
3. Developer generates a private-public key pair with a signing certificate and has the certificate signed by one or more protection domain root certificates
4. The developer's certificate is used to sign the MIDlet JAR and create the associated JAD entries
5. MIDlet JAR can be distributed with a suitably populated JAD and run on a MIDP 2.0 compliant device with the appropriate protection domain root certificate

Example - Protection Domain Stakeholder Owns Signing Certificate
This encodes the signers identity (not the MIDlet suite developer) into the JAD. If the certificate is revoked, all MIDlets signed with this particular certificate will have their execution permissions revoked.

1. Developer creates MIDlet network application
2. Developer encodes permissions into JAR manifest and creates final MIDlet JAR
3. The protection domain stakeholder's signing certificate (not necessarily the root cert) is used to sign the MIDlet JAR and create the associated JAD entries
4. MIDlet JAR can be distributed with a suitably populated JAD and run on a MIDP 2.0 compliant device with the appropriate protection domain root certificate

CHAPTER 5

Package java.lang

Description

MID Profile Language Classes included from Java 2 Standard Edition. In addition to the `java.lang` classes specified in the Connected Limited Device Configuration the Mobile Information Device Profile includes the following class from Java 2 Standard Edition.

- java.lang.IllegalStateException.java

`IllegalStateException`s are thrown when illegal transitions are requested, such as scheduling a `TimerTask` or in the containment of user interface components.

System Functions

The MIDP is based on the Connected, Limited Device Configuration (CLDC). Some features of the CLDC are modified or extended by the MIDP.

System Properties

The MIDP defines the following property values (in addition to those defined in the CLDC specification) that MUST be made available to the application using `java.lang.System.getProperty`:

System Properties Defined by MIDP

System Property	Description
`microedition.locale`	The current locale of the device, may be `null`
`microedition.profiles`	is a blank (Unicode U+0020) separated list of the J2ME profiles that this device supports. For MIDP 2.0 devices, this property MUST contain at least “MIDP-2.0”

Other properties may be available from other profiles or the implementation.

Property microedition.locale

The locale property, if not `null`, MUST consist of the language and MAY optionally also contain the country code, and variant separated by “-” (Unicode U+002D). For example, “fr-FR” or “en-US.” (Note: the MIDP 1.0 specification used the HTTP formatting of language tags as defined in RFC3066 (http://www.ietf.org/rfc/rfc3066.txt) *Tags for the Identification of Languages*. This is different from the J2SE definition for `Locale` printed strings where fields are separated by “_” (Unicode U+005F).)

The language codes MUST be the lower-case, two-letter codes as defined by ISO-639 (http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt).

The country code MUST be the upper-case, two-letter codes as defined by ISO-3166 (http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html).

Application Resource Files

Application resource files are accessed using `getResourceAsStream(String name)` in `java.lang.Class` . In the MIDP specification, `getResourceAsStream` is used to allow resource files to be retrieved from the MIDlet Suite's JAR file.

Resource names refer to the contents of the MIDlet Suite JAR file. Absolute pathnames, beginning with "/" are fully qualified file names within the jar file.

Relative pathnames, not beginning with "/" are relative to the class upon which `getResourceAsStream` is called. Relative names are converted to absolute by prepending a "/" followed by the fully qualified package with "." characters converted to "/" and a separator of "/". The resulting string is reduced to canonical form by applying as many times as possible the following:

- All occurences of "/./" are replaced with "/".
- All occurences of "/segment/../" are replaced with "/" where segment does not contain "/".

The canonical resource name is the absolute pathname of the resource within the JAR.

In no case can the path extend outside the JAR file, and resources outside the JAR file MUST NOT be accessible. For example, using "../../" does NOT point outside the JAR file. If there are any remaining "." or ".." characters they are treated literally in locating the resource. No resource can exist with that name so `null` is returned from `Class.getResourceAsStream`. Also, devices MUST NOT allow classfiles to be read from the JAR file as resources, but all other files MUST be accessible.

System.exit

The behavior of `java.lang.System.exit` MUST throw a `java.lang.SecurityException` when invoked by a MIDlet. The only way a MIDlet can indicate that it is complete is by calling `MIDlet.notifyDestroyed` .

Runtime.exit

The behavior of `java.lang.Runtime.exit` MUST throw a `java.lang.SecurityException` when invoked by a MIDlet. The only way a MIDlet can indicate that it is complete is by calling `MIDlet.notifyDestroyed` .

Since: MIDP 1.0

Class Summary	
Interfaces	
Classes	
Exceptions	
`IllegalStateException`[3]	Signals that a method has been invoked at an illegal or inappropriate time.
[7]	
Errors	

java.lang
IllegalStateException

Declaration

```
public class IllegalStateException extends RuntimeException
```

```
Object
  |
  +--Throwable
        |
        +--Exception
              |
              +--RuntimeException
                    |
                    +--IllegalStateException
```

Description

Signals that a method has been invoked at an illegal or inappropriate time. In other words, the Java environment or Java application is not in an appropriate state for the requested operation.

Since: MIDP 1.0

Member Summary	
Constructors	
	`IllegalStateException()`[37] `IllegalStateException(String s)`[38]

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()`
Methods inherited from class `Throwable`
`getMessage(), printStackTrace(), toString()`

Constructors

IllegalStateException()

Declaration:

```
public IllegalStateException()
```

IllegalStateException(String)

Description:
Constructs an IllegalStateException with no detail message.

IllegalStateException(String)

Declaration:
```
public IllegalStateException(String s)
```

Description:
Constructs an IllegalStateException with the specified detail message. A detail message is a String that describes this particular exception.

Parameters:
`s` - the String that contains a detailed message

CHAPTER 6

Package java.util

Description

MID Profile Utility Classes included from Java 2 Standard Edition. In addition to the `java.util` classes specified in the Connected Limited Device Configuration the Mobile Information Device Profile includes the following classes from Java 2 Standard Edition.

- java.util.Timer
- java.util.TimerTask

Timers provide facility for an application to schedule task for future execution in a background thread. TimerTasks may be scheduled using Timers for one-time execution, or for repeated execution at regular intervals.

Since: MIDP 1.0

Class Summary	
Interfaces	
Classes	
`Timer` 40	A facility for threads to schedule tasks for future execution in a background thread.
`TimerTask` 46	A task that can be scheduled for one-time or repeated execution by a `Timer`.
Exceptions	

java.util
Timer

Declaration

```
public class Timer

Object
  |
  +--java.util.Timer
```

Description

A facility for threads to schedule tasks for future execution in a background thread. Tasks may be scheduled for one-time execution, or for repeated execution at regular intervals.

Corresponding to each `Timer` object is a single background thread that is used to execute all of the timer's tasks, sequentially. Timer tasks should complete quickly. If a timer task takes excessive time to complete, it "hogs" the timer's task execution thread. This can, in turn, delay the execution of subsequent tasks, which may "bunch up" and execute in rapid succession when (and if) the offending task finally completes.

After the last live reference to a `Timer` object goes away *and* all outstanding tasks have completed execution, the timer's task execution thread terminates gracefully (and becomes subject to garbage collection). However, this can take arbitrarily long to occur. By default, the task execution thread does not run as a *daemon thread*, so it is capable of keeping an application from terminating. If a caller wants to terminate a timer's task execution thread rapidly, the caller should invoke the timer's `cancel` method.

If the timer's task execution thread terminates unexpectedly, any further attempt to schedule a task on the timer will result in an `IllegalStateException`, as if the timer's `cancel` method had been invoked.

This class is thread-safe: multiple threads can share a single `Timer` object without the need for external synchronization.

This class does *not* offer real-time guarantees: it schedules tasks using the `Object.wait(long)` method. The resolution of the Timer is implementation and device dependent.

Timers function only within a single VM and are cancelled when the VM exits. When the VM is started no timers exist, they are created only by application request.

Since: MIDP 1.0

See Also: `TimerTask`[46], `Object.wait(long)`

Member Summary	
Constructors	
	`Timer()`[41]
Methods	
`void`	`cancel()`[41]
`void`	`schedule(TimerTask task, Date time)`[41]
`void`	`schedule(TimerTask task, Date firstTime, long period)`[42]
`void`	`schedule(TimerTask task, long delay)`[42]
`void`	`schedule(TimerTask task, long delay, long period)`[43]

Member Summary	
void	scheduleAtFixedRate(TimerTask task, Date firstTime, long period)[43]
void	scheduleAtFixedRate(TimerTask task, long delay, long period)[44]

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Constructors

Timer()

Declaration:
`public Timer()`

Description:
Creates a new timer. The associated thread does *not* run as a daemon thread, which may prevent an application from terminating.

See Also: `Thread`, `cancel()`[41]

Methods

cancel()

Declaration:
`public void cancel()`

Description:
Terminates this timer, discarding any currently scheduled tasks. Does not interfere with a currently executing task (if it exists). Once a timer has been terminated, its execution thread terminates gracefully, and no more tasks may be scheduled on it.

Note that calling this method from within the run method of a timer task that was invoked by this timer absolutely guarantees that the ongoing task execution is the last task execution that will ever be performed by this timer.

This method may be called repeatedly; the second and subsequent calls have no effect.

schedule(TimerTask, Date)

Declaration:
`public void schedule(java.util.TimerTask`[46] `task, java.util.Date time)`

schedule(TimerTask, Date, long)

Description:
Schedules the specified task for execution at the specified time. If the time is in the past, the task is scheduled for immediate execution.

Parameters:
`task` - task to be scheduled.

`time` - time at which task is to be executed.

Throws:
`IllegalArgumentException` - if `time.getTime()` is negative.

`IllegalStateException`[37] - if task was already scheduled or cancelled, timer was cancelled, or timer thread terminated.

schedule(TimerTask, Date, long)

Declaration:
```
public void schedule(java.util.TimerTask[46] task, java.util.Date firstTime, long period)
```

Description:
Schedules the specified task for repeated *fixed-delay execution*, beginning at the specified time. Subsequent executions take place at approximately regular intervals, separated by the specified period.

In fixed-delay execution, each execution is scheduled relative to the actual execution time of the previous execution. If an execution is delayed for any reason (such as garbage collection or other background activity), subsequent executions will be delayed as well. In the long run, the frequency of execution will generally be slightly lower than the reciprocal of the specified period (assuming the system clock underlying `Object.wait(long)` is accurate).

Fixed-delay execution is appropriate for recurring activities that require “smoothness.” In other words, it is appropriate for activities where it is more important to keep the frequency accurate in the short run than in the long run. This includes most animation tasks, such as blinking a cursor at regular intervals. It also includes tasks wherein regular activity is performed in response to human input, such as automatically repeating a character as long as a key is held down.

Parameters:
`task` - task to be scheduled.

`firstTime` - First time at which task is to be executed.

`period` - time in milliseconds between successive task executions.

Throws:
`IllegalArgumentException` - if `time.getTime()` is negative.

`IllegalStateException`[37] - if task was already scheduled or cancelled, timer was cancelled, or timer thread terminated.

schedule(TimerTask, long)

Declaration:
```
public void schedule(java.util.TimerTask[46] task, long delay)
```

Description:
Schedules the specified task for execution after the specified delay.

Parameters:
`task` - task to be scheduled.

`delay` - delay in milliseconds before task is to be executed. Note that the actual delay may be different than the amount requested since the resolution of the Timer is implementation and device dependent.

Throws:

`IllegalArgumentException` - if `delay` is negative, or `delay + System.currentTimeMillis()` is negative.

`IllegalStateException`[37] - if task was already scheduled or cancelled, or timer was cancelled.

schedule(TimerTask, long, long)

Declaration:

```
public void schedule(java.util.TimerTask[46] task, long delay, long period)
```

Description:

Schedules the specified task for repeated *fixed-delay execution*, beginning after the specified delay. Subsequent executions take place at approximately regular intervals separated by the specified period. Note that the actual delay may be different than the amount requested since the resolution of the Timer is implementation and device dependent.

In fixed-delay execution, each execution is scheduled relative to the actual execution time of the previous execution. If an execution is delayed for any reason (such as garbage collection or other background activity), subsequent executions will be delayed as well. In the long run, the frequency of execution will generally be slightly lower than the reciprocal of the specified period (assuming the system clock underlying `Object.wait(long)` is accurate).

Fixed-delay execution is appropriate for recurring activities that require “smoothness.” In other words, it is appropriate for activities where it is more important to keep the frequency accurate in the short run than in the long run. This includes most animation tasks, such as blinking a cursor at regular intervals. It also includes tasks wherein regular activity is performed in response to human input, such as automatically repeating a character as long as a key is held down.

Parameters:

`task` - task to be scheduled.

`delay` - delay in milliseconds before task is to be executed. Note that the actual delay may be different than the amount requested since the resolution of the Timer is implementation and device dependent.

`period` - time in milliseconds between successive task executions.

Throws:

`IllegalArgumentException` - if `delay` is negative, or `delay + System.currentTimeMillis()` is negative.

`IllegalStateException`[37] - if task was already scheduled or cancelled, timer was cancelled, or timer thread terminated.

scheduleAtFixedRate(TimerTask, Date, long)

Declaration:

```
public void scheduleAtFixedRate(java.util.TimerTask[46] task, java.util.Date firstTime,
            long period)
```

Description:

Schedules the specified task for repeated *fixed-rate execution*, beginning at the specified time. Subsequent executions take place at approximately regular intervals, separated by the specified period.

In fixed-rate execution, each execution is scheduled relative to the scheduled execution time of the initial execution. If an execution is delayed for any reason (such as garbage collection or other background activity), two or more executions will occur in rapid succession to “catch up.” In the long run, the frequency of execution will be exactly the reciprocal of the specified period (assuming the system clock underlying `Object.wait(long)` is accurate).

Fixed-rate execution is appropriate for recurring activities that are sensitive to *absolute* time, such as ringing a chime every hour on the hour, or running scheduled maintenance every day at a particular time. It is also appropriate for for recurring activities where the total time to perform a fixed number of executions is important, such as a countdown timer that ticks once every second for ten seconds. Finally, fixed-rate execution is appropriate for scheduling multiple repeating timer tasks that must remain synchronized with respect to one another.

Parameters:

`task` - task to be scheduled.

`firstTime` - First time at which task is to be executed.

`period` - time in milliseconds between successive task executions.

Throws:

`IllegalArgumentException` - if `time.getTime()` is negative.

`IllegalStateException`[37] - if task was already scheduled or cancelled, timer was cancelled, or timer thread terminated.

scheduleAtFixedRate(TimerTask, long, long)

Declaration:

```
public void scheduleAtFixedRate(java.util.TimerTask[46] task, long delay, long period)
```

Description:

Schedules the specified task for repeated *fixed-rate execution*, beginning after the specified delay. Subsequent executions take place at approximately regular intervals, separated by the specified period.

In fixed-rate execution, each execution is scheduled relative to the scheduled execution time of the initial execution. If an execution is delayed for any reason (such as garbage collection or other background activity), two or more executions will occur in rapid succession to “catch up.” In the long run, the frequency of execution will be exactly the reciprocal of the specified period (assuming the system clock underlying `Object.wait(long)` is accurate).

Fixed-rate execution is appropriate for recurring activities that are sensitive to *absolute* time, such as ringing a chime every hour on the hour, or running scheduled maintenance every day at a particular time. It is also appropriate for for recurring activities where the total time to perform a fixed number of executions is important, such as a countdown timer that ticks once every second for ten seconds. Finally, fixed-rate execution is appropriate for scheduling multiple repeating timer tasks that must remain synchronized with respect to one another.

Parameters:

`task` - task to be scheduled.

`delay` - delay in milliseconds before task is to be executed. Note that the actual delay may be different than the amount requested since the resolution of the Timer is implementation and device dependent.

`period` - time in milliseconds between successive task executions.

Throws:

`IllegalArgumentException` - if `delay` is negative, or `delay + System.currentTimeMillis()` is negative.

`IllegalStateException`[37] - if task was already scheduled or cancelled, timer was cancelled, or timer thread terminated.

java.util
TimerTask

Declaration
```
public abstract class TimerTask implements Runnable

Object
  |
  +--java.util.TimerTask
```

All Implemented Interfaces: `Runnable`

Description
A task that can be scheduled for one-time or repeated execution by a `Timer`.

Since: MIDP 1.0

See Also: `Timer` 40

Member Summary	
Constructors	
protected	`TimerTask()` 46
Methods	
boolean	`cancel()` 47
abstract void	`run()` 47
long	`scheduledExecutionTime()` 47

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Constructors

TimerTask()

Declaration:
```
protected TimerTask()
```

Description:
Creates a new timer task.

Methods

cancel()

Declaration:
```
public boolean cancel()
```

Description:
Cancels this timer task. If the task has been scheduled for one-time execution and has not yet run, or has not yet been scheduled, it will never run. If the task has been scheduled for repeated execution, it will never run again. (If the task is running when this call occurs, the task will run to completion, but will never run again.)

Note that calling this method from within the `run` method of a repeating timer task absolutely guarantees that the timer task will not run again.

This method may be called repeatedly; the second and subsequent calls have no effect.

Returns: true if this task is scheduled for one-time execution and has not yet run, or this task is scheduled for repeated execution. Returns false if the task was scheduled for one-time execution and has already run, or if the task was never scheduled, or if the task was already cancelled. (Loosely speaking, this method returns `true` if it prevents one or more scheduled executions from taking place.)

run()

Declaration:
```
public abstract void run()
```

Description:
The action to be performed by this timer task.

Specified By: `run` in interface `Runnable`

scheduledExecutionTime()

Declaration:
```
public long scheduledExecutionTime()
```

Description:
Returns the *scheduled* execution time of the most recent *actual* execution of this task. (If this method is invoked while task execution is in progress, the return value is the scheduled execution time of the ongoing task execution.)

This method is typically invoked from within a task's run method, to determine whether the current execution of the task is sufficiently timely to warrant performing the scheduled activity:

```
public void run() {
    if (System.currentTimeMillis() - scheduledExecutionTime() >=
        MAX_TARDINESS)
            return;  // Too late; skip this execution.
    // Perform the task
}
```

This method is typically *not* used in conjunction with *fixed-delay execution* repeating tasks, as their scheduled execution times are allowed to drift over time, and so are not terribly significant.

scheduledExecutionTime()

Returns: the time at which the most recent execution of this task was scheduled to occur, in the format returned by Date.getTime(). The return value is undefined if the task has yet to commence its first execution.

See Also: `Date.getTime()`

CHAPTER 7

Package javax.microedition.io

Description

MID Profile includes networking support based on the `Generic Connection` framework from the *Connected, Limited Device Configuration.*

HTTP Networking

In addition to the `javax.microedition.io` classes specified in the *Connected Limited Device Configuration* the *Mobile Information Device Profile* includes the following interface for the HTTP access. An `HttpConnection` is returned from `Connector.open()` when an `"http://"` connection string is accessed.

- `javax.microedition.io.HttpConnection`

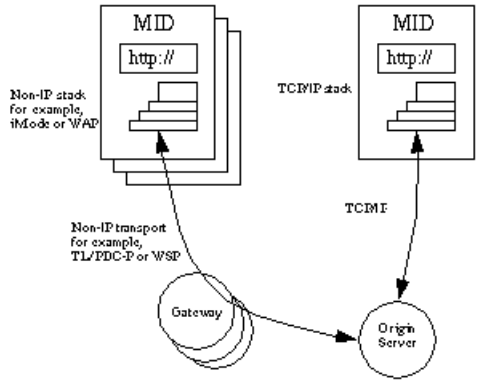
The MIDP extends the connectivity support provided by the Connected, Limited Device Configuration (CLDC) with specific functionality for the *GenericConnection* framework. The MIDP supports a subset of the HTTP protocol, which can be implemented using both IP protocols such as TCP/IP and non-IP protocols such as WAP and i-Mode, utilizing a gateway to provide access to HTTP servers on the Internet.

The *GenericConnection* framework is used to support client-server and datagram networks. Using only the protocols specified by the MIDP will allow the application to be portable to all MIDs. MIDP implementations MUST provide support for accessing HTTP 1.1 servers and services.

There are wide variations in wireless networks. It is the joint responsibility of the device and the wireless network to provide the application service. It may require a *gateway* that can bridge between the wireless transports specific to the network and the wired Internet. The client application and the Internet server MUST NOT need to be required to know either that non-IP networks are being used or the characteristics of those networks. While the client and server MAY both take advantage of such knowledge to optimize their transmissions, they MUST NOT be required to do so.

For example, a MID MAY have no in-device support for the Internet Protocol (IP). In this case, it would utilize a gateway to access the Internet, and the gateway would be responsible for some services, such as DNS name resolution for Internet URLs. The device and network may define and implement security and network access policies that restrict access.

HTTP Network Connection



The *GenericConnection* framework from the CLDC provides the base stream and content interfaces. The interface *HttpConnection* provides the additional functionality needed to set request headers, parse response headers, and perform other HTTP specific functions.

The interface MUST support:

HTTP 1.1

Each device implementing the MIDP MUST support opening connections using the following URL schemes (RFC2396 Uniform Resource Identifiers (URI): Generic Syntax)

"http" as defined by RFC2616 *Hypertext Transfer Protocol —- HTTP/1.1*

Each device implementing the MIDP MUST support the full specification of RFC2616 HEAD, GET and POST requests. The implementation MUST also support the absolute forms of URIs.

The implementation MUST pass all request headers supplied by the application and response headers as supplied by the network server. The ordering of request and response headers MAY be changed. While the headers may be transformed in transit, they MUST be reconstructed as equivalent headers on the device and server. Any transformations MUST be transparent to the application and origin server. The HTTP implementation does not automatically include any headers. The application itself is responsible for setting any request headers that it needs.

Connections may be implemented with any suitable protocol providing the ability to reliably transport the HTTP headers and data.(RFC2616 takes great care to not to mandate TCP streams as the only required transport mechanism.)

HTTP Request Headers

The HTTP 1.1 specification provides a rich set of request and response headers that allow the application to negotiate the form, format, language, and other attributes of the content retrieved. In the MIDP, the application is responsible for selection and processing of request and response headers. Only the *User-Agent* header is described in detail. Any other header that is mutually agreed upon with the server may be used.

User-Agent and Accept-Language Request Headers

For the MIDP, a simple *User-Agent* field may be used to identify the current device. As specified by RFC2616, the field contains blank separated features where the feature contains a name and optional version number.

The application is responsible for formatting and requesting that the *User-Agent* field be included in HTTP requests via the *setRequestProperty* method in the interface *javax.microedition.io.HttpConnection*. It can supply

any application-specific features that are appropriate, in addition to any of the profile-specific request header values listed below.

Applications are not required to be loaded onto the device using HTTP. But if they are, then the *User-Agent* request header should be included in requests to load an application descriptor or application JAR file onto the device. This will allow the server to provide the most appropriate application for the device.

The user-agent and accept-language fields SHOULD contain the following features as defined by system properties using *java.lang.System.getProperty*. If multiple values are present they will need to be reformatted into individual fields in the request header.

System Properties Used for User-Agent and Accept-Language Request Headers

System Property	Description
microedition.profiles	A blank (Unicode U+0020) separated list of the J2ME profiles that this device supports. For MIDP 2.0 devices, this property MUST contain at least "MIDP-2.0".
microedition.configuration	The J2ME configuration supported by this device. For example, "CLDC-1.0."
microedition.locale	The name of the current locale on this device. For example, "en-US."

HTTP Request Header Example
User-Agent: Profile/MIDP-2.0 Configuration/CLDC-1.0
Accept-Language: en-US

StreamConnection Behavior

All MIDP `StreamConnections` have one underlying `InputStream` and one `OutputStream`. Opening a `DataInputStream` counts as opening an `InputStream` and opening a `DataOutputStream` counts as opening an `OutputStream`. Trying to open another `InputStream` or another `OutputStream` from a `StreamConnections` causes an `IOException`. Trying to open `InputStream` or `OutputStream` after they have been closed causes an `IOException`.

After calling the `close` method, regardless of open streams, further method calls to connection will result in `IOExceptions` for those methods that are declared to throw `IOExceptions`. For the methods that do not throw exceptions, unknown results may be returned.

The methods of `StreamConnections` are not synchronized. The only stream method that can be called safely in another thread is `close`. When `close` is invoked on a stream that is excuting in another thread, any pending I/O method MUST throw an `InterruptedIOException`. In the above case implementations SHOULD try to throw the exception in a timely manner. When all open streams have been closed, and when the `StreamConnections` is closed, any pending I/O operations MUST be interrupted in a timely manner.

Secure Networking

Since the MIDP 2.0 release additional interfaces are available for secure communication with WWW network services. Secure interfaces are provided by HTTPS and SSL/TLS protocol access over the IP network. Refer to the package documentation of `javax.microedition.pki` for the details of certificate profile that applies to secure connections. An `HttpsConnection` is returned from `Connector.open()` when an `"https:/`

/" connection string is accessed. A SecureConnection is returned from Connector.open() when an "ssl://" connection string is accessed.

- javax.microedition.io.HttpsConnection
- javax.microedition.io.SecureConnection
- javax.microedition.io.SecurityInfo
- javax.microedition.pki.Certificate
- javax.microedition.pki.CertificateException

Low Level IP Networking

Since the MIDP 2.0 release, the MIDP specification also includes optional networking support for TCP/IP sockets and UDP/IP datagrams. For each of the following schemes, a host is specified for an outbound connection and the host is omitted for an inbound connection. The host can be a host name, a literal IPv4 address or a literal IPv6 addresss (according to RFC2732 square bracket characters '[' ']' may be used to designate an IPv6 address in URL strings). Implementations MUST be able to parse the URL string and recognize the address format used, but are not required to support all address formats and associated protocols.

When the host and port number are both omitted from the socket or datagram connection, the system will allocate an available port. The host and port numbers allocated in this fashion can be discovered using the getLocalAddress and getLocalPort methods. The colon (:) may be omitted when the connection string does not include the port parameter.

A SocketConnection is returned from Connector.open() when a "socket://host:port" connection string is accessed. A ServerSocketConnection is returned from Connector.open() when a "socket://:port" connection string is accessed. A UDPDatagramConnection is returned from Connector.open() when a "datagram://host:port" connection string is accessed.

- javax.microedition.io.SocketConnection
- javax.microedition.io.ServerSocketConnection
- javax.microedition.io.DatagramConnection
- javax.microedition.io.Datagram
- javax.microedition.io.UDPDatagramConnection

Push Applications

A PushRegistry is available in the MIDP 2.0 release which provides a MIDlet with a means of registering for network connection events, which may be delivered when the application is not currently running.

- javax.microedition.io.PushRegistry

Serial Port Communications

A CommConnection is available in the MIDP 2.0 release which provides a MIDlet with a means of registering for network accessing a local serial port as a stream connection.

- javax.microedition.io.CommConnection

Security of Networking Functions

The security model is found in the package javax.microedition.midlet and provides a framework that allows APIs and functions to be restricted to MIDlet suites that have been granted permissions either by

signing or explicitly by the user. (See Security for MIDlet suites for details about granting specific permissions to a `MIDlet` suite.)

The risks associated with a MIDlet suite's use of the network are related the potential for network abuse and to costs to the device owner since network use may result in charges. MIDP 2.0 provides a security framework in which network functions can be protected and allowed only to those applications that have requested and been granted appropriate permissions.

Each protocol is accessed by invoking `javax.microedition.io.Connector.open` with a URI including the protocol and arguments. The permissions below allow access to be granted individually to protocols. The functionality of the protocols is specified by subclasses of `Connection` interface that defines the syntax of the URI and any protocol specific methods. Devices are NOT REQUIRED to implement every protocol. If a protocol is implemented, the security framework specifies the naming of permissions according to the package and class name of the APIs used to access the protocol extended with the protocol name. The API providing access is `javax.microedition.io.Connector.open`. The table below defines the corresponding permissions for the protocols defined within this specification.

Permission	Protocol
`javax.microedition.io.Connector.http`	`http`
`javax.microedition.io.Connector.https`	`https`
`javax.microedition.io.Connector.datagram`	`datagram`
`javax.microedition.io.Connector.datagramreceiver`	`datagram server (without host)`
`javax.microedition.io.Connector.socket`	`socket`
`javax.microedition.io.Connector.serversocket`	`server socket (without host)`
`javax.microedition.io.Connector.ssl`	`ssl`
`javax.microedition.io.Connector.comm`	`comm`

Security of PushRegistry

The `PushRegistry` is protected using the security framework and permissions. The MIDlet suite must have the `javax.microedition.io.PushRegistry` permission to register an alarm based launch, to register dynamically using the `PushRegistry`, to make a static registration in the application descriptor and to determine if the user needs to be prompted prior to invoking MIDlet suite in response to a Push connection event or alarm. The protection domain defines the general behavior for user permissions with the interaction modes of "oneshot", "session", and "blanket". For the `PushRegistry` and the AMS, launching behavior is specialized:

- Oneshot: The user is prompted before the MIDlet suite is invoked to handle a push event or alarm and for each `PushRegistry` request; for example to register an alarm or a connection.
- Session: The user is prompted before the MIDlet suite is invoked to handle a push event or alarm, or before the first `PushRegistry` request; for example to register an alarm or a connection. Subsequently, when a MIDlet uses the `PushRegistry` the user is not prompted.
- Blanket: The user is prompted only once during installation, before the first time the MIDlet suite is invoked to handle a push event or alarm, or uses the `PushRegistry`.

The push mechanism uses protocols in which the device is acting as the server and connections can be accepted from other elements of the network. To use the push mechanisms the MIDlet suite will need the permission to use the server connection. For example, to register a chat program that can be started via push might use the following attributes in the manifest:

```
MIDlet-Push-1: socket://:79, com.sun.example.SampleChat, *
MIDlet-
Permissions: javax.microedition.io.PushRegistry, javax.microedition.io.Connector.servers
ocket
```

Since: MIDP 1.0

Class Summary

Interfaces

`CommConnection` 55	This interface defines a logical serial port connection.
`HttpConnection` 65	This interface defines the necessary methods and constants for an HTTP connection.
`HttpsConnection` 85	This interface defines the necessary methods and constants to establish a secure network connection.
`SecureConnection` 100	This interface defines the secure socket stream connection.
`SecurityInfo` 103	This interface defines methods to access information about a secure network connection.
`ServerSocketConnection` 105	This interface defines the server socket stream connection.
`SocketConnection` 108	This interface defines the socket stream connection.
`UDPDatagramConnection` 113	This interface defines a datagram connection which knows it's local end point address.

Classes

`Connector` 60	Factory class for creating new Connection objects.
`PushRegistry` 89	The `PushRegistry` maintains a list of inbound connections.

Exceptions

javax.microedition.io
CommConnection

Declaration
`public interface` **`CommConnection extends StreamConnection`**

All Superinterfaces: `Connection`, `InputConnection`, `OutputConnection`, `StreamConnection`

Description
This interface defines a logical serial port connection. A “logical” serial port is defined as a logical connection through which bytes are transferring serially. The logical serial port is defined within the underlying operating system and may not necessarily correspond to a physical RS-232 serial port. For instance, IrDA IRCOMM ports can commonly be configured as a logical serial port within the operating system so that it can act as a “logical” serial port.

A comm port is accessed using a Generic Connection Framework string with an explicit port identifier and embedded configuration parameters, each separated with a semi-colon (;).

Only one application may be connected to a particular serial port at a given time. An `java.io.IOException` is thrown, if an attempt is made to open the serial port with `Connector.open()` and the connection is already open.

A URI with the type and parameters is used to open the connection. The scheme (defined in RFC 2396) must be:

```
comm:<port identifier>[<optional parameters>]
```

The first parameter must be a port identifier, which is a logical device name. These identifiers are most likely device specific and should be used with care.

The valid identifiers for a particular device and OS can be queried through the method `System.getProperty()` using the key *“microedition.commports”*. A comma separated list of ports is returned which can be combined with a `comm:` prefix as the URL string to be used to open a serial port connection. (See port naming convention below.)

Any additional parameters must be separated by a semi-colon (;) and spaces are not allowed in the string. If a particular optional parameter is not applicable to a particular port, the parameter MAY be ignored. The port identifier MUST NOT contain a semi-colon (;).

Legal parameters are defined by the definition of the parameters below. Illegal or unrecognized parameters cause an `IllegalArgumentException`. If the value of a parameter is supported by the device, it must be honored. If the value of a parameter is not supported a `java.io.IOException` is thrown. If a `baudrate` parameter is requested, it is treated in the same way that the `setBaudRate` method handles baudrates. e.g., if the baudrate requested is not supported the system MAY substitute a valid baudrate, which can be discovered using the `getBaudRate` method.

Optional Parameters

Parameter	Default	Description
`baudrate`	`platform dependent`	The speed of the port.
`bitsperchar`	`8`	The number bits per character(`7` or `8`).
`stopbits`	`1`	The number of stop bits per char(`1` or `2`)
`parity`	`none`	The parity can be `odd`, `even`, or `none`.
`blocking`	`on`	If `on`, wait for a full buffer when reading.
`autocts`	`on`	If `on`, wait for the CTS line to be on before writing.
`autorts`	`on`	If `on`, turn on the RTS line when the input buffer is not full. If `off`, the RTS line is always on.

BNF Format for Connector.open() string

The URI must conform to the BNF syntax specified below. If the URI does not conform to this syntax, an `IllegalArgumentException` is thrown.

<comm_connection_string>	::= “**comm:**”<port_id>[<options_list>] ;
<port_id>	::= *string of alphanumeric characters*
<options_list>	::= *(<baud_rate_string>| <bitsperchar>| <stopbits>| <parity>| <blocking>| <autocts>| <autorts>) ; ; if an option duplicates a previous option in the ; option list, that option overrides the previous ; option
<baud_rate_string>	::= “**;baudrate=**”<baud_rate>
<baud_rate>	::= *string of digits*
<bitsperchar>	::= “**;bitsperchar=**”<bit_value>
<bit_value>	::= “7” | “8”
<stopbits>	::= “**;stopbits=**”<stop_value>
<stop_value>	::= “1” | “2”
<parity>	::= “**;parity=**”<parity_value>
<parity_value>	::= “even” | “odd” | “none”
<blocking>	::= “**;blocking=**”<on_off>
<autocts>	::= “**;autocts=**”<on_off>
<autorts>	::= “**;autorts=**”<on_off>
<on_off>	::= “on” | “off”

Security

Access to serial ports is restricted to prevent unauthorized transmission or reception of data. The security model applied to the serial port connection is defined in the implementing profile. The security model may be applied on the invocation of the `Connector.open()` method with a valid serial port connection string. Should the application not be granted access to the serial port through the profile authorization scheme, a `java.lang.SecurityException` will be thrown from the `Connector.open()` method. The security model MAY also be applied during execution, specifically when the methods `openInputStream()`, `openDataInputStream()`, `openOutputStream()`, and `openDataOutputStream()` are invoked.

Examples

The following example shows how a `CommConnection` would be used to access a simple loopback program.

```
CommConnection cc = (CommConnection)
           Connector.open(“comm:com0;baudrate=19200”);
int baudrate = cc.getBaudRate();
InputStream is  = cc.openInputStream();
OutputStream os = cc.openOutputStream();
int ch = 0;
while(ch != 'Z') {
    os.write(ch);
    ch = is.read();
    ch++;
}
is.close();
os.close();
cc.close();
```

The following example shows how a `CommConnection` would be used to discover available comm ports.

```
String port1;
String ports = System.getProperty(“microedition.commports”);
int comma = ports.indexOf(',');
if (comma > 0) {
    // Parse the first port from the available ports list.
    port1 = ports.substring(0, comma);
} else {
    // Only one serial port available.
    port1 =ports;
}
```

Recommended Port Naming Convention

Logical port names can be defined to match platform naming conventions using any combination of alphanumeric characters. However, it is recommended that ports be named consistently among the implementations of this class according to a proposed convention. VM implementations should follow the following convention:
Port names contain a text abbreviation indicating port capabilities followed by a sequential number for the port. The following device name types should be used:

- COM#, where COM is for RS-232 ports and # is a number assigned to the port
- IR#, where IR is for IrDA IRCOMM ports and # is a number assigned to the port

This naming scheme allows API users to generally determine the type of port that they would like to use. For instance, if a application desires to “beam” a piece of data, the app could look for “IR#” ports for opening the connection. The alternative is a trial and error approach with all available ports.

Since: MIDP 2.0

Member Summary
Methods
`int getBaudRate()`[58]
`int setBaudRate(int baudrate)`[58]

Inherited Member Summary
Methods inherited from interface `Connection`
`close()`
Methods inherited from interface `InputConnection`
`openDataInputStream(), openInputStream()`
Methods inherited from interface `OutputConnection`
`openDataOutputStream(), openOutputStream()`

Methods

getBaudRate()

Declaration:
`public int getBaudRate()`

Description:
Gets the baudrate for the serial port connection.

Returns: the baudrate of the connection

See Also: `setBaudRate(int)`[58]

setBaudRate(int)

Declaration:
`public int setBaudRate(int baudrate)`

Description:
Sets the baudrate for the serial port connection. If the requested `baudrate` is not supported on the platform, then the system MAY use an alternate valid setting. The alternate value can be accessed using the `getBaudRate` method.

Parameters:
`baudrate` - the baudrate for the connection

Returns: the previous baudrate of the connection

See Also: `getBaudRate()`[58]

javax.microedition.io
Connector

Declaration

```
public class Connector

Object
  |
  +--javax.microedition.io.Connector
```

Description

Factory class for creating new Connection objects.

The creation of Connections is performed dynamically by looking up a protocol implementation class whose name is formed from the platform name (read from a system property) and the protocol name of the requested connection (extracted from the parameter string supplied by the application programmer.) The parameter string that describes the target should conform to the URL format as described in RFC 2396. This takes the general form:

`{scheme}:[{target}][{parms}]`

where `{scheme}` is the name of a protocol such as *http*.

The `{target}` is normally some kind of network address.

Any `{parms}` are formed as a series of equates of the form ";x=y". Example: ";type=a".

An optional second parameter may be specified to the open function. This is a mode flag that indicates to the protocol handler the intentions of the calling code. The options here specify if the connection is going to be read (READ), written (WRITE), or both (READ_WRITE). The validity of these flag settings is protocol dependent. For instance, a connection for a printer would not allow read access, and would throw an IllegalArgumentException. If the mode parameter is not specified, READ_WRITE is used by default.

An optional third parameter is a boolean flag that indicates if the calling code can handle timeout exceptions. If this flag is set, the protocol implementation may throw an InterruptedIOException when it detects a timeout condition. This flag is only a hint to the protocol handler, and it does not guarantee that such exceptions will actually be thrown. If this parameter is not set, no timeout exceptions will be thrown.

Because connections are frequently opened just to gain access to a specific input or output stream, four convenience functions are provided for this purpose. See also: `DatagramConnection` for information relating to datagram addressing

Since: CLDC 1.0

Member Summary	
Fields	
static int	READ[61]
static int	READ_WRITE[61]
static int	WRITE[61]
Methods	
static Connection	open(String name)[62]

Member Summary	
static Connection	open(String name, int mode)[62]
static Connection	open(String name, int mode, boolean timeouts)[62]
static java.io.DataInputStream	openDataInputStream(String name)[63]
static java.io.DataOutputStream	openDataOutputStream(String name)[63]
static java.io.InputStream	openInputStream(String name)[64]
static java.io.OutputStream	openOutputStream(String name)[64]

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Fields

READ

Declaration:
`public static final int READ`

Description:
Access mode READ.

The value `1` is assigned to `READ`.

READ_WRITE

Declaration:
`public static final int READ_WRITE`

Description:
Access mode READ_WRITE.

The value `3` is assigned to `READ_WRITE`.

WRITE

Declaration:
`public static final int WRITE`

Description:
Access mode WRITE.

The value `2` is assigned to `WRITE`.

Methods

open(String)

Declaration:
```
public static javax.microedition.io.Connection open(String name)
            throws IOException
```

Description:
Create and open a Connection.

Parameters:
`name` - The URL for the connection.

Returns: A new Connection object.

Throws:
`IllegalArgumentException` - If a parameter is invalid.

`ConnectionNotFoundException` - If the requested connection cannot be made, or the protocol type does not exist.

`java.io.IOException` - If some other kind of I/O error occurs.

`SecurityException` - If a requested protocol handler is not permitted.

open(String, int)

Declaration:
```
public static javax.microedition.io.Connection open(String name, int mode)
            throws IOException
```

Description:
Create and open a Connection.

Parameters:
`name` - The URL for the connection.

`mode` - The access mode.

Returns: A new Connection object.

Throws:
`IllegalArgumentException` - If a parameter is invalid.

`ConnectionNotFoundException` - If the requested connection cannot be made, or the protocol type does not exist.

`java.io.IOException` - If some other kind of I/O error occurs.

`SecurityException` - If a requested protocol handler is not permitted.

open(String, int, boolean)

Declaration:
```
public static javax.microedition.io.Connection open(String name, int mode,
            boolean timeouts)
            throws IOException
```

Description:
Create and open a Connection.

Parameters:

`name` - The URL for the connection

`mode` - The access mode

`timeouts` - A flag to indicate that the caller wants timeout exceptions

Returns: A new Connection object

Throws:

`IllegalArgumentException` - If a parameter is invalid.

`ConnectionNotFoundException` - if the requested connection cannot be made, or the protocol type does not exist.

`java.io.IOException` - If some other kind of I/O error occurs.

`SecurityException` - If a requested protocol handler is not permitted.

openDataInputStream(String)

Declaration:

```
public static java.io.DataInputStream openDataInputStream(String name)
            throws IOException
```

Description:

Create and open a connection input stream.

Parameters:

`name` - The URL for the connection.

Returns: A DataInputStream.

Throws:

`IllegalArgumentException` - If a parameter is invalid.

`ConnectionNotFoundException` - If the connection cannot be found.

`java.io.IOException` - If some other kind of I/O error occurs.

`SecurityException` - If access to the requested stream is not permitted.

openDataOutputStream(String)

Declaration:

```
public static java.io.DataOutputStream openDataOutputStream(String name)
            throws IOException
```

Description:

Create and open a connection output stream.

Parameters:

`name` - The URL for the connection.

Returns: A DataOutputStream.

Throws:

`IllegalArgumentException` - If a parameter is invalid.

`ConnectionNotFoundException` - If the connection cannot be found.

`java.io.IOException` - If some other kind of I/O error occurs.

`SecurityException` - If access to the requested stream is not permitted.

openInputStream(String)

Declaration:

```
public static java.io.InputStream openInputStream(String name)
            throws IOException
```

Description:
Create and open a connection input stream.

Parameters:

`name` - The URL for the connection.

Returns: An InputStream.

Throws:

`IllegalArgumentException` - If a parameter is invalid.

`ConnectionNotFoundException` - If the connection cannot be found.

`java.io.IOException` - If some other kind of I/O error occurs.

`SecurityException` - If access to the requested stream is not permitted.

openOutputStream(String)

Declaration:

```
public static java.io.OutputStream openOutputStream(String name)
            throws IOException
```

Description:
Create and open a connection output stream.

Parameters:

`name` - The URL for the connection.

Returns: An OutputStream.

Throws:

`IllegalArgumentException` - If a parameter is invalid.

`ConnectionNotFoundException` - If the connection cannot be found.

`java.io.IOException` - If some other kind of I/O error occurs.

`SecurityException` - If access to the requested stream is not permitted.

javax.microedition.io
HttpConnection

Declaration
`public interface` **`HttpConnection extends ContentConnection`**

All Superinterfaces: `Connection`, `ContentConnection`, `InputConnection`, `OutputConnection`, `StreamConnection`

All Known Subinterfaces: `HttpsConnection`[85]

Description
This interface defines the necessary methods and constants for an HTTP connection.

HTTP is a request-response protocol in which the parameters of request must be set before the request is sent. The connection exists in one of three states:

- Setup, in which the request parameters can be set
- Connected, in which request parameters have been sent and the response is expected
- Closed, the final state, in which the HTTP connection as been terminated

The following methods may be invoked only in the Setup state:

- `setRequestMethod`
- `setRequestProperty`

The transition from Setup to Connected is caused by any method that requires data to be sent to or received from the server.

The following methods cause the transition to the Connected state when the connection is in Setup state.

- `openInputStream`
- `openDataInputStream`
- `getLength`
- `getType`
- `getEncoding`
- `getHeaderField`
- `getResponseCode`
- `getResponseMessage`
- `getHeaderFieldInt`
- `getHeaderFieldDate`
- `getExpiration`
- `getDate`
- `getLastModified`
- `getHeaderField`

- `getHeaderFieldKey`

The following methods may be invoked while the connection is in Setup or Connected state.

- `close`
- `getRequestMethod`
- `getRequestProperty`
- `getURL`
- `getProtocol`
- `getHost`
- `getFile`
- `getRef`
- `getPort`
- `getQuery`

After an output stream has been opened by the `openOutputStream` or `openDataOutputStream` methods, attempts to change the request parameters via `setRequestMethod` or the `setRequestProperty` are ignored. Once the request parameters have been sent, these methods will throw an `IOException`. When an output stream is closed via the `OutputStream.close` or `DataOutputStream.close` methods, the connection enters the Connected state. When the output stream is flushed via the `OutputStream.flush` or `DataOutputStream.flush` methods, the request parameters MUST be sent along with any data written to the stream.

The transition to Closed state from any other state is caused by the `close` method and the closing all of the streams that were opened from the connection.

Example using StreamConnection

Simple read of a URL using `StreamConnection`. No HTTP specific behavior is needed or used. (**Note:** this example ignores all HTTP response headers and the HTTP response code. Since a proxy or server may have sent an error response page, an application can not distinquish which data is retreived in the `InputStream`.)

`Connector.open` is used to open URL and a `StreamConnection` is returned. From the `StreamConnection` the `InputStream` is opened. It is used to read every character until end of file (-1). If an exception is thrown the connection and stream are closed.

```
        void getViaStreamConnection(String url) throws IOException {
            StreamConnection c = null;
            InputStream s = null;
            try {
                c = (StreamConnection)Connector.open(url);
                s = c.openInputStream();
                int ch;
                while ((ch = s.read()) != -1) {
                    ...
                }
            } finally {
                if (s != null)
                    s.close();
                if (c != null)
                    c.close();
            }
        }
```

Example using ContentConnection

Simple read of a URL using `ContentConnection`. No HTTP specific behavior is needed or used.

`Connector.open` is used to open url and a `ContentConnection` is returned. The `ContentConnection` may be able to provide the length. If the length is available, it is used to read the data in bulk. From the `ContentConnection` the `InputStream` is opened. It is used to read every character until end of file (-1). If an exception is thrown the connection and stream are closed.

```
void getViaContentConnection(String url) throws IOException {
    ContentConnection c = null;
    DataInputStream is = null;
    try {
        c = (ContentConnection)Connector.open(url);
        int len = (int)c.getLength();
        dis = c.openDataInputStream();
        if (len > 0) {
            byte[] data = new byte[len];
            dis.readFully(data);
        } else {
            int ch;
            while ((ch = dis.read()) != -1) {
                ...
            }
        }
    } finally {
        if (dis != null)
            dis.close();
        if (c != null)
            c.close();
    }
}
```

Example using HttpConnection

Read the HTTP headers and the data using `HttpConnection`.

`Connector.open` is used to open url and a `HttpConnection` is returned. The HTTP headers are read and processed. If the length is available, it is used to read the data in bulk. From the `HttpConnection` the `InputStream` is opened. It is used to read every character until end of file (-1). If an exception is thrown the connection and stream are closed.

```
void getViaHttpConnection(String url) throws IOException {
    HttpConnection c = null;
    InputStream is = null;
    int rc;
    try {
        c = (HttpConnection)Connector.open(url);
        // Getting the response code will open the connection,
        // send the request, and read the HTTP response headers.
        // The headers are stored until requested.
        rc = c.getResponseCode();
        if (rc != HttpConnection.HTTP_OK) {
            throw new IOException("HTTP response code: " + rc);
        }
        is = c.openInputStream();
        // Get the ContentType
        String type = c.getType();
        // Get the length and process the data
        int len = (int)c.getLength();
        if (len > 0) {
            int actual = 0;
            int bytesread = 0 ;
            byte[] data = new byte[len];
            while ((bytesread != len) && (actual != -1)) {
               actual = is.read(data, bytesread, len - bytesread);
               bytesread += actual;
            }
        } else {
            int ch;
            while ((ch = is.read()) != -1) {
                ...
            }
        }
    } catch (ClassCastException e) {
        throw new IllegalArgumentException("Not an HTTP URL");
    } finally {
        if (is != null)
            is.close();
        if (c != null)
            c.close();
    }
}
```

Example using POST with HttpConnection

Post a request with some headers and content to the server and process the headers and content.

`Connector.open` is used to open url and a `HttpConnection` is returned. The request method is set to POST and request headers set. A simple command is written and flushed. The HTTP headers are read and processed. If the length is available, it is used to read the data in bulk. From the `HttpConnection` the `InputStream` is opened. It is used to read every character until end of file (-1). If an exception is thrown the connection and stream is closed.

```
void postViaHttpConnection(String url) throws IOException {
    HttpConnection c = null;
    InputStream is = null;
    OutputStream os = null;
    int rc;
    try {
        c = (HttpConnection)Connector.open(url);
        // Set the request method and headers
        c.setRequestMethod(HttpConnection.POST);
        c.setRequestProperty(“If-Modified-Since”,
            “29 Oct 1999 19:43:31 GMT”);
        c.setRequestProperty(“User-Agent”,
            “Profile/MIDP-2.0 Configuration/CLDC-1.0”);
        c.setRequestProperty(“Content-Language”, “en-US”);
        // Getting the output stream may flush the headers
        os = c.openOutputStream();
        os.write(“LIST games\n”.getBytes());
        os.flush();           // Optional, getResponseCode will flush
        // Getting the response code will open the connection,
        // send the request, and read the HTTP response headers.
        // The headers are stored until requested.
        rc = c.getResponseCode();
        if (rc != HttpConnection.HTTP_OK) {
            throw new IOException(“HTTP response code: ” + rc);
        }
        is = c.openInputStream();
        // Get the ContentType
        String type = c.getType();
        processType(type);
        // Get the length and process the data
        int len = (int)c.getLength();
        if (len > 0) {
             int actual = 0;
             int bytesread = 0 ;
             byte[] data = new byte[len];
             while ((bytesread != len) && (actual != -1)) {
                actual = is.read(data, bytesread, len - bytesread);
                bytesread += actual;
             }
            process(data);
        } else {
            int ch;
            while ((ch = is.read()) != -1) {
                process((byte)ch);
            }
        }
    } catch (ClassCastException e) {
        throw new IllegalArgumentException(“Not an HTTP URL”);
    } finally {
        if (is != null)
            is.close();
        if (os != null)
            os.close();
        if (c != null)
            c.close();
    }
}
```

Simplified Stream Methods on Connector

Please note the following: The `Connector` class defines the following convenience methods for retrieving an input or output stream directly for a specified URL:

- `InputStream openInputStream(String url)`
- `DataInputStream openDataInputStream(String url)`
- `OutputStream openOutputStream(String url)`

- `DataOutputStream openDataOutputStream(String url)`

Please be aware that using these methods implies certain restrictions. You will not get a reference to the actual connection, but rather just references to the input or output stream of the connection. Not having a reference to the connection means that you will not be able to manipulate or query the connection directly. This in turn means that you will not be able to call any of the following methods:

- `getRequestMethod()`
- `setRequestMethod()`
- `getRequestProperty()`
- `setRequestProperty()`
- `getLength()`
- `getType()`
- `getEncoding()`
- `getHeaderField()`
- `getResponseCode()`
- `getResponseMessage()`
- `getHeaderFieldInt`
- `getHeaderFieldDate`
- `getExpiration`
- `getDate`
- `getLastModified`
- `getHeaderField`
- `getHeaderFieldKey`

Since: MIDP 1.0

Member Summary

Fields

static	GET[72]
java.lang.String	
static	HEAD[72]
java.lang.String	
static int	HTTP_ACCEPTED[72]
static int	HTTP_BAD_GATEWAY[72]
static int	HTTP_BAD_METHOD[73]
static int	HTTP_BAD_REQUEST[73]
static int	HTTP_CLIENT_TIMEOUT[73]
static int	HTTP_CONFLICT[73]
static int	HTTP_CREATED[73]
static int	HTTP_ENTITY_TOO_LARGE[73]
static int	HTTP_EXPECT_FAILED[73]
static int	HTTP_FORBIDDEN[74]

Member Summary

static int	HTTP_GATEWAY_TIMEOUT[74]
static int	HTTP_GONE[74]
static int	HTTP_INTERNAL_ERROR[74]
static int	HTTP_LENGTH_REQUIRED[74]
static int	HTTP_MOVED_PERM[74]
static int	HTTP_MOVED_TEMP[75]
static int	HTTP_MULT_CHOICE[75]
static int	HTTP_NO_CONTENT[75]
static int	HTTP_NOT_ACCEPTABLE[75]
static int	HTTP_NOT_AUTHORITATIVE[75]
static int	HTTP_NOT_FOUND[75]
static int	HTTP_NOT_IMPLEMENTED[76]
static int	HTTP_NOT_MODIFIED[76]
static int	HTTP_OK[76]
static int	HTTP_PARTIAL[76]
static int	HTTP_PAYMENT_REQUIRED[76]
static int	HTTP_PRECON_FAILED[76]
static int	HTTP_PROXY_AUTH[76]
static int	HTTP_REQ_TOO_LONG[77]
static int	HTTP_RESET[77]
static int	HTTP_SEE_OTHER[77]
static int	HTTP_TEMP_REDIRECT[77]
static int	HTTP_UNAUTHORIZED[77]
static int	HTTP_UNAVAILABLE[77]
static int	HTTP_UNSUPPORTED_RANGE[77]
static int	HTTP_UNSUPPORTED_TYPE[78]
static int	HTTP_USE_PROXY[78]
static int	HTTP_VERSION[78]
static	POST[78]
java.lang.String	

Methods

long	getDate()[78]
long	getExpiration()[79]
java.lang.String	getFile()[79]
java.lang.String	getHeaderField(int n)[79]
java.lang.String	getHeaderField(String name)[79]
long	getHeaderFieldDate(String name, long def)[80]
int	getHeaderFieldInt(String name, int def)[80]
java.lang.String	getHeaderFieldKey(int n)[80]
java.lang.String	getHost()[81]
long	getLastModified()[81]
int	getPort()[81]
java.lang.String	getProtocol()[81]
java.lang.String	getQuery()[81]
java.lang.String	getRef()[82]
java.lang.String	getRequestMethod()[82]
java.lang.String	getRequestProperty(String key)[82]
int	getResponseCode()[82]
java.lang.String	getResponseMessage()[83]
java.lang.String	getURL()[83]
void	setRequestMethod(String method)[83]
void	setRequestProperty(String key, String value)[84]

Inherited Member Summary

Methods inherited from interface `Connection`
`close()`
Methods inherited from interface `ContentConnection`
`getEncoding(), getLength(), getType()`
Methods inherited from interface `InputConnection`
`openDataInputStream(), openInputStream()`
Methods inherited from interface `OutputConnection`
`openDataOutputStream(), openOutputStream()`

Fields

GET

Declaration:
`public static final String GET`

Description:
HTTP Get method.

HEAD

Declaration:
`public static final String HEAD`

Description:
HTTP Head method.

HTTP_ACCEPTED

Declaration:
`public static final int HTTP_ACCEPTED`

Description:
202: The request has been accepted for processing, but the processing has not been completed.

HTTP_BAD_GATEWAY

Declaration:
`public static final int HTTP_BAD_GATEWAY`

Description:
502: The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request.

HTTP_BAD_METHOD

Declaration:
`public static final int HTTP_BAD_METHOD`

Description:
405: The method specified in the Request-Line is not allowed for the resource identified by the Request-URI.

HTTP_BAD_REQUEST

Declaration:
`public static final int HTTP_BAD_REQUEST`

Description:
400: The request could not be understood by the server due to malformed syntax.

HTTP_CLIENT_TIMEOUT

Declaration:
`public static final int HTTP_CLIENT_TIMEOUT`

Description:
408: The client did not produce a request within the time that the server was prepared to wait. The client MAY repeat the request without modifications at any later time.

HTTP_CONFLICT

Declaration:
`public static final int HTTP_CONFLICT`

Description:
409: The request could not be completed due to a conflict with the current state of the resource.

HTTP_CREATED

Declaration:
`public static final int HTTP_CREATED`

Description:
201: The request has been fulfilled and resulted in a new resource being created.

HTTP_ENTITY_TOO_LARGE

Declaration:
`public static final int HTTP_ENTITY_TOO_LARGE`

Description:
413: The server is refusing to process a request because the request entity is larger than the server is willing or able to process.

HTTP_EXPECT_FAILED

Declaration:
`public static final int HTTP_EXPECT_FAILED`

HTTP_FORBIDDEN

Description:
417: The expectation given in an Expect request-header field could not be met by this server, or, if the server is a proxy, the server has unambiguous evidence that the request could not be met by the next-hop server.

HTTP_FORBIDDEN

Declaration:
```
public static final int HTTP_FORBIDDEN
```

Description:
403: The server understood the request, but is refusing to fulfill it. Authorization will not help and the request SHOULD NOT be repeated.

HTTP_GATEWAY_TIMEOUT

Declaration:
```
public static final int HTTP_GATEWAY_TIMEOUT
```

Description:
504: The server, while acting as a gateway or proxy, did not receive a timely response from the upstream server specified by the URI or some other auxiliary server it needed to access in attempting to complete the request.

HTTP_GONE

Declaration:
```
public static final int HTTP_GONE
```

Description:
410: The requested resource is no longer available at the server and no forwarding address is known.

HTTP_INTERNAL_ERROR

Declaration:
```
public static final int HTTP_INTERNAL_ERROR
```

Description:
500: The server encountered an unexpected condition which prevented it from fulfilling the request.

HTTP_LENGTH_REQUIRED

Declaration:
```
public static final int HTTP_LENGTH_REQUIRED
```

Description:
411: The server refuses to accept the request without a defined Content- Length.

HTTP_MOVED_PERM

Declaration:
```
public static final int HTTP_MOVED_PERM
```

Description:
301: The requested resource has been assigned a new permanent URI and any future references to this resource SHOULD use one of the returned URIs.

HTTP_MOVED_TEMP

Declaration:
```
public static final int HTTP_MOVED_TEMP
```

Description:
302: The requested resource resides temporarily under a different URI. (**Note:** the name of this status code reflects the earlier publication of RFC2068, which was changed in RFC2616 from “moved temporalily” to “found”. The semantics were not changed. The `Location` header indicates where the application should resend the request.)

HTTP_MULT_CHOICE

Declaration:
```
public static final int HTTP_MULT_CHOICE
```

Description:
300: The requested resource corresponds to any one of a set of representations, each with its own specific location, and agent- driven negotiation information is being provided so that the user (or user agent) can select a preferred representation and redirect its request to that location.

HTTP_NO_CONTENT

Declaration:
```
public static final int HTTP_NO_CONTENT
```

Description:
204: The server has fulfilled the request but does not need to return an entity-body, and might want to return updated meta-information.

HTTP_NOT_ACCEPTABLE

Declaration:
```
public static final int HTTP_NOT_ACCEPTABLE
```

Description:
406: The resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headers sent in the request.

HTTP_NOT_AUTHORITATIVE

Declaration:
```
public static final int HTTP_NOT_AUTHORITATIVE
```

Description:
203: The returned meta-information in the entity-header is not the definitive set as available from the origin server.

HTTP_NOT_FOUND

Declaration:
```
public static final int HTTP_NOT_FOUND
```

Description:
404: The server has not found anything matching the Request-URI. No indication is given of whether the condition is temporary or permanent.

HTTP_NOT_IMPLEMENTED

Declaration:
`public static final int HTTP_NOT_IMPLEMENTED`

Description:
501: The server does not support the functionality required to fulfill the request.

HTTP_NOT_MODIFIED

Declaration:
`public static final int HTTP_NOT_MODIFIED`

Description:
304: If the client has performed a conditional GET request and access is allowed, but the document has not been modified, the server SHOULD respond with this status code.

HTTP_OK

Declaration:
`public static final int HTTP_OK`

Description:
200: The request has succeeded.

HTTP_PARTIAL

Declaration:
`public static final int HTTP_PARTIAL`

Description:
206: The server has fulfilled the partial GET request for the resource.

HTTP_PAYMENT_REQUIRED

Declaration:
`public static final int HTTP_PAYMENT_REQUIRED`

Description:
402: This code is reserved for future use.

HTTP_PRECON_FAILED

Declaration:
`public static final int HTTP_PRECON_FAILED`

Description:
412: The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server.

HTTP_PROXY_AUTH

Declaration:
`public static final int HTTP_PROXY_AUTH`

Description:
407: This code is similar to 401 (Unauthorized), but indicates that the client must first authenticate itself with the proxy.

HTTP_REQ_TOO_LONG

Declaration:
`public static final int HTTP_REQ_TOO_LONG`

Description:
414: The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret.

HTTP_RESET

Declaration:
`public static final int HTTP_RESET`

Description:
205: The server has fulfilled the request and the user agent SHOULD reset the document view which caused the request to be sent.

HTTP_SEE_OTHER

Declaration:
`public static final int HTTP_SEE_OTHER`

Description:
303: The response to the request can be found under a different URI and SHOULD be retrieved using a GET method on that resource.

HTTP_TEMP_REDIRECT

Declaration:
`public static final int HTTP_TEMP_REDIRECT`

Description:
307: The requested resource resides temporarily under a different URI.

HTTP_UNAUTHORIZED

Declaration:
`public static final int HTTP_UNAUTHORIZED`

Description:
401: The request requires user authentication. The response MUST include a WWW-Authenticate header field containing a challenge applicable to the requested resource.

HTTP_UNAVAILABLE

Declaration:
`public static final int HTTP_UNAVAILABLE`

Description:
503: The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.

HTTP_UNSUPPORTED_RANGE

Declaration:
`public static final int HTTP_UNSUPPORTED_RANGE`

HTTP_UNSUPPORTED_TYPE

Description:
416: A server SHOULD return a response with this status code if a request included a Range request-header field , and none of the range-specifier values in this field overlap the current extent of the selected resource, and the request did not include an If-Range request-header field.

HTTP_UNSUPPORTED_TYPE

Declaration:
```
public static final int HTTP_UNSUPPORTED_TYPE
```

Description:
415: The server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.

HTTP_USE_PROXY

Declaration:
```
public static final int HTTP_USE_PROXY
```

Description:
305: The requested resource MUST be accessed through the proxy given by the Location field.

HTTP_VERSION

Declaration:
```
public static final int HTTP_VERSION
```

Description:
505: The server does not support, or refuses to support, the HTTP protocol version that was used in the request message.

POST

Declaration:
```
public static final String POST
```

Description:
HTTP Post method.

Methods

getDate()

Declaration:
```
public long getDate()
            throws IOException
```

Description:
Returns the value of the `date` header field.

Returns: the sending date of the resource that the URL references, or `0` if not known. The value returned is the number of milliseconds since January 1, 1970 GMT.

Throws:
`java.io.IOException` - if an error occurred connecting to the server.

getExpiration()

Declaration:
```
public long getExpiration()
            throws IOException
```

Description:
Returns the value of the `expires` header field.

Returns: the expiration date of the resource that this URL references, or 0 if not known. The value is the number of milliseconds since January 1, 1970 GMT.

Throws:
`java.io.IOException` - if an error occurred connecting to the server.

getFile()

Declaration:
```
public String getFile()
```

Description:
Returns the file portion of the URL of this `HttpConnection`.

Returns: the file portion of the URL of this `HttpConnection`. `null` is returned if there is no file.

getHeaderField(int)

Declaration:
```
public String getHeaderField(int n)
            throws IOException
```

Description:
Gets a header field value by index.

Parameters:
`n` - the index of the header field

Returns: the value of the nth header field or `null` if the array index is out of range. An empty String is returned if the field does not have a value.

Throws:
`java.io.IOException` - if an error occurred connecting to the server.

getHeaderField(String)

Declaration:
```
public String getHeaderField(String name)
            throws IOException
```

Description:
Returns the value of the named header field.

Parameters:
`name` - of a header field.

Returns: the value of the named header field, or `null` if there is no such field in the header.

Throws:
`java.io.IOException` - if an error occurred connecting to the server.

getHeaderFieldDate(String, long)

Declaration:

```
public long getHeaderFieldDate(String name, long def)
            throws IOException
```

Description:

Returns the value of the named field parsed as date. The result is the number of milliseconds since January 1, 1970 GMT represented by the named field.

This form of `getHeaderField` exists because some connection types (e.g., `http-ng`) have pre-parsed headers. Classes for that connection type can override this method and short-circuit the parsing.

Parameters:

`name` - the name of the header field.

`def` - a default value.

Returns: the value of the field, parsed as a date. The value of the `def` argument is returned if the field is missing or malformed.

Throws:

`java.io.IOException` - if an error occurred connecting to the server.

getHeaderFieldInt(String, int)

Declaration:

```
public int getHeaderFieldInt(String name, int def)
            throws IOException
```

Description:

Returns the value of the named field parsed as a number.

This form of `getHeaderField` exists because some connection types (e.g., `http-ng`) have pre-parsed headers. Classes for that connection type can override this method and short-circuit the parsing.

Parameters:

`name` - the name of the header field.

`def` - the default value.

Returns: the value of the named field, parsed as an integer. The `def` value is returned if the field is missing or malformed.

Throws:

`java.io.IOException` - if an error occurred connecting to the server.

getHeaderFieldKey(int)

Declaration:

```
public String getHeaderFieldKey(int n)
            throws IOException
```

Description:

Gets a header field key by index.

Parameters:

`n` - the index of the header field

Returns: the key of the nth header field or `null` if the array index is out of range.

Throws:
`java.io.IOException` - if an error occurred connecting to the server.

getHost()

Declaration:
```
public String getHost()
```

Description:
Returns the host information of the URL of this `HttpConnection`. e.g. host name or IPv4 address

Returns: the host information of the URL of this `HttpConnection`.

getLastModified()

Declaration:
```
public long getLastModified()
            throws IOException
```

Description:
Returns the value of the `last-modified` header field. The result is the number of milliseconds since January 1, 1970 GMT.

Returns: the date the resource referenced by this `HttpConnection` was last modified, or 0 if not known.

Throws:
`java.io.IOException` - if an error occurred connecting to the server.

getPort()

Declaration:
```
public int getPort()
```

Description:
Returns the network port number of the URL for this `HttpConnection`.

Returns: the network port number of the URL for this `HttpConnection`. The default HTTP port number (80) is returned if there was no port number in the string passed to `Connector.open`.

getProtocol()

Declaration:
```
public String getProtocol()
```

Description:
Returns the protocol name of the URL of this `HttpConnection`. e.g., http or https

Returns: the protocol of the URL of this `HttpConnection`.

getQuery()

Declaration:
```
public String getQuery()
```

Description:
Returns the query portion of the URL of this `HttpConnection`. RFC2396 defines the query component as the text after the first question-mark (?) character in the URL.

Returns: the query portion of the URL of this `HttpConnection`. `null` is returned if there is no value.

getRef()

Declaration:
```
public String getRef()
```

Description:
Returns the ref portion of the URL of this `HttpConnection`. RFC2396 specifies the optional fragment identifier as the the text after the crosshatch (#) character in the URL. This information may be used by the user agent as additional reference information after the resource is successfully retrieved. The format and interpretation of the fragment identifier is dependent on the media type[RFC2046] of the retrieved information.

Returns: the ref portion of the URL of this `HttpConnection`. `null` is returned if there is no value.

getRequestMethod()

Declaration:
```
public String getRequestMethod()
```

Description:
Get the current request method. e.g. HEAD, GET, POST The default value is GET.

Returns: the HTTP request method

See Also: `setRequestMethod(String)` *83*

getRequestProperty(String)

Declaration:
```
public String getRequestProperty(String key)
```

Description:
Returns the value of the named general request property for this connection.

Parameters:
`key` - the keyword by which the request property is known (e.g., “accept”).

Returns: the value of the named general request property for this connection. If there is no key with the specified name then `null` is returned.

See Also: `setRequestProperty(String, String)` *84*

getResponseCode()

Declaration:
```
public int getResponseCode()
            throws IOException
```

Description:
Returns the HTTP response status code. It parses responses like:

```
HTTP/1.0 200 OK
HTTP/1.0 401 Unauthorized
```

and extracts the ints 200 and 401 respectively. from the response (i.e., the response is not valid HTTP).

Returns: the HTTP Status-Code or -1 if no status code can be discerned.

Throws:
`java.io.IOException` - if an error occurred connecting to the server.

getResponseMessage()

Declaration:
```
public String getResponseMessage()
            throws IOException
```

Description:
Gets the HTTP response message, if any, returned along with the response code from a server. From responses like:

```
HTTP/1.0 200 OK
HTTP/1.0 404 Not Found
```

Extracts the Strings “OK” and “Not Found” respectively. Returns null if none could be discerned from the responses (the result was not valid HTTP).

Returns: the HTTP response message, or `null`

Throws:
`java.io.IOException` - if an error occurred connecting to the server.

getURL()

Declaration:
```
public String getURL()
```

Description:
Return a string representation of the URL for this connection.

Returns: the string representation of the URL for this connection.

setRequestMethod(String)

Declaration:
```
public void setRequestMethod(String method)
            throws IOException
```

Description:
Set the method for the URL request, one of:

- GET
- POST
- HEAD

are legal, subject to protocol restrictions. The default method is GET.

Parameters:
`method` - the HTTP method

Throws:
`java.io.IOException` - if the method cannot be reset or if the requested method isn’t valid for HTTP.

See Also: `getRequestMethod()` 82

setRequestProperty(String, String)

Declaration:

```
public void setRequestProperty(String key, String value)
            throws IOException
```

Description:

Sets the general request property. If a property with the key already exists, overwrite its value with the new value.

Note: HTTP requires all request properties which can legally have multiple instances with the same key to use a comma-separated list syntax which enables multiple properties to be appended into a single property.

Parameters:

`key` - the keyword by which the request is known (e.g., "`accept`").

`value` - the value associated with it.

Throws:

`java.io.IOException` - is thrown if the connection is in the connected state.

See Also: `getRequestProperty(String)` *82*

javax.microedition.io
HttpsConnection

Declaration

```
public interface HttpsConnection extends HttpConnection65
```

All Superinterfaces: `Connection`, `ContentConnection`, `HttpConnection`[65], `InputConnection`, `OutputConnection`, `StreamConnection`

Description

This interface defines the necessary methods and constants to establish a secure network connection. The URI format with scheme `https` when passed to `Connector.open` will return a `HttpsConnection`. RFC 2818 (http://www.ietf.org/rfc/rfc2818.txt) defines the scheme.

A secure connection MUST be implemented by one or more of the following specifications:

- HTTP over TLS as documented in RFC 2818 (http://www.ietf.org/rfc/rfc2818.txt) and TLS Protocol Version 1.0 as specified in RFC 2246 (http://www.ietf.org/rfc/rfc2246.txt).
- SSL V3 as specified in The SSL Protocol Version 3.0 (http://home.netscape.com/eng/ssl3/draft302.txt)
- WTLS as specified in WAP Forum Specifications June 2000 (WAP 1.2.1) conformance release (http://www.wapforum.org/what/technical_1_2_1.htm) - Wireless Transport Layer Security document WAP-199.
- WAP(TM) TLS Profile and Tunneling Specification as specified in WAP-219-TLS-20010411-a (http://www.wapforum.com/what/technical.htm)

HTTPS is the secure version of HTTP (IETF RFC2616), a request-response protocol in which the parameters of the request must be set before the request is sent.

In addition to the normal `IOExceptions` that may occur during invocation of the various methods that cause a transition to the Connected state, `CertificateException` (a subtype of `IOException`) may be thrown to indicate various failures related to establishing the secure link. The secure link is necessary in the `Connected` state so the headers can be sent securely. The secure link may be established as early as the invocation of `Connector.open()` and related methods for opening input and output streams and failure related to certificate exceptions may be reported.

Example

Open a HTTPS connection, set its parameters, then read the HTTP response.

`Connector.open` is used to open the URL and an `HttpsConnection` is returned. The HTTP headers are read and processed. If the length is available, it is used to read the data in bulk. From the `HttpsConnection` the `InputStream` is opened. It is used to read every character until end of file (-1). If an exception is thrown the connection and stream are closed.

setRequestProperty(String, String)

```
void getViaHttpsConnection(String url)
       throws CertificateException, IOException {
    HttpsConnection c = null;
    InputStream is = null;
    try {
        c = (HttpsConnection)Connector.open(url);
        // Getting the InputStream ensures that the connection
        // is opened (if it was not already handled by
        // Connector.open()) and the SSL handshake is exchanged,
        // and the HTTP response headers are read.
        // These are stored until requested.
        is = c.openDataInputStream();
        if c.getResponseCode() == HttpConnection.HTTP_OK) {
            // Get the length and process the data
            int len = (int)c.getLength();
            if (len > 0) {
                byte[] data = new byte[len];
                int actual = is.readFully(data);
                ...
            } else {
                int ch;
                while ((ch = is.read()) != -1) {
                    ...
                }
            }
        } else {
          ...
        }
    } finally {
        if (is != null)
            is.close();
        if (c != null)
            c.close();
    }
}
```

Since: MIDP 2.0

See Also: javax.microedition.pki.CertificateException 458

Member Summary

Methods

int	getPort() 87
SecurityInfo	getSecurityInfo() 88

Inherited Member Summary

Fields inherited from interface HttpConnection 65

Inherited Member Summary

GET[72], HEAD[72], HTTP_ACCEPTED[72], HTTP_BAD_GATEWAY[72], HTTP_BAD_METHOD[73], HTTP_BAD_REQUEST[73], HTTP_CLIENT_TIMEOUT[73], HTTP_CONFLICT[73], HTTP_CREATED[73], HTTP_ENTITY_TOO_LARGE[73], HTTP_EXPECT_FAILED[73], HTTP_FORBIDDEN[74], HTTP_GATEWAY_TIMEOUT[74], HTTP_GONE[74], HTTP_INTERNAL_ERROR[74], HTTP_LENGTH_REQUIRED[74], HTTP_MOVED_PERM[74], HTTP_MOVED_TEMP[75], HTTP_MULT_CHOICE[75], HTTP_NOT_ACCEPTABLE[75], HTTP_NOT_AUTHORITATIVE[75], HTTP_NOT_FOUND[75], HTTP_NOT_IMPLEMENTED[76], HTTP_NOT_MODIFIED[76], HTTP_NO_CONTENT[75], HTTP_OK[76], HTTP_PARTIAL[76], HTTP_PAYMENT_REQUIRED[76], HTTP_PRECON_FAILED[76], HTTP_PROXY_AUTH[76], HTTP_REQ_TOO_LONG[77], HTTP_RESET[77], HTTP_SEE_OTHER[77], HTTP_TEMP_REDIRECT[77], HTTP_UNAUTHORIZED[77], HTTP_UNAVAILABLE[77], HTTP_UNSUPPORTED_RANGE[77], HTTP_UNSUPPORTED_TYPE[78], HTTP_USE_PROXY[78], HTTP_VERSION[78], POST[78]

Methods inherited from interface `Connection`

`close()`

Methods inherited from interface `ContentConnection`

`getEncoding()`, `getLength()`, `getType()`

Methods inherited from interface `HttpConnection`[65]

getDate()[78], getExpiration()[79], getFile()[79], getHeaderField(int)[79], getHeaderField(int)[79], getHeaderFieldDate(String, long)[80], getHeaderFieldInt(String, int)[80], getHeaderFieldKey(int)[80], getHost()[81], getLastModified()[81], getProtocol()[81], getQuery()[81], getRef()[82], getRequestMethod()[82], getRequestProperty(String)[82], getResponseCode()[82], getResponseMessage()[83], getURL()[83], setRequestMethod(String)[83], setRequestProperty(String, String)[84]

Methods inherited from interface `InputConnection`

`openDataInputStream()`, `openInputStream()`

Methods inherited from interface `OutputConnection`

`openDataOutputStream()`, `openOutputStream()`

Methods

getPort()

Declaration:
`public int getPort()`

Description:
Returns the network port number of the URL for this `HttpsConnection`.

Overrides: `getPort`[81] in interface `HttpConnection`[65]

Returns: the network port number of the URL for this `HttpsConnection`. The default HTTPS port number (443) is returned if there was no port number in the string passed to `Connector.open`.

getSecurityInfo()

Declaration:

```
public javax.microedition.io.SecurityInfo[103] getSecurityInfo()
            throws IOException
```

Description:

Return the security information associated with this successfully opened connection. If the connection is still in `Setup` state then the connection is initiated to establish the secure connection to the server. The method returns when the connection is established and the `Certificate` supplied by the server has been validated. The `SecurityInfo` is only returned if the connection has been successfully made to the server.

Returns: the security information associated with this open connection.

Throws:

`java.io.IOException` - if an arbitrary connection failure occurs

javax.microedition.io
PushRegistry

Declaration

```
public class PushRegistry

Object
  |
  +--javax.microedition.io.PushRegistry
```

Description

The `PushRegistry` maintains a list of inbound connections. An application can register the inbound connections with an entry in the application descriptor file or dynamically by calling the `registerConnection` method.

While an application is running, it is responsible for all I/O operations associated with the inbound connection. When the application is not running, the application management software(AMS) listens for inbound notification requests. When a notification arrives for a registered `MIDlet`, the AMS will start the `MIDlet` via the normal invocation of `MIDlet.startApp` method.

Installation Handling of Declared Connections

To avoid collisions on inbound generic connections, the application descriptor file MUST include information about static connections that are needed by the `MIDlet` suite. If all the static Push declarations in the application descriptor can not be fulfilled during the installation, the user MUST be notified that there are conflicts and the MIDlet suite MUST NOT be installed. (See *Over The Air User Initiated Provisioning Specification* section for errors reported in the event of conflicts.) Conditions when the declarations can not be fulfilled include: syntax errors in the Push attributes, declaration for a connection end point (e.g. port number) that is already reserved in the device, declaration for a protocol that is not supported for Push in the device, and declaration referencing a `MIDlet` class that is not listed in the `MIDlet-<n>` attributes of the same application descriptor. If the `MIDlet` suite can function meaningfully even if a Push registration can't be fulfilled, it MUST register the Push connections using the dynamic registration methods in the `PushRegistry`.

A conflict-free installation reserves each requested connection for the exclusive use of the `MIDlets` in the suite. While the suite is installed, any attempt by other applications to open one of the reserved connections will fail with an `IOException`. A call from a `MIDlet` to `Connector.open()` on a connection reserved for its suite will always succeed, assuming the suite does not already have the connection open.

If two `MIDlet` suites have a static push connection in common, they cannot be installed together and both function correctly. The end user would typically have to uninstall one before being able to successfully install the other.

Push Registration Attribute

Each push registration entry contains the following information :

`MIDlet-Push-``<n>: <ConnectionURL>, <MIDletClassName>, <AllowedSender>`

where :

- `MIDlet-Push-<n>` = the Push registration attribute name. Multiple push registrations can be provided in a `MIDlet` suite. The numeric value for <n> starts from 1 and MUST use consecutive ordinal numbers for additional entries. The first missing entry terminates the list. Any additional entries are ignored.
- `ConnectionURL` = the connection string used in `Connector.open()`

- `MIDletClassName` = the `MIDlet` that is responsible for the connection. The named `MIDlet` MUST be registered in the descriptor file or the jar file manifest with a `MIDlet-<n>` record. (This information is needed when displaying messages to the user about the application when push connections are detected, or when the user grants/revokes priveleges for the application.) If the named `MIDlet` appears more than once in the suite, the first matching entry is used.
- `AllowedSender` = a designated filter that restricts which senders are valid for launching the requested `MIDlet`. The syntax and semantics of the `AllowedSender` field depend on the addressing format used for the protocol. However, every syntax for this field MUST support using the wildcard characters "*" and "?". The semantics of those wildcard are:
  - "*" matches any string, including an empty string
  - "?" matches any single character

When the value of this field is just the wildcard character "*", connections will be accepted from any originating source. For Push attributes using the `datagram` and `socket` URLs (if supported by the platform), this field contains a numeric IP address in the same format for IPv4 and IPv6 as used in the respective URLs (IPv6 address including the square brackets as in the URL). It is possible to use the wildcards also in these IP addresses, e.g. "129.70.40.*" would allow subnet resolution. Note that the port number is not part of the filter for `datagram` and `socket` connections.

The MIDP 2.0 specification defines the syntax for `datagram` and `socket` inbound connections. When other specifications define push semantics for additional connection types, they must define the expected syntax for the filter field, as well as the expected format for the connection URL string.

Example Descriptor File Declarative Notation

The following is a sample descriptor file entry that would reserve a stream socket at port 79 and a datagram connection at port 50000. (Port numbers are maintained by IANA and cover well-known, user-registered and dynamic port numbers) [See IANA Port Number Registry (http://www.iana.org/numbers.html#P)]

```
MIDlet-Push-1: socket://:79, com.sun.example.SampleChat, *
MIDlet-Push-2: datagram://:50000, com.sun.example.SampleChat, *
```

Buffered Messages

The requirements for buffering of messages are specific to each protocol used for Push and are defined separately for each protocol. There is no general requirement related to buffering that would apply to all protocols. If the implementation buffers messages, these messages MUST be provided to the `MIDlet` when the `MIDlet` is started and it opens the related `Connection` that it has registered for Push.

When datagram connections are supported with Push, the implementation MUST guarantee that when a `MIDlet` registered for datagram Push is started in response to an incoming datagram, at least the datagram that caused the startup of the `MIDlet` is buffered by the implementation and will be available to the `MIDlet` when the `MIDlet` opens the `UDPDatagramConnection` after startup.

When socket connections are supported with Push, the implementation MUST guarantee that when a `MIDlet` registered for socket Push is started in response to an incoming socket connection, this connection can be accepted by the `MIDlet` by opening the `ServerSocketConnection` after startup, provided that the connection hasn't timed out meanwhile.

Connection vs Push Registration Support

Not all generic connections will be appropriate for use as push application transport. Even if a protocol is supported on the device as an inbound connection type, it is not required to be enabled as a valid push mechanism. e.g. a platform might support server socket connections in a `MIDlet`, but might not support inbound socket connections for push launch capability. A `ConnectionNotFoundException` is thrown

from the `registerConnection` and from the `registerAlarm` methods, when the platform does not support that optional capability.

AMS Connection Handoff

Responsibility for registered push connections is shared between the AMS and the `MIDlet` that handles the I/O operations on the inbound connection. To prevent any data from being lost, an application is responsible for all I/O operations on the connection from the time it calls `Connector.open()` until it calls `Connection.close()`.

The AMS listens for inbound connection notifications. This MAY be handled via a native callback or polling mechanism looking for new inbound data. The AMS is responsible for enforcing the Security of PushRegistry and presenting notifications (if any) to the user before invoking the MIDlet suite.

The AMS is responsible for the shutdown of any running applications (if necessary) prior to the invocation of the push `MIDlet` method.

After the AMS has started the push application, the `MIDlet` is responsible for opening the connections and for all subsequent I/O operations. An application that needs to perform blocking I/O operations SHOULD use a separate thread to allow for interactive user operations. Once the application has been started and the connection has been opened, the AMS is no longer responsible for listening for push notifications for that connection. The application is responsible for reading all inbound data.

If an application has finished with all inbound data it MAY `close()` the connection. If the connection is closed, then neither the AMS nor the application will be listening for push notifications. Inbound data could be lost, if the application closes the connection before all data has been received.

When the application is destroyed, the AMS resumes its responsiblity to watch for inbound connections.

A push application SHOULD behave in a predictable manner when handling asynchronous data via the push mechanism. A well behaved application SHOULD inform the user that data has been processed. (While it is possible to write applications that do not use any user visible interfaces, this could lead to a confused end user experience to launch an application that only performs a background function.)

Dynamic Connections Registered from a Running MIDlet

There are cases when defining a well known port registered with IANA is not necessary. Simple applications may just wish to exchange data using a private protocol between a `MIDlet` and server application.

To accomodate this type of application, a mechanism is provided to dynamically allocate a connection and to register that information, as if it was known, when the application was installed. This information can then be sent to an agent on the network to use as the mechanism to communicate with the registered `MIDlet`.

For instance, if a `UDPDatagramConnection` is opened and a port number, was not specified, then the application is requesting a dynamic port to be allocated from the ports that are currently available. By calling `PushRegistry.registerConnection()` the `MIDlet` informs the AMS that it is the target for inbound communication, even after the `MIDlet` has been destroyed (See `MIDlet` life cycle for definition of “destroyed” state). If the application is deleted from the phone, then its dynamic communication connections are unregistered automatically.

AMS Runtime Handling - Implementation Notes

During installation each `MIDlet` that is expecting inbound communication on a well known address has the information recorded with the AMS from the push registration attribute in the manifest or application descriptor file. Once the installation has been successfully completed, (e.g. For the OTA recommended practices - when the *Installation notification message* has been successfully transmitted, the application is officially installed.) the `MIDlet` MAY then receive inbound communication. e.g. the push notification event.

When the AMS is started, it checks the list of registered connections and begins listening for inbound communication. When a notification arrives the AMS starts the registered `MIDlet`. The `MIDlet` then opens the connection with `Connector.open()` method to perform whatever I/O operations are needed for the particular connection type. e.g. for a server socket the application uses `acceptAndOpen()` to get the socket connected and for a datagram connection the application uses `receive()` to read the delivered message.

For message oriented transports the inbound message MAY be read by the AMS and saved for delivery to the `MIDlet` when it requests to read the data. For stream oriented transports the connection MAY be lost if the connection is not accepted before the server end of the connection request timeouts.

When a `MIDlet` is started in response to a registered push connection notification, it is platform dependent what happens to the current running application. The `MIDlet` life cycle defines the expected behaviors that an interrupted `MIDlet` could see from a call to `pauseApp()` or from `destroyApp()`.

Sample Usage Scenarios

Usage scenario 1: The suite includes a `MIDlet` with a well known port for communication. During the `startApp` processing a thread is launched to handle the incoming data. Using a separate thread is the recommended practice for avoiding conflicts between blocking I/O operations and the normal user interaction events. The thread continues to receive messages until the `MIDlet` is destroyed.

Sample Chat Descriptor File -

In this sample, the descriptor file includes a static push connection registration. It also includes an indication that this `MIDlet` requires permission to use a datagram connection for inbound push messages. (See Security of Push Functions in the package overview for details about `MIDlet` permissions.) **Note:** this sample is appropriate for bursts of datagrams. It is written to loop on the connection, processing received messages.

```
MIDlet-Name: SunNetwork - Chat Demo
MIDlet-Version: 1.0
MIDlet-Vendor: Sun Microsystems, Inc.
MIDlet-Description: Network demonstration programs for MIDP
MicroEdition-Profile: MIDP-2.0
MicroEdition-Configuration: CLDC-1.0
MIDlet-1: InstantMessage, /icons/Chat.png, example.chat.SampleChat, *
MIDlet-Push-1: datagram://:79,  example.chat.SampleChat, *
MIDlet-Permissions: javax.microedition.io.PushRegistry, \\
                    javax.microedition.io.Connector.datagramreceiver
```

Sample Chat MIDlet Processing -

```
public class SampleChat extends MIDlet {
    // Current inbound message connection.
    DatagramConnection conn;
    // Flag to terminate the message reading thread.
    boolean done_reading;
    public void startApp() {
        // List of active connections.
        String connections[];
        // Check to see if this session was started due to
        // inbound connection notification.
        connections = PushRegistry.listConnections(true);
        // Start an inbound message thread for available
        // inbound messages for the statically configured
        // connection in the descriptor file.
        for (int i=0; i < connections.length; i++) {
          Thread t = new Thread (new MessageHandler(
                           connections[i]));
          t.start();
        }
        ...
      }
    }
    // Stop reading inbound messages and release the push
    // connection to the AMS listener.
    public void destroyApp(boolean conditional) {
       done_reading = true;
       if (conn != null)
           conn.close();
       // Optionally, notify network service that we're
       // done with the current session.
       ...
    }
    // Optionally, notify network service.
    public void pauseApp() {
        ...
    }
 // Inner class to handle inbound messages on a separate thread.
 class MessageHandler implements Runnable {
     String connUrl ;
     MessageHandler(String url) {
          connUrl = url ;
     }
     // Fetch messages in a blocking receive loop.
     public void run() {
       try {
         // Get a connection handle for inbound messages
         // and a buffer to hold the inbound message.
         DatagramConnection conn = (DatagramConnection)
              Connector.open(connUrl);
         Datagram data = conn.newDatagram(conn.getMaximumLength());
         // Read the inbound messages
         while (!done_reading) {
            conn.receive(data);
           ...
```

```
            }
          } catch (IOException ioe) {
          ...
      }
      ...
```

Usage scenario 2: The suite includes a `MIDlet` that dynamically allocates port the first time it is started.

Sample Ping Descriptor File -
In this sample, the descriptor file includes an entry indicating that the application will need permission to use the datagram connection for inbound push messages. The dynamic connection is allocated in the constructor the first time it is run. The open connection is used during this session and can be reopened in a subsequent session in response to a inbound connection notification.

```
MIDlet-Name: SunNetwork - Demos
MIDlet-Version: 1.0
MIDlet-Vendor: Sun Microsystems, Inc.
MIDlet-Description: Network demonstration programs for MIDP
MicroEdition-Profile: MIDP-2.0
MicroEdition-Configuration: CLDC-1.0
MIDlet-1: JustCallMe, /icons/Ping.png, example.ping.SamplePingMe, *
MIDlet-Permissions: javax.microedition.io.PushRegistry, \\
                    javax.microedition.io.Connector.datagramreceiver
```

Sample Ping MIDlet Processing -

```
public class SamplePingMe extends MIDlet {
   // Name of the current application for push registration.
   String myName = “example.chat.SamplePingMe”;
   // List of registered push connections.
   String connections[];
   // Inbound datagram connection
   UDPDatagramConnection dconn;
   public SamplePingMe() {
       // Check to see if the ping connection has been registered.
       // This is a dynamic connection allocated on first
       // time execution of this MIDlet.
       connections = PushRegistry.listConnections(false);
       if (connections.length == 0) {
           // Request a dynamic port for out-of-band notices.
           // (Omitting the port number let's the system allocate
           //  an available port number.)
           try {
               dconn = (UDPDatagramConnection)
                       Connector.open(“datagram://”);
               String dport = “datagram://:”  + dconn.getLocalPort();
               // Register the port so the MIDlet will wake up, if messages
               // are posted after the MIDlet exits.
               PushRegistry.registerConnection(dport, myName, “*”);
               // Post my datagram address to the network
                 ...
           } catch (IOException ioe) {
               ...
           } catch (ClassNotFoundException cnfe) {
               ...
           }
   }
   public void startApp() {
      // Open the connection if it's not already open.
      if (dconn == null) {
          // This is not the first time this is run, because the
          // dconn hasn't been opened by the constructor.
        // Check if the startup has been due to an incoming
        // datagram.
        connections = PushRegistry.listConnections(true);
        if (connections.length > 0) {
           // There is a pending datagram that can be received.
           dconn = (UDPDatagramConnection)
               Connector.open(connections[0]);
           // Read the datagram
           Datagram d = dconn.newDatagram(dconn.getMaximumLength());
           dconn.receive(d);
        } else {
           // There are not any pending datagrams, but open
           // the connection for later use.
           connections = PushRegistry.listConnections(false);
           if (connections.length > 0) {
                   dconn = (UDPDatagramConnection)
                       Connector.open(connections[0]);
           }
     }
```

```
  }
public void destroyApp(boolean unconditional) {
    // Close the connection before exiting
    if(dconn != null){
       dconn.close()
       dconn = null
    }
}
...
```

Since: MIDP 2.0

Member Summary	
Methods	
static	`getFilter(String connection)`[96]
java.lang.String	
static	`getMIDlet(String connection)`[97]
java.lang.String	
static	`listConnections(boolean available)`[97]
java.lang.String[]	
static long	`registerAlarm(String midlet, long time)`[97]
static void	`registerConnection(String connection, String midlet, String filter)`[98]
static boolean	`unregisterConnection(String connection)`[99]

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Methods

getFilter(String)

Declaration:
`public static String` **`getFilter`**`(String connection)`

Description:
Retrieve the registered filter for a requested connection.

Parameters:
`connection` - generic connection *protocol*, *host* and *port number* (optional parameters may be included separated with semi-colons (;))

Returns: a filter string indicating which senders are allowed to cause the `MIDlet` to be launched or `null`, if the connection was not registered by the current `MIDlet` suite or if the connection argument was `null`

See Also: `registerConnection(String, String, String)`[98]

getMIDlet(String)

Declaration:
```
public static String getMIDlet(String connection)
```

Description:
Retrieve the registered `MIDlet` for a requested connection.

Parameters:
`connection` - generic connection *protocol*, *host* and *port number* (optional parameters may be included separated with semi-colons (;))

Returns: class name of the `MIDlet` to be launched, when new external data is available, or `null` if the connection was not registered by the current `MIDlet` suite or if the connection argument was `null`

See Also: `registerConnection(String, String, String)`[98]

listConnections(boolean)

Declaration:
```
public static String[] listConnections(boolean available)
```

Description:
Return a list of registered connections for the current `MIDlet` suite.

Parameters:
`available` - if `true`, only return the list of connections with input available, otherwise return the complete list of registered connections for the current `MIDlet` suite

Returns: array of registered connection strings, where each connection is represented by the generic connection *protocol*, *host* and *port number* identification

registerAlarm(String, long)

Declaration:
```
public static long registerAlarm(String midlet, long time)
            throws ClassNotFoundException, ConnectionNotFoundException
```

Description:
Register a time to launch the specified application. The `PushRegistry` supports one outstanding wake up time per `MIDlet` in the current suite. An application is expected to use a `TimerTask` for notification of time based events while the application is running.

If a wakeup time is already registered, the previous value will be returned, otherwise a zero is returned the first time the alarm is registered.

Parameters:
`midlet` - class name of the `MIDlet` within the current running `MIDlet` suite to be launched, when the alarm time has been reached. The named `MIDlet` MUST be registered in the descriptor file or the jar file manifest with a MIDlet-<n> record. This parameter has the same semantics as the MIDletClassName in the Push registration attribute defined above in the class description.

`time` - time at which the `MIDlet` is to be executed in the format returned by `Date.getTime()`

Returns: the time at which the most recent execution of this `MIDlet` was scheduled to occur, in the format returned by `Date.getTime()`

Throws:

`ConnectionNotFoundException` - if the runtime system does not support alarm based application launch

`ClassNotFoundException` - if the `MIDlet` class name can not be found in the current `MIDlet` suite or if this class is not included in any of the MIDlet-<n> records in the descriptor file or the jar file manifest or if the `midlet` argument is `null`

`SecurityException` - if the `MIDlet` does not have permission to register an alarm

See Also: `java.util.Date.getTime()`, `java.util.Timer`[40], `java.util.TimerTask`[46]

registerConnection(String, String, String)

Declaration:

```
public static void registerConnection(String connection, String midlet, String filter)
            throws ClassNotFoundException, IOException
```

Description:

Register a dynamic connection with the application management software. Once registered, the dynamic connection acts just like a connection preallocated from the descriptor file.

The arguments for the dynamic connection registration are the same as the Push Registration Attribute used for static registrations.

If the `connection` or `filter` arguments are `null`, then an `IllegalArgumentException` will be thrown. If the `midlet` argument is `null` a `ClassNotFoundException` will be thrown.

Parameters:

`connection` - generic connection *protocol*, *host* and *port number* (optional parameters may be included separated with semi-colons (;))

`midlet` - class name of the `MIDlet` to be launched, when new external data is available. The named `MIDlet` MUST be registered in the descriptor file or the jar file manifest with a MIDlet-<n> record. This parameter has the same semantics as the MIDletClassName in the Push registration attribute defined above in the class description.

`filter` - a connection URL string indicating which senders are allowed to cause the `MIDlet` to be launched

Throws:

`IllegalArgumentException` - if the connection string is not valid, or if the filter string is not valid

`ConnectionNotFoundException` - if the runtime system does not support push delivery for the requested connection protocol

`java.io.IOException` - if the connection is already registered or if there are insufficient resources to handle the registration request

`ClassNotFoundException` - if the `MIDlet` class name can not be found in the current `MIDlet` suite or if this class is not included in any of the MIDlet-<n> records in the descriptor file or the jar file manifest

`SecurityException` - if the `MIDlet` does not have permission to register a connection

See Also: `unregisterConnection(String)`[99]

unregisterConnection(String)

Declaration:
```
public static boolean unregisterConnection(String connection)
```

Description:
Remove a dynamic connection registration.

Parameters:
`connection` - generic connection *protocol*, *host* and *port number*

Returns: `true` if the unregistration was successful, `false` if the connection was not registered or if the connection argument was `null`

Throws:
`SecurityException` - if the connection was registered by another `MIDlet` suite

See Also: `registerConnection(String, String, String)` *98*

javax.microedition.io
SecureConnection

Declaration
`public interface` **`SecureConnection extends`** `SocketConnection`[108]

All Superinterfaces: `Connection`, `InputConnection`, `OutputConnection`, `SocketConnection`[108], `StreamConnection`

Description
This interface defines the secure socket stream connection. A secure connection is established using `Connector.open` with the scheme "ssl" and the secure connection is established before `open` returns. If the secure connection cannot be established due to errors related to certificates a `CertificateException` is thrown.

A secure socket is accessed using a generic connection string with an explicit host and port number. The host may be specified as a fully qualified host name or IPv4 number. e.g. `ssl://host.com:79` defines a target socket on the `host.com` system at port `79`.

Note that RFC1900 recommends the use of names rather than IP numbers for best results in the event of IP number reassignment.

A secure connection MUST be implemented by one or more of the following specifications:

- TLS Protocol Version 1.0 as specified in RFC 2246 (http://www.ietf.org/rfc/rfc2246.txt).
- SSL V3 as specified in The SSL Protocol Version 3.0 (http://home.netscape.com/eng/ssl3/draft302.txt)
- WAP(TM) TLS Profile and Tunneling Specification as specified in WAP-219-TLS-20010411-a (http://www.wapforum.com/what/technical.htm)

BNF Format for Connector.open() string

The URI must conform to the BNF syntax specified below. If the URI does not conform to this syntax, an `IllegalArgumentException` is thrown.

<socket_connection_string>	::= "**ssl:**//"<hostport>
<hostport>	::= *host* ":" *port*
<host>	::= *host name or IP address*
<port>	::= *numeric port number*

Examples

The following examples show how a `SecureConnection` would be used to access a sample loopback program.

```
SecureConnection sc = (SecureConnection)
                      Connector.open("ssl://host.com:79");
SecurityInfo info = sc.getSecurityInfo();
boolean isTLS = (info.getProtocolName().equals("TLS"));

sc.setSocketOption(SocketConnection.LINGER, 5);
InputStream is  = sc.openInputStream();
OutputStream os = sc.openOutputStream();
os.write("\r\n".getBytes());
int ch = 0;
while(ch != -1) {
    ch = is.read();
}
is.close();
os.close();
sc.close();
```

Since: MIDP 2.0

Member Summary

Methods

SecurityInfo getSecurityInfo()[102]

Inherited Member Summary

Fields inherited from interface SocketConnection[108]

DELAY[110], KEEPALIVE[110], LINGER[110], RCVBUF[110], SNDBUF[110]

Methods inherited from interface Connection

close()

Methods inherited from interface InputConnection

openDataInputStream(), openInputStream()

Methods inherited from interface OutputConnection

openDataOutputStream(), openOutputStream()

Methods inherited from interface SocketConnection[108]

getAddress()[110], getLocalAddress()[111], getLocalPort()[111], getPort()[111], getSocketOption(byte)[112], setSocketOption(byte, int)[112]

Methods

getSecurityInfo()

Declaration:

```
public javax.microedition.io.SecurityInfo[103] getSecurityInfo()
            throws IOException
```

Description:

Return the security information associated with this connection when it was opened.

Returns: the security information associated with this open connection.

Throws:

`java.io.IOException` - if an arbitrary connection failure occurs

javax.microedition.io
SecurityInfo

Declaration
```
public interface SecurityInfo
```

Description
This interface defines methods to access information about a secure network connection. Protocols that implement secure connections may use this interface to report the security parameters of the connection.

It provides the certificate, protocol, version, and cipher suite, etc. in use.

Since: MIDP 2.0

See Also: `javax.microedition.pki.CertificateException`[458], `SecureConnection`[100], `HttpsConnection`[85]

Member Summary	
Methods	
`java.lang.String`	`getCipherSuite()`[103]
`java.lang.String`	`getProtocolName()`[104]
`java.lang.String`	`getProtocolVersion()`[104]
`javax.microedition.pki.Certificate`	`getServerCertificate()`[104]

Methods

getCipherSuite()

Declaration:
```
public String getCipherSuite()
```

Description:
Returns the name of the cipher suite in use for the connection. The name returned is from the CipherSuite column of the CipherSuite definitions table in Appendix C of RFC 2246. If the cipher suite is not in Appendix C, the name returned is non-null and its contents are not specified. For non-TLS implementions the cipher suite name should be selected according to the actual key exchange, cipher, and hash combination used to establish the connection, so that regardless of whether the secure connection uses SSL V3 or TLS 1.0 or WTLS or WAP TLS Profile and Tunneling, equivalent cipher suites have the same name.

Returns: a `String` containing the name of the cipher suite in use.

getProtocolName()

Declaration:
`public String getProtocolName()`

Description:
Returns the secure protocol name.

Returns: a `String` containing the secure protocol identifier; if TLS (RFC 2246) or WAP TLS Profile and Tunneling (WAP-219-TLS) is used for the connection the return value is “TLS”; if SSL V3 (The SSL Protocol Version 3.0) is used for the connection; the return value is “SSL”); if WTLS (WAP 199) is used for the connection the return value is “WTLS”.

getProtocolVersion()

Declaration:
`public String getProtocolVersion()`

Description:
Returns the protocol version. If appropriate, it should contain the major and minor versions for the protocol separated with a “.” (Unicode U+002E).

```
For SSL V3 it MUST return “3.0”
For TLS 1.0 it MUST return “3.1”
For WTLS (WAP-199) it MUST return “1”
For WAP TLS Profile and Tunneling Specification it MUST return “3.1”
```

Returns: a String containing the version of the protocol; the return value MUST NOT be `null`.

getServerCertificate()

Declaration:
`public javax.microedition.pki.Certificate`[455] `getServerCertificate()`

Description:
Returns the `Certificate` used to establish the secure connection with the server.

Returns: the `Certificate` used to establish the secure connection with the server.

javax.microedition.io
ServerSocketConnection

Declaration
`public interface` **`ServerSocketConnection extends StreamConnectionNotifier`**

All Superinterfaces: `Connection`, `StreamConnectionNotifier`

Description
This interface defines the server socket stream connection.

A server socket is accessed using a generic connection string with the host omitted. For example, `socket://:79` defines an inbound server socket on port `79`. The host can be discovered using the `getLocalAddress` method.

The `acceptAndOpen()` method returns a `SocketConnection` instance. In addition to the normal `StreamConnection` behavior, the `SocketConnection` supports accessing the IP end point addresses of the live connection and access to socket options that control the buffering and timing delays associated with specific application usage of the connection.

Access to server socket connections may be restricted by the security policy of the device. `Connector.open` MUST check access for the initial server socket connection and `acceptAndOpen` MUST check before returning each new `SocketConnection`.

A server socket can be used to dynamically select an available port by omitting both the host and the port parameters in the connection URL string. For example, `socket://` defines an inbound server socket on a port which is allocated by the system. To discover the assigned port number use the `getLocalPort` method.

BNF Format for Connector.open() string

The URI must conform to the BNF syntax specified below. If the URI does not conform to this syntax, an `IllegalArgumentException` is thrown.

<socket_connection_string>	::= “**socket://**” | “**socket://**”<hostport>
<hostport>	::= *host* “:” *port*
<host>	::= omitted for inbound connections, See SocketConnection
<port>	::= *numeric port number* (omitted for system assigned port)

Examples

The following examples show how a `ServerSocketConnection` would be used to access a sample loopback program.

```
// Create the server listening socket for port 1234
ServerSocketConnection scn = (ServerSocketConnection)
                          Connector.open("socket://:1234");
// Wait for a connection.
SocketConnection sc = (SocketConnection) scn.acceptAndOpen();
// Set application specific hints on the socket.
sc.setSocketOption(DELAY, 0);
sc.setSocketOption(LINGER, 0);
sc.setSocketOption(KEEPALIVE, 0);
sc.setSocketOption(RCVBUF, 128);
sc.setSocketOption(SNDBUF, 128);
// Get the input stream of the connection.
DataInputStream is = sc.openDataInputStream();
// Get the output stream of the connection.
DataOutputStream os = sc.openDataOutputStream();
// Read the input data.
String result = is.readUTF();
// Echo the data back to the sender.
os.writeUTF(result);
// Close everything.
is.close();
os.close();
sc.close();
scn.close();
..
```

Since: MIDP 2.0

Member Summary	
Methods	
java.lang.String	getLocalAddress()[106]
int	getLocalPort()[107]

Inherited Member Summary
Methods inherited from interface `Connection`
`close()`
Methods inherited from interface `StreamConnectionNotifier`
`acceptAndOpen()`

Methods

getLocalAddress()

Declaration:

```
public String getLocalAddress()
            throws IOException
```

Description:
Gets the local address to which the socket is bound.

The host address(IP number) that can be used to connect to this end of the socket connection from an external system. Since IP addresses may be dynamically assigned, a remote application will need to be robust in the face of IP number reasssignment.

The local hostname (if available) can be accessed from `System.getProperty("microedition.hostname")`

Returns: the local address to which the socket is bound.

Throws:
`java.io.IOException` - if the connection was closed

See Also: `SocketConnection`[108]

getLocalPort()

Declaration:
```
public int getLocalPort()
            throws IOException
```

Description:
Returns the local port to which this socket is bound.

Returns: the local port number to which this socket is connected.

Throws:
`java.io.IOException` - if the connection was closed

See Also: `SocketConnection`[108]

javax.microedition.io
SocketConnection

Declaration
`public interface` **`SocketConnection extends StreamConnection`**

All Superinterfaces: `Connection`, `InputConnection`, `OutputConnection`, `StreamConnection`

All Known Subinterfaces: `SecureConnection`[100]

Description
This interface defines the socket stream connection.

A socket is accessed using a generic connection string with an explicit host and port number. The host may be specified as a fully qualified host name or IPv4 number. e.g. `socket://host.com:79` defines a target socket on the `host.com` system at port `79`.

Note that RFC1900 recommends the use of names rather than IP numbers for best results in the event of IP number reassignment.

Closing Streams
Every `StreamConnection` provides a `Connection` object as well as an `InputStream` and `OutputStream` to handle the I/O associated with the connection. Each of these interfaces has its own `close()` method. For systems that support duplex communication over the socket connection, closing of the input or output stream SHOULD shutdown just that side of the connection. e.g. closing the `InputStream` will permit the `OutputStream` to continue sending data.

Once the input or output stream has been closed, it can only be reopened with a call to `Connector.open()`. The application will receive an `IOException` if an attempt is made to reopen the stream.

BNF Format for Connector.open() string

The URI must conform to the BNF syntax specified below. If the URI does not conform to this syntax, an `IllegalArgumentException` is thrown.

<socket_connection_string>	::= “**socket:**//”<hostport>
<hostport>	::= *host* “:” *port*
<host>	::= *host name or IP address* (omitted for inbound connections, See ServerSocketConnection)
<port>	::= *numeric port number*

Examples

The following examples show how a `SocketConnection` would be used to access a sample loopback program.

```
SocketConnection sc = (SocketConnection)
                      Connector.open("socket://host.com:79");
sc.setSocketOption(SocketConnection.LINGER, 5);
InputStream is  = sc.openInputStream();
OutputStream os = sc.openOutputStream();
os.write("\r\n".getBytes());
int ch = 0;
while(ch != -1) {
    ch = is.read();
}
is.close();
os.close();
sc.close();
```

Since: MIDP 2.0

Member Summary

Fields

static byte	DELAY[110]
static byte	KEEPALIVE[110]
static byte	LINGER[110]
static byte	RCVBUF[110]
static byte	SNDBUF[110]

Methods

java.lang.String	getAddress()[110]
java.lang.String	getLocalAddress()[111]
int	getLocalPort()[111]
int	getPort()[111]
int	getSocketOption(byte option)[112]
void	setSocketOption(byte option, int value)[112]

Inherited Member Summary

Methods inherited from interface `Connection`

`close()`

Methods inherited from interface `InputConnection`

`openDataInputStream(), openInputStream()`

Methods inherited from interface `OutputConnection`

`openDataOutputStream(), openOutputStream()`

Fields

DELAY

Declaration:
```
public static final byte DELAY
```

Description:
Socket option for the small buffer *writing delay* (0). Set to zero to disable Nagle algorithm for small buffer operations. Set to a non-zero value to enable.

KEEPALIVE

Declaration:
```
public static final byte KEEPALIVE
```

Description:
Socket option for the *keep alive* feature (2). Setting KEEPALIVE to zero will disable the feature. Setting KEEPALIVE to a non-zero value will enable the feature.

LINGER

Declaration:
```
public static final byte LINGER
```

Description:
Socket option for the *linger time* to wait in seconds before closing a connection with pending data output (1). Setting the linger time to zero disables the linger wait interval.

RCVBUF

Declaration:
```
public static final byte RCVBUF
```

Description:
Socket option for the size of the *receiving buffer* (3).

SNDBUF

Declaration:
```
public static final byte SNDBUF
```

Description:
Socket option for the size of the *sending buffer* (4).

Methods

getAddress()

Declaration:
```
public String getAddress()
            throws IOException
```

Description:
Gets the remote address to which the socket is bound. The address can be either the remote host name or the IP address(if available).

Returns: the remote address to which the socket is bound.

Throws:
`java.io.IOException` - if the connection was closed.

getLocalAddress()

Declaration:
```
public String getLocalAddress()
            throws IOException
```

Description:
Gets the local address to which the socket is bound.

The host address(IP number) that can be used to connect to this end of the socket connection from an external system. Since IP addresses may be dynamically assigned, a remote application will need to be robust in the face of IP number reasssignment.

The local hostname (if available) can be accessed from
`System.getProperty("microedition.hostname")`

Returns: the local address to which the socket is bound.

Throws:
`java.io.IOException` - if the connection was closed.

See Also: `ServerSocketConnection`$_{105}$

getLocalPort()

Declaration:
```
public int getLocalPort()
            throws IOException
```

Description:
Returns the local port to which this socket is bound.

Returns: the local port number to which this socket is connected.

Throws:
`java.io.IOException` - if the connection was closed.

See Also: `ServerSocketConnection`$_{105}$

getPort()

Declaration:
```
public int getPort()
            throws IOException
```

Description:
Returns the remote port to which this socket is bound.

Returns: the remote port number to which this socket is connected.

Throws:
`java.io.IOException` - if the connection was closed.

getSocketOption(byte)

Declaration:

```
public int getSocketOption(byte option)
            throws IllegalArgumentException, IOException
```

Description:
Get a socket option for the connection.

Parameters:
`option` - socket option identifier (KEEPALIVE, LINGER, SNDBUF, RCVBUF, or DELAY)

Returns: numeric value for specified option or -1 if the value is not available.

Throws:
`IllegalArgumentException` - if the option identifier is not valid

`java.io.IOException` - if the connection was closed

See Also: `setSocketOption(byte, int)` 112

setSocketOption(byte, int)

Declaration:

```
public void setSocketOption(byte option, int value)
            throws IllegalArgumentException, IOException
```

Description:
Set a socket option for the connection.

Options inform the low level networking code about intended usage patterns that the application will use in dealing with the socket connection.

Calling `setSocketOption` to assign buffer sizes is a hint to the platform of the sizes to set the underlying network I/O buffers. Calling `getSocketOption` can be used to see what sizes the system is using. The system MAY adjust the buffer sizes to account for better throughput available from Maximum Transmission Unit (MTU) and Maximum Segment Size (MSS) data available from current network information.

Parameters:
`option` - socket option identifier (KEEPALIVE, LINGER, SNDBUF, RCVBUF, or DELAY)

`value` - numeric value for specified option

Throws:
`IllegalArgumentException` - if the value is not valid (e.g. negative value) or if the option identifier is not valid

`java.io.IOException` - if the connection was closed

See Also: `getSocketOption(byte)` 112

javax.microedition.io
UDPDatagramConnection

Declaration
`public interface` **`UDPDatagramConnection extends DatagramConnection`**

All Superinterfaces: `Connection`, `DatagramConnection`

Description
This interface defines a datagram connection which knows it's local end point address. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the `SocketConnection`.

A `UDPDatagramConnection` is returned from `Connector.open()` in response to a request to open a `datagram://` URL connection string. If the connection string omits both the `host` and `port` fields in the URL string, then the system will allocate an available port. The local address and the local port can be discovered using the accessor methods within this interface.

The syntax described here for the datagram URL connection string is also valid for the `Datagram.setAddress()` method used to assign a destination address to a `Datagram` to be sent. e.g., `datagram://`*host:port*

BNF Format for Connector.open() string

The URI must conform to the BNF syntax specified below. If the URI does not conform to this syntax, an `IllegalArgumentException` is thrown.

<datagram_connection_string>	::= "**datagram://**" | "**datagram://**"<hostport>
<hostport>	::= *host* ":" *port*
<host>	::= *host name or IP address* (omitted for inbound connections)
<port>	::= *numeric port number* (omitted for system assigned port)

Since: MIDP 2.0

Member Summary	
Methods	
`java.lang.String`	`getLocalAddress()`[114]
`int`	`getLocalPort()`[114]

Inherited Member Summary

Methods inherited from interface `Connection`

`close()`

Methods inherited from interface `DatagramConnection`

`getMaximumLength()`, `getNominalLength()`, `newDatagram(byte[], int, String)`, `newDatagram(byte[], int, String)`, `newDatagram(byte[], int, String)`, `receive(Datagram)`, `send(Datagram)`

Methods

getLocalAddress()

Declaration:

```
public String getLocalAddress()
            throws IOException
```

Description:

Gets the local address to which the datagram connection is bound.

The host address(IP number) that can be used to connect to this end of the datagram connection from an external system. Since IP addresses may be dynamically assigned, a remote application will need to be robust in the face of IP number reassignment.

The local hostname (if available) can be accessed from `System.getProperty("microedition.hostname")`

Returns: the local address to which the datagram connection is bound.

Throws:

`java.io.IOException` - if the connection was closed.

See Also: `ServerSocketConnection`$_{105}$

getLocalPort()

Declaration:

```
public int getLocalPort()
            throws IOException
```

Description:

Returns the local port to which this datagram connection is bound.

Returns: the local port number to which this datagram connection is connected.

Throws:

`java.io.IOException` - if the connection was closed.

See Also: `ServerSocketConnection`$_{105}$

CHAPTER 8

Package javax.microedition.lcdui

Description

The UI API provides a set of features for implementation of user interfaces for MIDP applications.

User Interface

The main criteria for the MIDP have been drafted with mobile information devices in mind (i.e., mobile phones and pagers). These devices differ from desktop systems in many ways, especially how the user interacts with them. The following UI-related requirements are important when designing the user interface API:

- The devices and applications should be useful to users who are not necessarily experts in using computers.
- The devices and applications should be useful in situations where the user cannot pay full attention to the application. For example, many phone-type devices will be operated with one hand.
- The form factors and UI concepts of the device differ between devices, especially from desktop systems. For example, the display sizes are smaller, and the input devices do not always include pointing devices.
- The applications run on MIDs should have UIs that are compatible to the native applications so that the user finds them easy to use.

Given the capabilities of devices that will implement the MIDP and the above requirements, the MIDPEG decided not to simply subset the existing Java UI, which is the Abstract Windowing Toolkit (AWT). Reasons for this decision include:

- Although AWT was designed for desktop computers and optimized to these devices, it also suffers from assumptions based on this heritage.
- When a user interacts with AWT, event objects are created dynamically. These objects are short-lived and exist only until each associated event is processed by the system. At this point, the event object becomes garbage and must be reclaimed by the system’s garbage collector. The limited CPU and memory subsystems of a MID typically cannot handle this behavior.
- AWT has a rich but desktop-based feature set. This feature set includes support for features not found on MIDs. For example, AWT has extensive support for window management (e.g., overlapping windows, window resize, etc.). MIDs have small displays which are not large enough to display multiple overlapping windows. The limited display size also makes resizing a window impractical. As such, the windowing and layout manager support within AWT is not required for MIDs.
- AWT assumes certain user interaction models. The component set of AWT was designed to work with a pointer device (e.g., a mouse or pen input). As mentioned earlier, this assumption is valid for only a small subset of MIDs since many of these devices have only a keypad for user input.

Structure of the MIDP UI API

The MIDP UI is logically composed of two APIs: the high-level and the low-level.

The high-level API is designed for business applications whose client parts run on MIDs. For these applications, portability across devices is important. To achieve this portability, the high-level API employs a high level of

abstraction and provides very little control over look and feel. This abstraction is further manifested in the following ways:

- The actual drawing to the MID's display is performed by the implementation. Applications do not define the visual appearance (e.g., shape, color, font, etc.) of the components.
- Navigation, scrolling, and other primitive interaction is encapsulated by the implementation, and the application is not aware of these interactions.
- Applications cannot access concrete input devices like specific individual keys.

In other words, when using the high-level API, it is assumed that the underlying implementation will do the necessary adaptation to the device's hardware and native UI style. The classes that provide the high-level API are the subclasses of `Screen`315.

The low-level API, on the other hand, provides very little abstraction. This API is designed for applications that need precise placement and control of graphic elements, as well as access to low-level input events. Some applications also need to access special, device-specific features. A typical example of such an application would be a game.

Using the low-level API, an application can:

- Have full control of what is drawn on the display.
- Listen for primitive events like key presses and releases.
- Access concrete keys and other input devices.

The classes that provide the low-level API are `Canvas`139 and `Graphics`247.

Applications that program to the low-level API are not guaranteed to be portable, since the low-level API provides the means to access details that are specific to a particular device. If the application does not use these features, it will be portable. It is recommended that applications use only the platform-independent part of the low-level API whenever possible. This means that the applications should not directly assume the existence of any keys other than those defined in the `Canvas` class, and they should not depend on a specific screen size. Rather, the application game-key event mapping mechanism should be used instead of concrete keys, and the application should inquire about the size of the display and adjust itself accordingly.

Class Hierarchy

The central abstraction of the MIDP's UI is a `Displayable` object, which encapsulates device-specific graphics rendering with user input. Only one `Displayable` may be visible at a time, and and the user can see and interact with only contents of that `Displayable`.

The `Screen` class is a subclass of `Displayable` that takes care of all user interaction with high-level user interface component. The `Screen` subclasses handle rendering, interaction, traversal, and scrolling, with only higher-level events being passed on to the application.

The rationale behind this design is based on the different display and input solutions found in MIDP devices. These differences imply that the component layout, scrolling, and focus traversal will be implemented differently on different devices. If an application were required to be aware of these issues, portability would be compromised. Simple screenfuls also organize the user interface into manageable pieces, resulting in user interfaces that are easy to use and learn.

There are three categories of `Displayable` objects:

- Screens that encapsulate a complex user interface component (e.g., classes `List` or `TextBox`). The structure of these screens is predefined, and the application cannot add other components to these screens.
- Generic screens (instances of the `Form` class) that can contain `Item` objects to represent user interface components. The application can populate `Form` objects with an arbitrary number of text, image, and other

components; however, it is recommended that `Form` objects be kept simple and that they should be used to contain only a few, closely-related user interface components.

- Screens that are used in context of the low-level API (i.e., subclasses of class `Canvas`).

Each `Displayable` can have a title, a `Ticker` and a set of `Commands` attached to it.

The class `Display` acts as the display manager that is instantiated for each active `MIDlet` and provides methods to retrieve information about the device's display capabilities. A `Displayable` is made visible by calling the `setCurrent()` method of `Display`. When a `Displayable` is made current, it replaces the previous `Displayable`.

Class Overview

It is anticipated that most applications will utilize screens with predefined structures like `List` , `TextBox` , and `Alert` . These classes are used in the following ways:

- `List` is used when the user should select from a predefined set of choices.
- `TextBox` is used when asking textual input.
- `Alert` is used to display temporary messages containing text and images.

A special class `Form` is defined for cases where screens with a predefined structure are not sufficient. For example, an application may have two `TextFields`, or a `TextField` and a simple `ChoiceGroup` . Although this class (`Form`) allows creation of arbitrary combinations of components, developers should keep the limited display size in mind and create only simple `Forms` .

`Form` is designed to contain a small number of closely related UI elements. These elements are the subclasses of `Item`: `ImageItem`, `StringItem`, `TextField`, `ChoiceGroup`, `Gauge`, and `CustomItem`. The classes `ImageItem` and `StringItem` are convenience classes that make certain operations with `Form` and `Alert` easier. By subclassing `CustomItem` application developers can introduce `Items` with a new visual representation and interactive elements. If the components do not all fit on the screen, the implementation may either make the form scrollable or implement some components so that they can either popup in a new screen or expand when the user edits the element.

Interplay with Application Manager

The user interface, like any other resource in the API, is to be controlled according to the principle of MIDP application management. The UI expects the following conditions from the application management software:

- `getDisplay()` is callable starting from `MIDlet`'s constructor until `destroyApp()` has returned.
- The `Display` object is the same until `destroyApp()` is called.
- The `Displayable` object set by `setCurrent()` is not changed by the application manager.

The application manager assumes that the application behaves as follows with respect to the `MIDlet` events:

- `startApp` - The application may call `setCurrent()` for the first screen. The application manager makes `Displayable` really visible when `startApp()` returns. Note that `startApp()` can be called several times if `pauseApp()` is called in between. This means that initialization should not take place, and the application should not accidentally switch to another screen with `setCurrent()` .
- `pauseApp` - The application should release as many threads as possible. Also, if starting with another screen when the application is re-activated, the new screen should be set with `setCurrent()` .
- `destroyApp` - The application may delete created objects.

Event Handling

User interaction causes events, and the implementation notifies the application of the events by making corresponding callbacks. There are four kinds of UI callbacks:

- Abstract commands that are part of the high-level API
- Low-level events that represent single key presses and releases (and pointer events, if a pointer is available)
- Calls to the `paint()` method of a `Canvas` class
- Calls to a `Runnable` object's `run()` method requested by a call to `callSerially()` of class `Display`

All UI callbacks are serialized, so they will never occur in parallel. That is, the implementation will never call an callback before a prior call to *any* other callback has returned. This property enables applications to be assured that processing of a previous user event will have completed before the next event is delivered. If multiple UI callbacks are pending, the next is called as soon as possible after the previous UI callback returns. The implementation also guarantees that the call to `run()` requested by a call to `callSerially()` is made after any pending repaint requests have been satisfied.

There is one exception to the callback serialization rule, which occurs when the `Canvas.serviceRepaints`[153] method is called. This method causes the the `Canvas.paint` method to be called and waits for it to complete. This occurs even if the caller of `serviceRepaints` is itself within an active callback. There is further discussion of this issue below.

The following callbacks are all serialized with respect to each other:

- `Canvas.hideNotify`[149]
- `Canvas.keyPressed`[150]
- `Canvas.keyRepeated`[150]
- `Canvas.keyReleased`[150]
- `Canvas.paint`[151]
- `Canvas.pointerDragged`[152]
- `Canvas.pointerPressed`[152]
- `Canvas.pointerReleased`[152]
- `Canvas.showNotify`[154]
- `Canvas.sizeChanged`[154]
- `CommandListener.commandAction`[183]
- `CustomItem.getMinContentHeight`[191]
- `CustomItem.getMinContentWidth`[191]
- `CustomItem.getPrefContentHeight`[191]
- `CustomItem.getPrefContentWidth`[192]
- `CustomItem.hideNotify`[192]
- `CustomItem.keyPressed`[193]
- `CustomItem.keyRepeated`[193]

- CustomItem.keyReleased 193
- CustomItem.paint 193
- CustomItem.pointerDragged 194
- CustomItem.pointerPressed 194
- CustomItem.pointerReleased 195
- CustomItem.showNotify 195
- CustomItem.sizeChanged 196
- CustomItem.traverse 196
- CustomItem.traverseOut 199
- Displayable.sizeChanged 222
- ItemCommandListener.commandAction 300
- ItemStateListener.itemStateChanged 301
- Runnable.run resulting from a call to Display.callSerially 210

Note that Timer 40 events are not considered UI events. Timer callbacks may run concurrently with UI event callbacks, although TimerTask 46 callbacks scheduled on the same Timer are serialized with each other. Applications that use timers must guard their data structures against concurrent access from timer threads and UI event callbacks. Alternatively, applications may have their timer callbacks use Display.callSerially 210 so that work triggered by timer events can be serialized with the UI event callbacks.

Abstract Commands

Since MIDP UI is highly abstract, it does not dictate any concrete user interaction technique like soft buttons or menus. Also, low-level user interactions such as traversal or scrolling are not visible to the application. MIDP applications define Commands , and the implementation may manifest these via either soft buttons, menus, or whatever mechanisms are appropriate for that device.

Commands are installed to a Displayable (Canvas or Screen) with a method addCommand of class Displayable .

The native style of the device may assume that certain types of commands are placed on standard places. For example, the “go-back” operation may always be mapped to the right soft button. The Command class allows the application to communicate such a semantic meaning to the implementation so that these standard mappings can be effected.

The implementation does not actually implement any of the semantics of the Command. The attributes of a Command are used only for mapping it onto the user interface. The actual semantics of a Command are always implemented by the application in a CommandListener.

Command objects have attributes:

- Label: Shown to the user as a hint. A single Command can have two versions of labels: short and long. The implementation decides whether the short or long version is appropriate for a given situation. For example, an implementaion can choose to use a short version of a given Command near a soft button and the long version of the Command in a menu.
- Type: The purpose of a command. The implementation will use the command type for placing the command appropriately within the device’s user interface. Commands with similar types may, for example,

be found near each other in certain dedicated place in the user interface. Often, devices will have policy for placement and presentation of certain operations. For example, a “backward navigation” command might be always placed on the right soft key on a particular device, but it might be placed on the left soft key on a different device. The `Command` class provides fixed set of command types that provide `MIDlet` the capability to tell the device implementation the intent of a `Command`. The application can use the `BACK` command type for commands that perform backward navigation. On the devices mentioned above, this type information would be used to assign the command to the appropriate soft key.

- Priority: Defines the relative importance between `Commands` of the same type. A command with a lower priority value is more important than a command of the same type but with a higher priority value. If possible, a more important command is presented before, or is more easily accessible, than a less important one.

Device-Provided Operations

In many high-level UI classes there are also some additional operations available in the user interface. The additional operations are not visible to applications, only to the end-user. The set of operations available depends totally on the user interface design of the specific device. For example, an operation that allows the user to change the mode for text input between alphabetic and numeric is needed in devices that have only an ITU-T keypad. More complex input systems will require additional operations. Some of operations available are presented in the user interface in the same way the application-defined commands are. End-users need not understand which operations are provided by the application and which provided by the system. Not all operations are available in every implementation. For example, a system that has a word-lookup-based text input scheme will generally provide additional operations within the `TextBox` class. A system that lacks such an input scheme will also lack the corresponding operations.

Some operations are available on all devices, but the way the operation is implemented may differ greatly from device to device. Examples of this kind of operation are: the mechanism used to navigate between `List` elements and `Form` items, the selection of `List` elements, moving an insertion position within a text editor, and so forth. Some devices do not allow the direct editing of the value of an `Item`, but instead require the user to switch to an off-screen editor. In such devices, there must be a dedicated selection operation that can be used to invoke the off-screen editor. The selection of a `List` elements could be, for example, implemented with a dedicated “Go” or “Select” or some other similar key. Some devices have no dedicated selection key and must select elements using some other means.

On devices where the selection operation is performed using a dedicated select key, this key will often not have a label displayed for it. It is appropriate for the implementation to use this key in situations where its meaning is obvious. For example, if the user is presented with a set of mutually exclusive options, the selection key will obviously select one of those options. However, in a device that doesn’t have a dedicated select key, it is likely that the selection operation will be performed using a soft key that requires a label. The ability to set the select-command for a `List` of type `IMPLICIT` and the ability to set the default command for an `Item` are provided so that the application can set the label for this operation and so it can receive notification when this operation occurs.

High-Level API for Events

The handling of events in the high-level API is based on a listener model. `Screens` and `Canvases` may have listeners for commands. An object willing to be a listener should implement an interface `CommandListener` that has one method:

```
void commandAction(Command c, Displayable d);
```

The application gets these events if the `Screen` or `Canvas` has attached `Commands` and if there is a registered listener. A unicast-version of the listener model is adopted, so the `Screen` or `Canvas` can have one listener at a time.

There is also a listener interface for state changes of the `Items` in a `Form` . The method

```
void itemStateChanged(Item item);
```

defined in interface `ItemStateListener` is called when the value of an interactive `Gauge` , `ChoiceGroup` , or `TextField` changes. It is not expected that the listener will be called after every change. However, if the value of an Item has been changed, the listener will be called for the change sometime before it is called for another item or before a command is delivered to the `Form's CommandListener`. It is suggested that the change listener is called at least after focus (or equivalent) is lost from field. The listener should only be called if the field's value has actually changed.

Low-Level API for Events

Low-level graphics and events have the following methods to handle low-level key events:

```
public void keyPressed(int keyCode);
public void keyReleased(int keyCode);
public void keyRepeated(int keyCode);
```

The last call, `keyRepeated` , is not necessarily available in all devices. The applications can check the availability of repeat actions by calling the following method of the `Canvas` :

```
public static boolean hasRepeatEvents();
```

The API requires that there be standard key codes for the ITU-T keypad (0-9, *, #), but no keypad layout is required by the API. Although an implementation may provide additional keys, applications relying on these keys are not portable.

In addition, the class `Canvas` has methods for handling abstract game events. An implementation maps all these key events to suitable keys on the device. For example, a device with four-way navigation and a select key in the middle could use those keys, but a simpler device may use certain keys on the numeric keypad (e.g., `2`, `4`, `5`, `6`, `8`). These game events allow development of portable applications that use the low-level events. The API defines a set of abstract key-events: `UP`, `DOWN`, `LEFT`, `RIGHT`, `FIRE`, `GAME_A`, `GAME_B`, `GAME_C`, and `GAME_D`.

An application can get the mapping of the key events to abstract key events by calling:

```
public static int getGameAction(int keyCode);
```

If the logic of the application is based on the values returned by this method, the application is portable and run regardless of the keypad design.

It is also possible to map an abstract event to a key with:

```
public static int getKeyCode(int gameAction);
```

where `gameAction` is `UP`,`DOWN`, `LEFT`, `RIGHT`, `FIRE`, etc. On some devices, more than one key is mapped to the same game action, in which case the `getKeyCode` method will return just one of them. Properly-written applications should map the key code to an abstract key event and make decisions based on the result.

The mapping between keys and abstract events does not change during the execution of the game.

The following is an example of how an application can use game actions to interpret keystrokes.

```
class MovingBlocksCanvas extends Canvas {
    public void keyPressed(int keyCode) {
        int action = getGameAction(keyCode);
        switch (action) {
        case LEFT:
            moveBlockLeft();
            break;
        case RIGHT:
            ...
        }
    }
}
```

The low-level API also has support for pointer events, but since the following input mechanisms may not be present in all devices, the following callback methods may never be called in some devices:

```
public void pointerPressed(int x, int y);
public void pointerReleased(int x, int y);
public void pointerDragged(int x, int y);
```

The application may check whether the pointer is available by calling the following methods of class `Canvas` :

```
public static boolean hasPointerEvents();
public static boolean hasPointerMotionEvents();
```

Interplay of High-Level Commands and the Low-Level API

The class `Canvas` , which is used for low-level events and drawing, is a subclass of `Displayable` , and applications can attach `Commands` to it. This is useful for jumping to an options setup `Screen` in the middle of a game. Another example could be a map-based navigation application where keys are used for moving in the map but commands are used for higher-level actions.

Some devices may not have the means to invoke commands when `Canvas` and the low-level event mechanism are in use. In that case, the implementation may provide a means to switch to a command mode and back. This command mode might pop up a menu over the contents of the `Canvas`. In this case, the `Canvas` methods `hideNotify()` and `showNotify()` will be called to indicate when the `Canvas` has been obscured and unobscured, respectively.

The `Canvas` may have a title and a `Ticker` like the `Screen` objects. However, `Canvas` also has a full-screen mode where the title and the `Ticker` are not displayed. Setting this mode indicates that the application wishes for the `Canvas` to occupy as much of the physical display as is possible. In this mode, the title may be reused by the implementation as the title for pop-up menus. In normal (not full-screen) mode, the appearance of the `Canvas` should be similar to that of `Screen` classes, so that visual continuity is retained when the application switches between low-level `Canvas` objects and high-level `Screen` objects.

Graphics and Text in Low-Level API

The Redrawing Scheme

Repainting is done automatically for all `Screens` , but not for `Canvas` ; therefore, developers utilizing the low-level API must ; understand its repainting scheme.

In the low-level API, repainting of `Canvas` is done asynchronously so that several repaint requests may be implemented within a single call as an optimization. This means that the application requests the repainting by calling the method `repaint()` of class `Canvas` . The actual drawing is done in the method `paint()` —- which is provided by the subclass `Canvas` —- and does not necessarily happen synchronously to `repaint()` . It may happen later, and several repaint requests may cause one single call to `paint()` . The application can flush the repaint requests by calling `serviceRepaints()` .

As an example, assume that an application moves a box of width `wid` and height `ht` from coordinates (`x1,y1`) to coordinates (`x2,y2`), where `x2>x1` and `y2>y1` :

```
// move coordinates of box
box.x = x2;
box.y = y2;

// ensure old region repainted (with background)
canvas.repaint(x1,y1, wid, ht);

// make new region repainted
canvas.repaint(x2,y2, wid, ht);

// make everything really repainted
canvas.serviceRepaints();
```

The last call causes the repaint thread to be scheduled. The repaint thread finds the two requests from the event queue and repaints the region that is a union of the repaint area:

```
graphics.clipRect(x1,y1, (x2-x1+wid), (y2-y1+ht));
canvas.paint(graphics);
```

In this imaginary part of an implementation, the call `canvas.paint()` causes the application-defined `paint()` method to be called.

Drawing Model

The primary drawing operation is pixel replacement, which is used for geometric rendering operations such as lines and rectangles. With offscreen images, support for full transparency is required, and support for partial transparency (alpha blending) is optional.

A 24-bit color model is provided with 8 bits each for the red, green, and blue components of a color. Not all devices support 24-bit color, so they will map colors requested by the application into colors available on the device. Facilities are provided in the `Display` class for obtaining device characteristics, such as whether color is available and how many distinct gray levels are available. This enables applications to adapt their behavior to a device without compromising device independence.

Graphics may be rendered either directly to the display or to an off-screen image buffer. The destination of rendered graphics depends on the origin of the graphics object. A graphics object for rendering to the display is passed to the `Canvas` object's `paint()` method. This is the only way to obtain a graphics object whose destination is the display. Furthermore, applications may draw by using this graphics object only for the duration of the `paint()` method.

A graphics object for rendering to an off-screen image buffer may be obtained by calling the `getGraphics()` method on the desired image. These graphics objects may be held indefinitely by the application, and requests may be issued on these graphics objects at any time.

The `Graphics` class has a current color that is set with the `setColor()` method. All geometric rendering, including lines, rectangles, and arcs, uses the current color. The pixel representing the current color replaces the destination pixel in these operations. There is no background color. Painting of any background be performed explicitly by the application using the `setColor()` and rendering calls.

Support for full transparency is required, and support for partial transparency (alpha blending) is optional. Transparency (both full and partial) exists only in off-screen images loaded from PNG files or from arrays of ARGB data. Images created in such a fashion are *immutable* in that the application is precluded from making any changes to the pixel data contained within the image. Rendering is defined in such a way that the destination of any rendering operation always consists entirely of fully opaque pixels.

Coordinate System

The origin `(0,0)` of the available drawing area and images is in the upper-left corner of the display. The numeric values of the x-coordinates monotonically increase from left to right, and the numeric values of the y-coordinates monotonically increase from top to bottom. Applications may assume that horizontal and vertical distances in the coordinate system represent equal distances on the actual device display. If the shape of the pixels of the device is significantly different from square, the implementation of the UI will do the required coordinate transformation. A facility is provided for translating the origin of the coordinate system. All coordinates are specified as integers.

The coordinate system represents locations between pixels, not the pixels themselves. Therefore, the first pixel in the upper left corner of the display lies in the square bounded by coordinates `(0,0), (1,0), (0,1), (1,1)`.

An application may inquire about the available drawing area by calling the following methods of `Canvas` :

```
public static final int getWidth();
public static final int getHeight();
```

Font Support

An application may request one of the font attributes specified below. However, the underlying implementation may use a subset of what is specified. So it is up to the implementation to return a font that most closely resembles the requested font.

Each font in the system is implemented individually. A programmer will call the static `getFont()` method instead of instantiating new `Font` objects. This paradigm eliminates the garbage creation normally associated with the use of fonts.

The `Font` class provides calls that access font metrics. The following attributes may be used to request a font (from the `Font` class):

- Size: `SMALL`, `MEDIUM`, `LARGE`.
- Face: `PROPORTIONAL`, `MONOSPACE`, `SYSTEM`.
- Style: `PLAIN`, `BOLD`, `ITALIC`, `UNDERLINED`.

Concurrency

The UI API has been designed to be thread-safe. The methods may be called from callbacks, `TimerTasks`, or other threads created by the application. Also, the implementation generally does not hold any locks on objects visible to the application. This means that the applications' threads can synchronize with themselves and with the event callbacks by locking any object according to a synchronization policy defined by the application. One exception to this rule occurs with the `Canvas.serviceRepaints`[153] method. This method calls and awaits completion of the `paint` method. Strictly speaking, `serviceRepaints` might not call `paint` directly, but instead it might cause another thread to call `paint`. In either case, `serviceRepaints` blocks until `paint` has returned. This is a significant point because of the following case. Suppose the caller of `serviceRepaints` holds a lock that is also needed by the `paint` method. Since `paint` might be called from another thread, that thread will block trying to acquire the lock. However, this lock is held by the caller of `serviceRepaints`, which is blocked waiting for `paint` to return. The result is deadlock. In order to avoid deadlock, the caller of `serviceRepaints` *must not* hold any locks needed by the `paint` method.

The UI API includes also a mechanism similar to other UI toolkits for serializing actions with the event stream. The method `Display.callSerially`[210] requests that the `run` method of a `Runnable` object be called, serialized with the event stream. Code that uses `serviceRepaints()` can usually be rewritten to use `callSerially()`. The following code illustrates this technique:

```
class MyCanvas extends Canvas {
    void doStuff() {
        // <code fragment 1>
        serviceRepaints();
        // <code fragment 2>
    }
}
```

The following code is an alternative way of implementing the same functionality:

Implementation Notes

The implementation of a `List` or `ChoiceGroup` may include keyboard shortcuts for focusing and selecting the choice elements, but the use of these shortcuts is not visible to the application program.

```
class MyClass extends Canvas
  implements Runnable {
   void doStuff() {
      // <code fragment 1>
      callSerially(this);
   }
   // called only after all pending repaints served
   public void run() {
      // <code fragment 2>;
   }
}
```

In some implementations the UI components —- `Screens` and `Items` —- will be based on native components. It is up to the implementation to free the used resources when the Java objects are not needed anymore. One possible implementation scenario is a hook in the garbage collector of KVM.

Since: MIDP 1.0

Class Summary

Interfaces

`Choice` 155	Choice defines an API for a user interface components implementing selection from predefined number of choices.
`CommandListener` 183	This interface is used by applications which need to receive high-level events from the implementation.
`ItemCommandListener` 300	A listener type for receiving notification of commands that have been invoked on `Item` 287 objects.
`ItemStateListener` 301	This interface is used by applications which need to receive events that indicate changes in the internal state of the interactive items within a `Form` 231 screen.

Classes

`Alert` 128	An alert is a screen that shows data to the user and waits for a certain period of time before proceeding to the next `Displayable`.
`AlertType` 136	The `AlertType` provides an indication of the nature of alerts.
`Canvas` 139	The `Canvas` class is a base class for writing applications that need to handle low-level events and to issue graphics calls for drawing to the display.
`ChoiceGroup` 166	A `ChoiceGroup` is a group of selectable elements intended to be placed within a `Form` 231 .
`Command` 175	The `Command` class is a construct that encapsulates the semantic information of an action.
`CustomItem` 184	A CustomItem is customizable by subclassing to introduce new visual and interactive elements into `Forms`.
`DateField` 201	A `DateField` is an editable component for presenting date and time (calendar) information that may be placed into a `Form`.
`Display` 205	`Display` represents the manager of the display and input devices of the system.

Class Summary	
Displayable 218	An object that has the capability of being placed on the display.
Font 223	The Font class represents fonts and font metrics.
Form 231	A Form is a Screen that contains an arbitrary mixture of items: images, read-only text fields, editable text fields, editable date fields, gauges, choice groups, and custom items.
Gauge 240	Implements a graphical display, such as a bar graph, of an integer value.
Graphics 247	Provides simple 2D geometric rendering capability.
Image 270	The Image class is used to hold graphical image data.
ImageItem 281	An item that can contain an image.
Item 287	A superclass for components that can be added to a Form 231 .
List 303	A Screen containing list of choices.
Screen 315	The common superclass of all high-level user interface classes.
Spacer 316	A blank, non-interactive item that has a settable minimum size.
StringItem 319	An item that can contain a string.
TextBox 323	The TextBox class is a Screen that allows the user to enter and edit text.
TextField 330	A TextField is an editable text component that may be placed into a Form 231 .
Ticker 345	Implements a “ticker-tape”, a piece of text that runs continuously across the display.

javax.microedition.lcdui
Alert

Declaration

```
public class Alert extends Screen 315
```

```
Object
  |
  +--javax.microedition.lcdui.Displayable 218
        |
        +--javax.microedition.lcdui.Screen 315
              |
              +--javax.microedition.lcdui.Alert
```

Description

An alert is a screen that shows data to the user and waits for a certain period of time before proceeding to the next `Displayable`. An alert can contain a text string and an image. The intended use of `Alert` is to inform the user about errors and other exceptional conditions.

The application can set the alert time to be infinity with `setTimeout(Alert.FOREVER)` in which case the `Alert` is considered to be *modal* and the implementation provide a feature that allows the user to "dismiss" the alert, whereupon the next `Displayable` is displayed as if the timeout had expired immediately.

If an application specifies an alert to be of a timed variety *and* gives it too much content such that it must scroll, then it automatically becomes a modal alert.

An alert may have an `AlertType` associated with it to provide an indication of the nature of the alert. The implementation may use this type to play an appropriate sound when the `Alert` is presented to the user. See `AlertType.playSound()` 138 .

An alert may contain an optional `Image`. The `Image` may be mutable or immutable. If the `Image` is mutable, the effect is as if a snapshot of its contents is taken at the time the `Alert` is constructed with this `Image` and when `setImage` is called with an `Image`. This snapshot is used whenever the contents of the `Alert` are to be displayed. Even if the application subsequently draws into the `Image`, the snapshot is not modified until the next call to `setImage`. The snapshot is *not* updated when the `Alert` becomes current or becomes visible on the display. (This is because the application does not have control over exactly when `Displayables` appear and disappear from the display.)

Activity Indicators

An alert may contain an optional `Gauge` 240 object that is used as an activity or progress indicator. By default, an `Alert` has no activity indicator; one may be set with the `setIndicator(Gauge)` 134 method. The `Gauge` object used for the activity indicator must conform to all of the following restrictions:

- it must be non-interactive;
- it must not be owned by another container (`Alert` or `Form`);
- it must not have any `Commands`;
- it must not have an `ItemCommandListener`;
- it must not have a label (that is, its label must be `null`;
- its preferred width and height must both be unlocked; and
- its layout value must be `LAYOUT_DEFAULT`.

It is an error for the application to attempt to use a `Gauge` object that violates any of these restrictions. In addition, when the `Gauge` object is being used as the indicator within an `Alert`, the application is prevented from modifying any of these pieces of the `Gauge's` state.

Commands and Listeners
Like the other `Displayable` classes, an `Alert` can accept `Commands`, which can be delivered to a `CommandListener` set by the application. The `Alert` class adds some special behavior for `Commands` and listeners.

When it is created, an `Alert` implicitly has the special `Command` `DISMISS_COMMAND`$_{130}$ present on it. If the application adds any other `Commands` to the `Alert`, `DISMISS_COMMAND` is implicitly removed. If the application removes all other `Commands`, `DISMISS_COMMAND` is implicitly restored. Attempts to add or remove `DISMISS_COMMAND` explicitly are ignored. Thus, there is always at least one `Command` present on an `Alert`.

If there are two or more `Commands` present on the `Alert`, it is automatically turned into a modal `Alert`, and the timeout value is always `FOREVER`$_{131}$. The `Alert` remains on the display until a `Command` is invoked. If the Alert has one Command (whether it is DISMISS_COMMAND or it is one provided by the application), the `Alert` may have the timed behavior as described above. When a timeout occurs, the effect is the same as if the user had invoked the `Command` explicitly.

When it is created, an `Alert` implicitly has a `CommandListener` called the *default listener* associated with it. This listener may be replaced by an application-provided listener through use of the `setCommandListener(CommandListener)`$_{134}$ method. If the application removes its listener by passing `null` to the `setCommandListener` method, the default listener is implicitly restored.

The `Display.setCurrent(Alert, Displayable)`$_{214}$ method and the `Display.setCurrent(Displayable)`$_{214}$ method (when called with an `Alert`) define special behavior for automatically advancing to another `Displayable` after the `Alert` is dismissed. This special behavior occurs only when the default listener is present on the `Alert` at the time it is dismissed or when a command is invoked. If the user invokes a `Command` and the default listener is present, the default listener ignores the `Command` and implements the automatic-advance behavior.

If the application has set its own `CommandListener`, the automatic-advance behavior is disabled. The listener code is responsible for advancing to another `Displayable`. When the application has provided a listener, `Commands` are invoked normally by passing them to the listener's `commandAction` method. The `Command` passed will be one of the `Commands` present on the `Alert`: either `DISMISS_COMMAND` or one of the application-provided `Commands`.

The application can restore the default listener by passing `null` to the `setCommandListener` method.

Note: An application may set a `Ticker`$_{345}$ with `Displayable.setTicker`$_{221}$ on an `Alert`, however it may not be displayed due to implementation restrictions.

Since: MIDP 1.0

See Also: `AlertType`$_{136}$

Member Summary	
Fields	
static Command	`DISMISS_COMMAND`$_{130}$
static int	`FOREVER`$_{131}$

Member Summary

Constructors

	`Alert(String title)`[131]
	`Alert(String title, String alertText, Image alertImage, AlertType alertType)`[131]

Methods

void	`addCommand(Command cmd)`[132]
int	`getDefaultTimeout()`[132]
Image	`getImage()`[132]
Gauge	`getIndicator()`[132]
java.lang.String	`getString()`[133]
int	`getTimeout()`[133]
AlertType	`getType()`[133]
void	`removeCommand(Command cmd)`[133]
void	`setCommandListener(CommandListener l)`[134]
void	`setImage(Image img)`[134]
void	`setIndicator(Gauge indicator)`[134]
void	`setString(String str)`[135]
void	`setTimeout(int time)`[135]
void	`setType(AlertType type)`[135]

Inherited Member Summary

Methods inherited from class `Displayable`[218]

`getHeight()`[219], `getTicker()`[219], `getTitle()`[220], `getWidth()`[220], `isShown()`[220], `setTicker(Ticker)`[221], `setTitle(String)`[221], `sizeChanged(int, int)`[222]

Methods inherited from class `Object`

`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait()`, `wait()`, `wait()`

Fields

DISMISS_COMMAND

Declaration:
`public static final javax.microedition.lcdui.Command`[175] **`DISMISS_COMMAND`**

Description:
A `Command` delivered to a listener to indicate that the `Alert` has been dismissed. This Command is implicitly present an on `Alert` whenever there are no other Commands present. The field values of `DISMISS_COMMAND` are as follows:

- label = “” (an empty string)
- type = Command.OK
- priority = 0

The label value visible to the application must be as specified above. However, the implementation may display `DISMISS_COMMAND` to the user using an implementation-specific label.

Attempting to add or remove `DISMISS_COMMAND` from an `Alert` has no effect. However, `DISMISS_COMMAND` is treated as an ordinary `Command` if it is used with other `Displayable` types.

Since: MIDP 2.0

FOREVER

Declaration:
```
public static final int FOREVER
```

Description:
`FOREVER` indicates that an `Alert` is kept visible until the user dismisses it. It is used as a value for the parameter to `setTimeout()`$_{135}$ to indicate that the alert is modal. Instead of waiting for a specified period of time, a modal `Alert` will wait for the user to take some explicit action, such as pressing a button, before proceeding to the next `Displayable`.

Value `-2` is assigned to `FOREVER`.

Constructors

Alert(String)

Declaration:
```
public Alert(String title)
```

Description:
Constructs a new, empty `Alert` object with the given title. If `null` is passed, the `Alert` will have no title. Calling this constructor is equivalent to calling

```
Alert(title, null, null, null)
```

Parameters:
`title` - the title string, or `null`

See Also: `Alert(String, String, Image, AlertType)`$_{131}$

Alert(String, String, Image, AlertType)

Declaration:
```
public Alert(String title, String alertText,
             javax.microedition.lcdui.Image270 alertImage,
             javax.microedition.lcdui.AlertType136 alertType)
```

Description:
Constructs a new `Alert` object with the given title, content string and image, and alert type. The layout of the contents is implementation dependent. The timeout value of this new alert is the same value that is returned by `getDefaultTimeout()`. The `Image` provided may either be mutable or immutable. The handling and behavior of specific `AlertTypes` is described in `AlertType`$_{136}$. `null` is allowed as the value of the `alertType` parameter and indicates that the `Alert` is not to have a specific alert type. `DISMISS_COMMAND` is the only `Command` present on the new `Alert`. The `CommandListener` associated with the new `Alert` is the *default listener*. Its behavior is described in more detail in the section Commands and Listeners.

Parameters:

`title` - the title string, or `null` if there is no title

`alertText` - the string contents, or `null` if there is no string

`alertImage` - the image contents, or `null` if there is no image

`alertType` - the type of the `Alert`, or `null` if the `Alert` has no specific type

Methods

addCommand(Command)

Declaration:
`public void` **`addCommand`**`(javax.microedition.lcdui.Command`[175] `cmd)`

Description:
Similar to `Displayable.addCommand(Command)`[219] , however when the application first adds a command to an `Alert`, `DISMISS_COMMAND`[130] is implicitly removed. Calling this method with `DISMISS_COMMAND` as the parameter has no effect.

Overrides: `addCommand`[219] in class `Displayable`[218]

Parameters:
`cmd` - the command to be added

Throws:
`NullPointerException` - if cmd is `null`

getDefaultTimeout()

Declaration:
`public int` **`getDefaultTimeout`**`()`

Description:
Gets the default time for showing an `Alert`. This is either a positive value, which indicates a time in milliseconds, or the special value `FOREVER`[131] , which indicates that `Alerts` are modal by default. The value returned will vary across implementations and is presumably tailored to be suitable for each.

Returns: default timeout in milliseconds, or `FOREVER`

getImage()

Declaration:
`public javax.microedition.lcdui.Image`[270] **`getImage`**`()`

Description:
Gets the `Image` used in the `Alert`.

Returns: the `Alert's` image, or `null` if there is no image

See Also: `setImage(Image)`[134]

getIndicator()

Declaration:
`public javax.microedition.lcdui.Gauge`[240] **`getIndicator`**`()`

Description:
Gets the activity indicator for this `Alert`.

Returns: a reference to this `Alert's` activity indicator, or `null` if there is none

Since: MIDP 2.0

See Also: `setIndicator(Gauge)`[134]

getString()

Declaration:
`public String getString()`

Description:
Gets the text string used in the `Alert`.

Returns: the `Alert's` text string, or `null` if there is no text

See Also: `setString(String)`[135]

getTimeout()

Declaration:
`public int getTimeout()`

Description:
Gets the time this `Alert` will be shown. This is either a positive value, which indicates a time in milliseconds, or the special value `FOREVER`, which indicates that this `Alert` is modal. This value is not necessarily the same value that might have been set by the application in a call to `setTimeout(int)`[135]. In particular, if the `Alert` is made modal because its contents is large enough to scroll, the value returned by `getTimeout` will be `FOREVER`.

Returns: timeout in milliseconds, or `FOREVER`

See Also: `setTimeout(int)`[135]

getType()

Declaration:
`public javax.microedition.lcdui.AlertType`[136] `getType()`

Description:
Gets the type of the `Alert`.

Returns: a reference to an instance of `AlertType`, or `null` if the `Alert` has no specific type

See Also: `setType(AlertType)`[135]

removeCommand(Command)

Declaration:
`public void removeCommand(javax.microedition.lcdui.Command`[175] `cmd)`

Description:
Similar to `Displayable.removeCommand(Command)`[220], however when the application removes the last command from an `Alert`, `DISMISS_COMMAND`[130] is implicitly added. Calling this method with `DISMISS_COMMAND` as the parameter has no effect.

Overrides: `removeCommand`[220] in class `Displayable`[218]

Parameters:
`cmd` - the command to be removed

setCommandListener(CommandListener)

Declaration:
```
public void setCommandListener(javax.microedition.lcdui.CommandListener l)
```

Description:
The same as `Displayable.setCommandListener(CommandListener)` but with the following additional semantics. If the listener parameter is `null`, the *default listener* is restored. See Commands and Listeners for the definition of the behavior of the default listener.

Overrides: `setCommandListener` in class `Displayable`

Parameters:
`l` - the new listener, or `null`

setImage(Image)

Declaration:
```
public void setImage(javax.microedition.lcdui.Image img)
```

Description:
Sets the `Image` used in the `Alert`. The `Image` may be mutable or immutable. If `img` is `null`, specifies that this `Alert` has no image. If `img` is mutable, the effect is as if a snapshot is taken of `img's` contents immediately prior to the call to `setImage`. This snapshot is used whenever the contents of the `Alert` are to be displayed. If `img` is already the `Image` of this `Alert`, the effect is as if a new snapshot of img's contents is taken. Thus, after painting into a mutable image contained by an `Alert`, the application can call

```
alert.setImage(alert.getImage());
```

to refresh the `Alert's` snapshot of its `Image`.

If the `Alert` is visible on the display when its contents are updated through a call to `setImage`, the display will be updated with the new snapshot as soon as it is feasible for the implementation to do so.

Parameters:
`img` - the `Alert's` image, or `null` if there is no image

See Also: `getImage()`

setIndicator(Gauge)

Declaration:
```
public void setIndicator(javax.microedition.lcdui.Gauge indicator)
```

Description:
Sets an activity indicator on this `Alert`. The activity indicator is a `Gauge` object. It must be in a restricted state in order for it to be used as the activity indicator for an `Alert`. The restrictions are listed above. If the `Gauge` object violates any of these restrictions, `IllegalArgumentException` is thrown.

If `indicator` is `null`, this removes any activity indicator present on this `Alert`.

Parameters:
`indicator` - the activity indicator for this `Alert`, or `null` if there is to be none

Throws:
`IllegalArgumentException` - if `indicator` does not meet the restrictions for its use in an `Alert`

Since: MIDP 2.0

See Also: `getIndicator()`$_{132}$

setString(String)

Declaration:
```
public void setString(String str)
```

Description:
Sets the text string used in the `Alert`.

If the `Alert` is visible on the display when its contents are updated through a call to `setString`, the display will be updated with the new contents as soon as it is feasible for the implementation to do so.

Parameters:
`str` - the `Alert's` text string, or `null` if there is no text

See Also: `getString()`$_{133}$

setTimeout(int)

Declaration:
```
public void setTimeout(int time)
```

Description:
Set the time for which the `Alert` is to be shown. This must either be a positive time value in milliseconds, or the special value `FOREVER`.

Parameters:
`time` - timeout in milliseconds, or `FOREVER`

Throws:
`IllegalArgumentException` - if time is not positive and is not `FOREVER`

See Also: `getTimeout()`$_{133}$

setType(AlertType)

Declaration:
```
public void setType(javax.microedition.lcdui.AlertType type)
```
(`javax.microedition.lcdui.AlertType` $_{136}$)

Description:
Sets the type of the `Alert`. The handling and behavior of specific `AlertTypes` is described in `AlertType`$_{136}$.

Parameters:
`type` - an `AlertType`, or `null` if the `Alert` has no specific type

See Also: `getType()`$_{133}$

javax.microedition.lcdui
AlertType

Declaration

```
public class AlertType

Object
  |
  +--javax.microedition.lcdui.AlertType
```

Description

The `AlertType` provides an indication of the nature of alerts. `Alerts` are used by an application to present various kinds of information to the user. An `AlertType` may be used to directly signal the user without changing the current `Displayable`. The `playSound` method can be used to spontaneously generate a sound to alert the user. For example, a game using a `Canvas` can use `playSound` to indicate success or progress. The predefined types are `INFO`, `WARNING`, `ERROR`, `ALARM`, and `CONFIRMATION`.

Since: MIDP 1.0

See Also: `Alert`[128]

Member Summary

Fields

static AlertType	ALARM[137]
static AlertType	CONFIRMATION[137]
static AlertType	ERROR[137]
static AlertType	INFO[137]
static AlertType	WARNING[137]

Constructors

protected	AlertType()[137]

Methods

boolean	playSound(Display display)[138]

Inherited Member Summary

Methods inherited from class `Object`

`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Fields

ALARM

Declaration:
`public static final javax.microedition.lcdui.AlertType`$_{136}$ **`ALARM`**

Description:
An `ALARM` `AlertType` is a hint to alert the user to an event for which the user has previously requested to be notified. For example, the message might say, “Staff meeting in five minutes.”

CONFIRMATION

Declaration:
`public static final javax.microedition.lcdui.AlertType`$_{136}$ **`CONFIRMATION`**

Description:
A `CONFIRMATION` `AlertType` is a hint to confirm user actions. For example, “Saved!” might be shown to indicate that a Save operation has completed.

ERROR

Declaration:
`public static final javax.microedition.lcdui.AlertType`$_{136}$ **`ERROR`**

Description:
An `ERROR` `AlertType` is a hint to alert the user to an erroneous operation. For example, an error alert might show the message, “There is not enough room to install the application.”

INFO

Declaration:
`public static final javax.microedition.lcdui.AlertType`$_{136}$ **`INFO`**

Description:
An `INFO` `AlertType` typically provides non-threatening information to the user. For example, a simple splash screen might be an `INFO` `AlertType`.

WARNING

Declaration:
`public static final javax.microedition.lcdui.AlertType`$_{136}$ **`WARNING`**

Description:
A `WARNING` `AlertType` is a hint to warn the user of a potentially dangerous operation. For example, the warning message may contain the message, “Warning: this operation will erase your data.”

Constructors

AlertType()

Declaration:
`protected` **`AlertType`**`()`

Description:
Protected constructor for subclasses.

Methods

playSound(Display)

Declaration:
```
public boolean playSound(javax.microedition.lcdui.Display display)
```

Description:
Alert the user by playing the sound for this `AlertType`. The `AlertType` instance is used as a hint by the device to generate an appropriate sound. Instances other than those predefined above may be ignored. The actual sound made by the device, if any, is determined by the device. The device may ignore the request, use the same sound for several `AlertTypes` or use any other means suitable to alert the user.

Parameters:
`display` - to which the `AlertType's` sound should be played.

Returns: `true` if the user was alerted, `false` otherwise.

Throws:
`NullPointerException` - if `display` is `null`

javax.microedition.lcdui
Canvas

Declaration

```
public abstract class Canvas extends Displayable218

Object
  |
  +--javax.microedition.lcdui.Displayable218
        |
        +--javax.microedition.lcdui.Canvas
```

Direct Known Subclasses: `javax.microedition.lcdui.game.GameCanvas`$_{349}$

Description

The `Canvas` class is a base class for writing applications that need to handle low-level events and to issue graphics calls for drawing to the display. Game applications will likely make heavy use of the `Canvas` class. From an application development perspective, the `Canvas` class is interchangeable with standard `Screen` classes, so an application may mix and match `Canvas` with high-level screens as needed. For example, a List screen may be used to select the track for a racing game, and a `Canvas` subclass would implement the actual game.

The `Canvas` provides the developer with methods to handle game actions, key events, and pointer events (if supported by the device). Methods are also provided to identify the device's capabilities and mapping of keys to game actions. The key events are reported with respect to *key codes*, which are directly bound to concrete keys on the device, use of which may hinder portability. Portable applications should use game actions instead of key codes.

Like other subclasses of `Displayable`, the `Canvas` class allows the application to register a listener for commands. Unlike other `Displayables`, however, the `Canvas` class requires applications to subclass it in order to use it. The `paint()` method is declared `abstract`, and so the application *must* provide an implementation in its subclass. Other event-reporting methods are not declared `abstract`, and their default implementations are empty (that is, they do nothing). This allows the application to override only the methods that report events in which the application has interest.

This is in contrast to the `Screen`$_{315}$ classes, which allow the application to define listeners and to register them with instances of the `Screen` classes. This style is not used for the `Canvas` class, because several new listener interfaces would need to be created, one for each kind of event that might be delivered. An alternative would be to have fewer listener interfaces, but this would require listeners to filter out events in which they had no interest.

Key Events

Applications receive keystroke events in which the individual keys are named within a space of *key codes*. Every key for which events are reported to MIDP applications is assigned a key code. The key code values are unique for each hardware key unless two keys are obvious synonyms for each other. MIDP defines the following key codes: `KEY_NUM0`$_{144}$, `KEY_NUM1`$_{145}$, `KEY_NUM2`$_{145}$, `KEY_NUM3`$_{145}$, `KEY_NUM4`$_{145}$, `KEY_NUM5`$_{145}$, `KEY_NUM6`$_{145}$, `KEY_NUM7`$_{146}$, `KEY_NUM8`$_{146}$, `KEY_NUM9`$_{146}$, `KEY_STAR`$_{146}$, and `KEY_POUND`$_{146}$. (These key codes correspond to keys on a ITU-T standard telephone keypad.) Other keys may be present on the keyboard, and they will generally have key codes distinct from those list above. In order to guarantee portability, applications should use only the standard key codes.

The standard key codes' values are equal to the Unicode encoding for the character that represents the key. If the device includes any other keys that have an obvious correspondence to a Unicode character, their key code values should equal the Unicode encoding for that character. For keys that have no corresponding Unicode character, the implementation must use negative values. Zero is defined to be an invalid key code. It is thus possible for an application to convert a keyCode into a Unicode character using the following code:

```
if (keyCode > 0) {
    char ch = (char)keyCode;
    // ...
}
```

This technique is useful only in certain limited cases. In particular, it is not sufficient for full textual input, because it does not handle upper and lower case, keyboard shift states, and characters that require more than one keystroke to enter. For textual input, applications should always use `TextBox`$_{323}$ or `TextField`$_{330}$ objects.

It is sometimes useful to find the *name* of a key in order to display a message about this key. In this case the application may use the `getKeyName()`$_{148}$ method to find a key's name.

Game Actions

Portable applications that need arrow key events and gaming-related events should use *game actions* in preference to key codes and key names. MIDP defines the following game actions: `UP`$_{147}$, `DOWN`$_{143}$, `LEFT`$_{146}$, `RIGHT`$_{147}$, `FIRE`$_{144}$, `GAME_A`$_{144}$, `GAME_B`$_{144}$, `GAME_C`$_{144}$, and `GAME_D`$_{144}$.

Each key code may be mapped to at most one game action. However, a game action may be associated with more than one key code. The application can translate a key code into a game action using the `getGameAction(int keyCode)`$_{147}$ method, and it can translate a game action into a key code using the `getKeyCode(int gameAction)`$_{148}$ method. There may be multiple keycodes associated with a particular game action, but `getKeyCode` returns only one of them. Supposing that `g` is a valid game action and `k` is a valid key code for a key associated with a game action, consider the following expressions:

```
g == getGameAction(getKeyCode(g))    // (1)
k == getKeyCode(getGameAction(k))    // (2)
```

Expression (1) is *always* true. However, expression (2) might be true but is *not necessarily* true.

The implementation is not allowed to change the mapping of game actions and key codes during execution of the application.

Portable applications that are interested in using game actions should translate every key event into a game action by calling the `getGameAction()`$_{147}$ method and then testing the result. For example, on some devices the game actions `UP`, `DOWN`, `LEFT` and `RIGHT` may be mapped to 4-way navigation arrow keys. In this case, `getKeyCode(UP)` would return a device-dependent code for the up-arrow key. On other devices, a possible mapping would be on the number keys `2`, `4`, `6` and `8`. In this case, `getKeyCode(UP)` would return `KEY_NUM2`. In both cases, the `getGameAction()` method would return the `LEFT` game action when the user presses the key that is a "natural left" on her device.

Commands

It is also possible for the user to issue `commands`$_{175}$ when a canvas is current. `Commands` are mapped to keys and menus in a device-specific fashion. For some devices the keys used for commands may overlap with the keys that will deliver key code events to the canvas. If this is the case, the device will provide a means transparent to the application that enables the user to select a mode that determines whether these keys will deliver commands or key code events to the application. When the `Canvas` is in normal mode (see below), the set of key code events available to a canvas will not change depending upon the number of commands present or the presence of a command listener. When the `Canvas` is in full-screen mode, if there is no command listener present, the device may choose to deliver key code events for keys that would otherwise be reserved for delivery of commands. Game developers should be aware that access to commands will vary greatly across devices, and that requiring the user to issue commands during game play may have a great impact on the ease with which the game can be played.

Event Delivery

The `Canvas` object defines several methods that are called by the implementation. These methods are primarily for the purpose of delivering events to the application, and so they are referred to as *event delivery* methods. The set of methods is:

- `showNotify()`
- `hideNotify()`
- `keyPressed()`
- `keyRepeated()`
- `keyReleased()`
- `pointerPressed()`
- `pointerDragged()`
- `pointerReleased()`
- `paint()`

These methods are all called serially. That is, the implementation will never call an event delivery method before a prior call to *any* of the event delivery methods has returned. The `serviceRepaints()` method is an exception to this rule, as it blocks until `paint()` is called and returns. This will occur even if the application is in the midst of one of the event delivery methods when it calls `serviceRepaints()`.

The `Display.callSerially()`$_{210}$ method can be used to serialize some application-defined work with the event stream. For further information, see the Event Handling and Concurrency sections of the package summary.

The key-related, pointer-related, and `paint()` methods will only be called while the `Canvas` is actually visible on the output device. These methods will therefore only be called on this `Canvas` object only after a call to `showNotify()` and before a call to `hideNotify()`. After `hideNotify()` has been called, none of the key, pointer, and `paint` methods will be called until after a subsequent call to `showNotify()` has returned. A call to a `run()` method resulting from `callSerially()` may occur irrespective of calls to `showNotify()` and `hideNotify()`.

The `showNotify()`$_{154}$ method is called prior to the `Canvas` actually being made visible on the display, and the `hideNotify()`$_{149}$ method is called after the `Canvas` has been removed from the display. The visibility state of a `Canvas` (or any other `Displayable` object) may be queried through the use of the `Displayable.isShown()`$_{220}$ method. The change in visibility state of a `Canvas` may be caused by the application management software moving `MIDlets` between foreground and background states, or by the system obscuring the `Canvas` with system screens. Thus, the calls to `showNotify()` and `hideNotify()`

are not under the control of the `MIDlet` and may occur fairly frequently. Application developers are encouraged to perform expensive setup and teardown tasks outside the `showNotify()` and `hideNotify()` methods in order to make them as lightweight as possible.

A `Canvas` can be in normal mode or in full-screen mode. In normal mode, space on the display may be occupied by command labels, a title, and a ticker. By setting a `Canvas` into full-screen mode, the application is requesting that the `Canvas` occupy as much of the display space as is possible. In full-screen mode, the title and ticker are not displayed even if they are present on the `Canvas`, and `Commands` may be presented using some alternative means (such as through a pop-up menu). Note that the implementation may still consume a portion of the display for things like status indicators, even if the displayed `Canvas` is in full-screen mode. In full-screen mode, although the title is not displayed, its text may still be used for other purposes, such as for the title of a pop-up menu of `Commands`.

`Canvas` objects are in normal mode by default. The normal vs. full-screen mode setting is controlled through the use of the `setFullScreenMode(boolean)`[153] method.

Calling `setFullScreenMode(boolean)`[153] may result in `sizeChanged()`[154] being called. The default implementation of this method does nothing. The application can override this method to handle changes in size of available drawing area.

Note: As mentioned in the "Specification Requirements" section of the overview, implementations must provide the user with an indication of network usage. If the indicator is rendered on screen, it must be visible when network activity occurs, even when the `Canvas` is in full-screen mode.

Since: MIDP 1.0

Member Summary

Fields

static int	DOWN[143]
static int	FIRE[144]
static int	GAME_A[144]
static int	GAME_B[144]
static int	GAME_C[144]
static int	GAME_D[144]
static int	KEY_NUM0[144]
static int	KEY_NUM1[145]
static int	KEY_NUM2[145]
static int	KEY_NUM3[145]
static int	KEY_NUM4[145]
static int	KEY_NUM5[145]
static int	KEY_NUM6[145]
static int	KEY_NUM7[146]
static int	KEY_NUM8[146]
static int	KEY_NUM9[146]
static int	KEY_POUND[146]
static int	KEY_STAR[146]
static int	LEFT[146]
static int	RIGHT[147]
static int	UP[147]

Constructors

protected	Canvas()[147]

Member Summary

Methods

int	getGameAction(int keyCode)[147]
int	getHeight()[148]
int	getKeyCode(int gameAction)[148]
java.lang.String	getKeyName(int keyCode)[148]
int	getWidth()[149]
boolean	hasPointerEvents()[149]
boolean	hasPointerMotionEvents()[149]
boolean	hasRepeatEvents()[149]
protected void	hideNotify()[149]
boolean	isDoubleBuffered()[150]
protected void	keyPressed(int keyCode)[150]
protected void	keyReleased(int keyCode)[150]
protected void	keyRepeated(int keyCode)[150]
protected abstract void	paint(Graphics g)[151]
protected void	pointerDragged(int x, int y)[152]
protected void	pointerPressed(int x, int y)[152]
protected void	pointerReleased(int x, int y)[152]
void	repaint()[152]
void	repaint(int x, int y, int width, int height)[153]
void	serviceRepaints()[153]
void	setFullScreenMode(boolean mode)[153]
protected void	showNotify()[154]
protected void	sizeChanged(int w, int h)[154]

Inherited Member Summary

Methods inherited from class `Displayable`[218]

addCommand(Command)[219], getTicker()[219], getTitle()[220], isShown()[220], removeCommand(Command)[220], setCommandListener(CommandListener)[221], setTicker(Ticker)[221], setTitle(String)[221]

Methods inherited from class `Object`

equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Fields

DOWN

Declaration:
`public static final int DOWN`

Description:
Constant for the DOWN game action.

Constant value 6 is set to DOWN.

FIRE

Declaration:
```
public static final int FIRE
```

Description:
Constant for the FIRE game action.

Constant value 8 is set to FIRE.

GAME_A

Declaration:
```
public static final int GAME_A
```

Description:
Constant for the general purpose “A” game action.

Constant value 9 is set to GAME_A.

GAME_B

Declaration:
```
public static final int GAME_B
```

Description:
Constant for the general purpose “B” game action.

Constant value 10 is set to GAME_B.

GAME_C

Declaration:
```
public static final int GAME_C
```

Description:
Constant for the general purpose “C” game action.

Constant value 11 is set to GAME_C.

GAME_D

Declaration:
```
public static final int GAME_D
```

Description:
Constant for the general purpose “D” game action.

Constant value 12 is set to GAME_D.

KEY_NUM0

Declaration:
```
public static final int KEY_NUM0
```

Description:
keyCode for ITU-T key 0.

Constant value 48 is set to KEY_NUM0.

KEY_NUM1

Declaration:
`public static final int KEY_NUM1`

Description:
keyCode for ITU-T key 1.

Constant value 49 is set to KEY_NUM1.

KEY_NUM2

Declaration:
`public static final int KEY_NUM2`

Description:
keyCode for ITU-T key 2.

Constant value 50 is set to KEY_NUM2.

KEY_NUM3

Declaration:
`public static final int KEY_NUM3`

Description:
keyCode for ITU-T key 3.

Constant value 51 is set to KEY_NUM3.

KEY_NUM4

Declaration:
`public static final int KEY_NUM4`

Description:
keyCode for ITU-T key 4.

Constant value 52 is set to KEY_NUM4.

KEY_NUM5

Declaration:
`public static final int KEY_NUM5`

Description:
keyCode for ITU-T key 5.

Constant value 53 is set to KEY_NUM5.

KEY_NUM6

Declaration:
`public static final int KEY_NUM6`

Description:
keyCode for ITU-T key 6.

Constant value 54 is set to KEY_NUM6.

KEY_NUM7

Declaration:
```
public static final int KEY_NUM7
```

Description:
keyCode for ITU-T key 7.

Constant value 55 is set to KEY_NUM7.

KEY_NUM8

Declaration:
```
public static final int KEY_NUM8
```

Description:
keyCode for ITU-T key 8.

Constant value 56 is set to KEY_NUM8.

KEY_NUM9

Declaration:
```
public static final int KEY_NUM9
```

Description:
keyCode for ITU-T key 9.

Constant value 57 is set to KEY_NUM09.

KEY_POUND

Declaration:
```
public static final int KEY_POUND
```

Description:
keyCode for ITU-T key “pound” (#).

Constant value 35 is set to KEY_POUND.

KEY_STAR

Declaration:
```
public static final int KEY_STAR
```

Description:
keyCode for ITU-T key “star” (*).

Constant value 42 is set to KEY_STAR.

LEFT

Declaration:
```
public static final int LEFT
```

Description:
Constant for the LEFT game action.

Constant value 2 is set to LEFT.

RIGHT

Declaration:
```
public static final int RIGHT
```

Description:
Constant for the RIGHT game action.

Constant value 5 is set to RIGHT.

UP

Declaration:
```
public static final int UP
```

Description:
Constant for the UP game action.

Constant value 1 is set to UP.

Constructors

Canvas()

Declaration:
```
protected Canvas()
```

Description:
Constructs a new Canvas object.

Methods

getGameAction(int)

Declaration:
```
public int getGameAction(int keyCode)
```

Description:
Gets the game action associated with the given key code of the device. Returns zero if no game action is associated with this key code. See above for further discussion of game actions.

The mapping between key codes and game actions will not change during the execution of the application.

Parameters:
keyCode - the key code

Returns: the game action corresponding to this key, or 0 if none

Throws:
IllegalArgumentException - if keyCode is not a valid key code

getHeight()

Declaration:
```
public int getHeight()
```

Description:
Gets the height in pixels of the displayable area of the `Canvas`. The value returned may change during execution. If it does, the application will be notified through a call to the `sizeChanged(int, int)`[154] method.

Overrides: `getHeight`[219] in class `Displayable`[218]

Returns: height of the displayable area

getKeyCode(int)

Declaration:
```
public int getKeyCode(int gameAction)
```

Description:
Gets a key code that corresponds to the specified game action on the device. The implementation is required to provide a mapping for every game action, so this method will always return a valid key code for every game action. See above for further discussion of game actions. There may be multiple keys associated with the same game action; however, this method will return only one of them. Applications should translate the key code of every key event into a game action using `getGameAction(int)`[147] and then interpret the resulting game action, instead of generating a table of key codes at using this method during initialization.

The mapping between key codes and game actions will not change during the execution of the application.

Parameters:
`gameAction` - the game action

Returns: a key code corresponding to this game action

Throws:
`IllegalArgumentException` - if `gameAction` is not a valid game action

getKeyName(int)

Declaration:
```
public String getKeyName(int keyCode)
```

Description:
Gets an informative key string for a key. The string returned will resemble the text physically printed on the key. This string is suitable for displaying to the user. For example, on a device with function keys `F1` through `F4`, calling this method on the `keyCode` for the `F1` key will return the string "`F1`". A typical use for this string will be to compose help text such as "Press `F1` to proceed."

This method will return a non-empty string for every valid key code.

There is no direct mapping from game actions to key names. To get the string name for a game action `GAME_A`, the application must call

```
getKeyName(getKeyCode(GAME_A));
```

Parameters:
`keyCode` - the key code being requested

Returns: a string name for the key

Throws:
`IllegalArgumentException` - if `keyCode` is not a valid key code

getWidth()

Declaration:
```
public int getWidth()
```

Description:
Gets the width in pixels of the displayable area of the `Canvas`. The value returned may change during execution. If it does, the application will be notified through a call to the `sizeChanged(int, int)`[154] method.

Overrides: `getWidth`[220] in class `Displayable`[218]

Returns: width of the displayable area

hasPointerEvents()

Declaration:
```
public boolean hasPointerEvents()
```

Description:
Checks if the platform supports pointer press and release events.

Returns: `true` if the device supports pointer events

hasPointerMotionEvents()

Declaration:
```
public boolean hasPointerMotionEvents()
```

Description:
Checks if the platform supports pointer motion events (pointer dragged). Applications may use this method to determine if the platform is capable of supporting motion events.

Returns: `true` if the device supports pointer motion events

hasRepeatEvents()

Declaration:
```
public boolean hasRepeatEvents()
```

Description:
Checks if the platform can generate repeat events when key is kept down.

Returns: `true` if the device supports repeat events

hideNotify()

Declaration:
```
protected void hideNotify()
```

Description:
The implementation calls `hideNotify()` shortly after the `Canvas` has been removed from the display. `Canvas` subclasses may override this method in order to pause animations, revoke timers, etc. The default implementation of this method in class `Canvas` is empty.

isDoubleBuffered()

Declaration:
```
public boolean isDoubleBuffered()
```

Description:
Checks if the `Canvas` is double buffered by the implementation.

Returns: `true` if double buffered, `false` otherwise

keyPressed(int)

Declaration:
```
protected void keyPressed(int keyCode)
```

Description:
Called when a key is pressed.

The `getGameAction()` method can be called to determine what game action, if any, is mapped to the key. Class `Canvas` has an empty implementation of this method, and the subclass has to redefine it if it wants to listen this method.

Parameters:
`keyCode` - the key code of the key that was pressed

keyReleased(int)

Declaration:
```
protected void keyReleased(int keyCode)
```

Description:
Called when a key is released.

The `getGameAction()` method can be called to determine what game action, if any, is mapped to the key. Class `Canvas` has an empty implementation of this method, and the subclass has to redefine it if it wants to listen this method.

Parameters:
`keyCode` - the key code of the key that was released

keyRepeated(int)

Declaration:
```
protected void keyRepeated(int keyCode)
```

Description:
Called when a key is repeated (held down).

The `getGameAction()` method can be called to determine what game action, if any, is mapped to the key. Class `Canvas` has an empty implementation of this method, and the subclass has to redefine it if it wants to listen this method.

Parameters:
`keyCode` - the key code of the key that was repeated

See Also: `hasRepeatEvents()`[149]

paint(Graphics)

Declaration:

```
protected abstract void paint(javax.microedition.lcdui.Graphics[247] g)
```

Description:

Renders the `Canvas`. The application must implement this method in order to paint any graphics.

The `Graphics` object's clip region defines the area of the screen that is considered to be invalid. A correctly-written `paint()` routine must paint *every* pixel within this region. This is necessary because the implementation is not required to clear the region prior to calling `paint()` on it. Thus, failing to paint every pixel may result in a portion of the previous screen image remaining visible.

Applications *must not* assume that they know the underlying source of the `paint()` call and use this assumption to paint only a subset of the pixels within the clip region. The reason is that this particular `paint()` call may have resulted from multiple `repaint()` requests, some of which may have been generated from outside the application. An application that paints only what it thinks is necessary to be painted may display incorrectly if the screen contents had been invalidated by, for example, an incoming telephone call.

Operations on this graphics object after the `paint()` call returns are undefined. Thus, the application *must not* cache this `Graphics` object for later use or use by another thread. It must only be used within the scope of this method.

The implementation may postpone visible effects of graphics operations until the end of the paint method.

The contents of the `Canvas` are never saved if it is hidden and then is made visible again. Thus, shortly after `showNotify()` is called, `paint()` will always be called with a `Graphics` object whose clip region specifies the entire displayable area of the `Canvas`. Applications *must not* rely on any contents being preserved from a previous occasion when the `Canvas` was current. This call to `paint()` will not necessarily occur before any other key or pointer methods are called on the `Canvas`. Applications whose repaint recomputation is expensive may create an offscreen `Image`, paint into it, and then draw this image on the `Canvas` when `paint()` is called.

The application code must never call `paint()`; it is called only by the implementation.

The `Graphics` object passed to the `paint()` method has the following properties:

- the destination is the actual display, or if double buffering is in effect, a back buffer for the display;
- the clip region includes at least one pixel within this `Canvas`;
- the current color is black;
- the font is the same as the font returned by `Font.getDefaultFont()`[227];
- the stroke style is `SOLID`[254];
- the origin of the coordinate system is located at the upper-left corner of the `Canvas`; and
- the `Canvas` is visible, that is, a call to `isShown()` will return `true`.

Parameters:

`g` - the `Graphics` object to be used for rendering the `Canvas`

pointerDragged(int, int)

Declaration:
```
protected void pointerDragged(int x, int y)
```

Description:
Called when the pointer is dragged.

The `hasPointerMotionEvents()`[149] method may be called to determine if the device supports pointer events. Class `Canvas` has an empty implementation of this method, and the subclass has to redefine it if it wants to listen this method.

Parameters:
`x` - the horizontal location where the pointer was dragged (relative to the `Canvas`)

`y` - the vertical location where the pointer was dragged (relative to the `Canvas`)

pointerPressed(int, int)

Declaration:
```
protected void pointerPressed(int x, int y)
```

Description:
Called when the pointer is pressed.

The `hasPointerEvents()`[149] method may be called to determine if the device supports pointer events. Class `Canvas` has an empty implementation of this method, and the subclass has to redefine it if it wants to listen this method.

Parameters:
`x` - the horizontal location where the pointer was pressed (relative to the `Canvas`)

`y` - the vertical location where the pointer was pressed (relative to the `Canvas`)

pointerReleased(int, int)

Declaration:
```
protected void pointerReleased(int x, int y)
```

Description:
Called when the pointer is released.

The `hasPointerEvents()`[149] method may be called to determine if the device supports pointer events. Class `Canvas` has an empty implementation of this method, and the subclass has to redefine it if it wants to listen this method.

Parameters:
`x` - the horizontal location where the pointer was released (relative to the `Canvas`)

`y` - the vertical location where the pointer was released (relative to the `Canvas`)

repaint()

Declaration:
```
public final void repaint()
```

Description:
Requests a repaint for the entire `Canvas`. The effect is identical to

```
repaint(0, 0, getWidth(), getHeight());
```

repaint(int, int, int, int)

Declaration:
```
public final void repaint(int x, int y, int width, int height)
```

Description:
Requests a repaint for the specified region of the `Canvas`. Calling this method may result in subsequent call to `paint()`, where the passed `Graphics` object's clip region will include at least the specified region.

If the canvas is not visible, or if width and height are zero or less, or if the rectangle does not specify a visible region of the display, this call has no effect.

The call to `paint()` occurs asynchronously of the call to `repaint()`. That is, `repaint()` will not block waiting for `paint()` to finish. The `paint()` method will either be called after the caller of `repaint()` returns to the implementation (if the caller is a callback) or on another thread entirely.

To synchronize with its `paint()` routine, applications can use either `Display.callSerially()` 210 or `serviceRepaints()` 153, or they can code explicit synchronization into their `paint()` routine.

The origin of the coordinate system is above and to the left of the pixel in the upper left corner of the displayable area of the `Canvas`. The X-coordinate is positive right and the Y-coordinate is positive downwards.

Parameters:

`x` - the x coordinate of the rectangle to be repainted

`y` - the y coordinate of the rectangle to be repainted

`width` - the width of the rectangle to be repainted

`height` - the height of the rectangle to be repainted

See Also: `Display.callSerially(Runnable)` 210, `serviceRepaints()` 153

serviceRepaints()

Declaration:
```
public final void serviceRepaints()
```

Description:
Forces any pending repaint requests to be serviced immediately. This method blocks until the pending requests have been serviced. If there are no pending repaints, or if this canvas is not visible on the display, this call does nothing and returns immediately.

Warning: This method blocks until the call to the application's `paint()` method returns. The application has no control over which thread calls `paint()`; it may vary from implementation to implementation. If the caller of `serviceRepaints()` holds a lock that the `paint()` method acquires, this may result in deadlock. Therefore, callers of `serviceRepaints()` *must not* hold any locks that might be acquired within the `paint()` method. The `Display.callSerially()` 210 method provides a facility where an application can be called back after painting has completed, avoiding the danger of deadlock.

See Also: `Display.callSerially(Runnable)` 210

setFullScreenMode(boolean)

Declaration:
```
public void setFullScreenMode(boolean mode)
```

Description:
Controls whether the `Canvas` is in full-screen mode or in normal mode.

Parameters:
`mode` - `true` if the `Canvas` is to be in full screen mode, `false` otherwise

Since: MIDP 2.0

showNotify()

Declaration:
```
protected void showNotify()
```

Description:
The implementation calls `showNotify()` immediately prior to this `Canvas` being made visible on the display. Canvas subclasses may override this method to perform tasks before being shown, such as setting up animations, starting timers, etc. The default implementation of this method in class `Canvas` is empty.

sizeChanged(int, int)

Declaration:
```
protected void sizeChanged(int w, int h)
```

Description:
Called when the drawable area of the `Canvas` has been changed. This method has augmented semantics compared to `Displayable.sizeChanged`[222].

In addition to the causes listed in `Displayable.sizeChanged`, a size change can occur on a `Canvas` because of a change between normal and full-screen modes.

If the size of a `Canvas` changes while it is actually visible on the display, it may trigger an automatic repaint request. If this occurs, the call to `sizeChanged` will occur prior to the call to `paint`. If the `Canvas` has become smaller, the implementation may choose not to trigger a repaint request if the remaining contents of the `Canvas` have been preserved. Similarly, if the `Canvas` has become larger, the implementation may choose to trigger a repaint only for the new region. In both cases, the preserved contents must remain stationary with respect to the origin of the `Canvas`. If the size change is significant to the contents of the `Canvas`, the application must explicitly issue a repaint request for the changed areas. Note that the application's repaint request should not cause multiple repaints, since it can be coalesced with repaint requests that are already pending.

If the size of a `Canvas` changes while it is not visible, the implementation may choose to delay calls to `sizeChanged` until immediately prior to the call to `showNotify`. In that case, there will be only one call to `sizeChanged`, regardless of the number of size changes.

An application that is sensitive to size changes can update instance variables in its implementation of `sizeChanged`. These updated values will be available to the code in the `showNotify`, `hideNotify`, and `paint` methods.

Overrides: `sizeChanged`[222] in class `Displayable`[218]

Parameters:
`w` - the new width in pixels of the drawable area of the `Canvas`

`h` - the new height in pixels of the drawable area of the `Canvas`

Since: MIDP 2.0

javax.microedition.lcdui
Choice

Declaration
```
public interface Choice
```

All Known Implementing Classes: ChoiceGroup[166], List[303]

Description
Choice defines an API for a user interface components implementing selection from predefined number of choices. Such UI components are List[303] and ChoiceGroup[166]. The contents of the Choice are represented with strings and images.

Each element of a Choice is composed of a text string part, an Image[270] part, and a font attribute that are all treated as a unit. The font attribute applies to the text part and can be controlled by the application. The application may provide null for the image if the element is not to have an image part. The implementation must display the image at the beginning of the text string. If the Choice also has a selection indicator (such as a radio button or a checkbox) placed at the beginning of the text string, the element's image should be placed between the selection indicator and the beginning of the text string.

When a new element is inserted or appended, the implementation provides a default font for the font attribute. This default font is the same font that is used if the application calls setFont(i, null). All ChoiceGroup instances must have the same default font, and all List instances must have the same default font. However, the default font used for Choice objects may differ from the font returned by Font.getDefaultFont[227].

The Image part of a Choice element may be mutable or immutable. If the Image is mutable, the effect is as if snapshot of its contents is taken at the time the Choice is constructed with this Image or when the Choice element is created or modified with the append[159], insert[162], or set[163] methods. The snapshot is used whenever the contents of the Choice element are to be displayed. Even if the application subsequently draws into the Image, the snapshot is not modified until the next call to one of the above methods. The snapshot is *not* updated when the Choice becomes visible on the display. (This is because the application does not have control over exactly when Displayables and Items appear and disappear from the display.)

The following code illustrates a technique to refresh the image part of element k of a Choice ch:

```
ch.set(k, ch.getString(k), ch.getImage(k));
```

If the application provides an image, the implementation may choose to truncate it if it exceeds the capacity of the device to display it. Images within any particular Choice object should all be of the same size, because the implementation is allowed to allocate the same amount of space for every element. The application can query the implementation's image size recommendation by calling Display.getBestImageWidth(int)[212] and Display.getBestImageHeight(int)[211].

If an element is very long or contains a line break, the implementation may display only a portion of it. If this occurs, the implementation should provide the user with a means to see as much as possible of the element. If this is done by wrapping an element to multiple lines, the second and subsequent lines should show a clear indication to the user that they are part of the same element and are not a new element.

The application can express a preference for the policy used by the implementation for display of long elements including those that contain line break characters. The characters after the first line break may only be visible if the policy permits it. The `setFitPolicy(int)`[163] and `getFitPolicy()`[160] methods control this preference. The valid settings are `TEXT_WRAP_DEFAULT`[158], `TEXT_WRAP_ON`[159], and `TEXT_WRAP_OFF`[159]. Unless specified otherwise by `Choice` implementation classes, the initial value of the element fit policy is `TEXT_WRAP_DEFAULT`.

After a `Choice` object has been created, elements may be inserted, appended, and deleted, and each element's string part and image part may be get and set. Elements within a `Choice` object are referred to by their indexes, which are consecutive integers in the range from zero to `size()-1`, with zero referring to the first element and `size()-1` to the last element.

There are four types of `Choices`: implicit-choice (valid only for `List`[303]), exclusive-choice, multiple-choice, and pop-up (valid only for `ChoiceGroup`[166]).

The exclusive-choice presents a series of elements and interacts with the user. That is, when the user selects an element, that element is shown to be selected using a distinct visual representation. If there are elements present in the `Choice`, one element must be selected at any given time. If at any time a situation would result where there are elements in the exclusive-choice but none is selected, the implementation will choose an element and select it. This situation can arise when an element is added to an empty `Choice`, when the selected element is deleted from the `Choice`, or when a `Choice` is created and populated with elements by a constructor. In these cases, the choice of which element is selected is left to the implementation. Applications for which the selected element is significant should set the selection explicitly. There is no way for the user to unselect an element within an exclusive `Choice`.

The popup choice is similar to the exclusive choice. The selection behavior of a popup choice is identical to that of an exclusive choice. However, a popup choice differs from an exclusive choice in presentation and interaction. In an exclusive choice, all elements should be displayed in-line. In a popup choice, the selected element should always be displayed, and the other elements should remain hidden until the user performs a specific action to show them. For example, an exclusive choice could be implemented as a series of radio buttons with one always selected. A popup choice could be implemented as a popup menu, with the selected element being displayed in the menu button.

The implicit choice is an exclusive choice where the focused or highlighted element is implicitly selected when a command is initiated. As with the exclusive choice, if there are elements present in the `Choice`, one element is always selected.

A multiple-choice presents a series of elements and allows the user to select any number of elements in any combination. As with exclusive-choice, the multiple-choice interacts with the user in object-operation mode. The visual appearance of a multiple-choice will likely have a visual representation distinct from the exclusive-choice that shows the selected state of each element as well as indicating to the user that multiple elements may be selected.

The selected state of an element is a property of the element. This state stays with that element if other elements are inserted or deleted, causing elements to be shifted around. For example, suppose element *n* is selected, and a new element is inserted at index zero. The selected element would now have index *n+1*. A similar rule applies to deletion. Assuming *n* is greater than zero, deleting element zero would leave element *n-1* selected. Setting the contents of an element leaves its selected state unchanged. When a new element is inserted or appended, it is always unselected (except in the special case of adding an element to an empty Exclusive, Popup, or Implicit Choice as mentioned above).

The selected state of a `Choice` object can be controlled by the application with the `setSelectedFlags`[164] and `setSelectedIndex`[164] methods. This state is available to the application through the `getSelectedFlags`[161] and `getSelectedIndex`[162] methods. The selected state reported by these methods is generally identical to what has been set by the application, with the following exceptions. Adding or removing elements may change the selection. When the `Choice` is present on the display, the

implementation's user interface policy and direct user interaction with the object may also affect the selection. For example, the implementation might update the selection to the current highlight location as the user is moving the highlight, or it might set the selection from the highlight only when the user is about to invoke a command. As another example, the implementation might move the highlight (and thus the selection) of an implicit `List` to the first element each time the `List` becomes current. When a `Choice` object is present on the display, applications should query its selected state only within a `CommandListener`[183] or a `ItemStateListener`[301] callback. Querying the state at other times might result in a value different from what has been set by the application (because the user or the implementation's UI policy might have changed it) and it might not reflect the user's intent (because the user might still in the process of making a selection).

Note: Methods have been added to the `Choice` interface in version 2.0. Adding methods to interfaces is normally an incompatible change. However, `Choice` does not appear as a *type* in any field, method parameter, or method return value, and so it is not useful for an application to create a class that implements the `Choice` interface. Future versions of this specification may make additional changes to the `Choice` interface. In order to remain compatible with future versions of this specification, applications should avoid creating classes that implement the `Choice` interface.

Since: MIDP 1.0

Member Summary

Fields

static int	EXCLUSIVE[158]
static int	IMPLICIT[158]
static int	MULTIPLE[158]
static int	POPUP[158]
static int	TEXT_WRAP_DEFAULT[158]
static int	TEXT_WRAP_OFF[159]
static int	TEXT_WRAP_ON[159]

Methods

int	append(String stringPart, Image imagePart)[159]
void	delete(int elementNum)[160]
void	deleteAll()[160]
int	getFitPolicy()[160]
Font	getFont(int elementNum)[160]
Image	getImage(int elementNum)[161]
int	getSelectedFlags(boolean[] selectedArray_return)[161]
int	getSelectedIndex()[162]
java.lang.String	getString(int elementNum)[162]
void	insert(int elementNum, String stringPart, Image imagePart)[162]
boolean	isSelected(int elementNum)[163]
void	set(int elementNum, String stringPart, Image imagePart)[163]
void	setFitPolicy(int fitPolicy)[163]
void	setFont(int elementNum, Font font)[164]
void	setSelectedFlags(boolean[] selectedArray)[164]
void	setSelectedIndex(int elementNum, boolean selected)[164]
int	size()[165]

Fields

EXCLUSIVE

Declaration:
`public static final int EXCLUSIVE`

Description:
`EXCLUSIVE` is a choice having exactly one element selected at time. All elements of an `EXCLUSIVE` type `Choice` should be displayed in-line. That is, the user should not need to perform any extra action to traverse among and select from the elements.

Value `1` is assigned to `EXCLUSIVE`.

IMPLICIT

Declaration:
`public static final int IMPLICIT`

Description:
`IMPLICIT` is a choice in which the currently focused element is selected when a `Command`$_{175}$ is initiated.

The `IMPLICIT` type is not valid for `ChoiceGroup`$_{166}$ objects.

Value `3` is assigned to `IMPLICIT`.

MULTIPLE

Declaration:
`public static final int MULTIPLE`

Description:
`MULTIPLE` is a choice that can have arbitrary number of elements selected at a time.

Value `2` is assigned to `MULTIPLE`.

POPUP

Declaration:
`public static final int POPUP`

Description:
`POPUP` is a choice having exactly one element selected at a time. The selected element is always shown. The other elements should be hidden until the user performs a particular action to show them. When the user performs this action, all elements become accessible. For example, an implementation could use a popup menu to display the elements of a `ChoiceGroup` of type `POPUP`.

The `POPUP` type is not valid for `List`$_{303}$ objects.

Value `4` is assigned to `POPUP`.

Since: MIDP 2.0

TEXT_WRAP_DEFAULT

Declaration:
`public static final int TEXT_WRAP_DEFAULT`

Description:
Constant for indicating that the application has no preference as to wrapping or truncation of text element contents and that the implementation should use its default behavior.

Field has the value `0`.

Since: MIDP 2.0

See Also: `getFitPolicy()`$_{160}$, `setFitPolicy(int)`$_{163}$

TEXT_WRAP_OFF

Declaration:
```
public static final int TEXT_WRAP_OFF
```

Description:
Constant for hinting that text element contents should be limited to a single line. Line ending is forced, for example by cropping, if there is too much text to fit to the line. The implementation should provide some means to present the full element contents. This may be done, for example, by using a special pop-up window or by scrolling the text of the focused element.

Implementations should indicate that cropping has occurred, for example, by placing an ellipsis at the point where the text contents have been cropped.

Field has the value `2`.

Since: MIDP 2.0

See Also: `getFitPolicy()`$_{160}$, `setFitPolicy(int)`$_{163}$

TEXT_WRAP_ON

Declaration:
```
public static final int TEXT_WRAP_ON
```

Description:
Constant for hinting that text element contents should be wrapped to to multiple lines if necessary to fit available content space. The Implementation may limit the maximum number of lines that it will actually present.

Field has the value `1`.

Since: MIDP 2.0

See Also: `getFitPolicy()`$_{160}$, `setFitPolicy(int)`$_{163}$

Methods

append(String, Image)

Declaration:
```
public int append(String stringPart, javax.microedition.lcdui.Image_270 imagePart)
```

Description:
Appends an element to the `Choice`. The added element will be the last element of the `Choice`. The size of the `Choice` grows by one.

Parameters:
`stringPart` - the string part of the element to be added

`imagePart` - the image part of the element to be added, or `null` if there is no image part

Returns: the assigned index of the element

Throws:
`NullPointerException` - if `stringPart` is `null`

delete(int)

Declaration:
```
public void delete(int elementNum)
```

Description:
Deletes the element referenced by `elementNum`. The size of the `Choice` shrinks by one. It is legal to delete all elements from a `Choice`. The `elementNum` parameter must be within the range `[0..size()-1]`, inclusive.

Parameters:
`elementNum` - the index of the element to be deleted

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

deleteAll()

Declaration:
```
public void deleteAll()
```

Description:
Deletes all elements from this `Choice`, leaving it with zero elements. This method does nothing if the `Choice` is already empty.

Since: MIDP 2.0

getFitPolicy()

Declaration:
```
public int getFitPolicy()
```

Description:
Gets the application's preferred policy for fitting `Choice` element contents to the available screen space. The value returned is the policy that had been set by the application, even if that value had been disregarded by the implementation.

Returns: one of `TEXT_WRAP_DEFAULT`$_{158}$, `TEXT_WRAP_ON`$_{159}$, or `TEXT_WRAP_OFF`$_{159}$

Since: MIDP 2.0

See Also: `setFitPolicy(int)`$_{163}$

getFont(int)

Declaration:
```
public javax.microedition.lcdui.Font223 getFont(int elementNum)
```

Description:
Gets the application's preferred font for rendering the specified element of this `Choice`. The value returned is the font that had been set by the application, even if that value had been disregarded by the

implementation. If no font had been set by the application, or if the application explicitly set the font to `null`, the value is the default font chosen by the implementation.

The `elementNum` parameter must be within the range `[0..size()-1]`, inclusive.

Parameters:
`elementNum` - the index of the element, starting from zero

Returns: the preferred font to use to render the element

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

Since: MIDP 2.0

See Also: `setFont(int, Font)`$_{164}$

getImage(int)

Declaration:
```
public javax.microedition.lcdui.Image270 getImage(int elementNum)
```

Description:
Gets the `Image` part of the element referenced by `elementNum`. The `elementNum` parameter must be within the range `[0..size()-1]`, inclusive.

Parameters:
`elementNum` - the index of the element to be queried

Returns: the image part of the element, or `null` if there is no image

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

See Also: `getString(int)`$_{162}$

getSelectedFlags(boolean[])

Declaration:
```
public int getSelectedFlags(boolean[] selectedArray_return)
```

Description:
Queries the state of a `Choice` and returns the state of all elements in the boolean array `selectedArray_return`. **Note:** this is a result parameter. It must be at least as long as the size of the `Choice` as returned by `size()`. If the array is longer, the extra elements are set to `false`.

This call is valid for all types of `Choices`. For `MULTIPLE`, any number of elements may be selected and set to `true` in the result array. For `EXCLUSIVE`, `POPUP`, and `IMPLICIT` exactly one element will be selected (unless there are zero elements in the `Choice`).

Parameters:
`selectedArray_return` - array to contain the results

Returns: the number of selected elements in the `Choice`

Throws:
`IllegalArgumentException` - if `selectedArray_return` is shorter than the size of the `Choice`.

`NullPointerException` - if `selectedArray_return` is `null`

See Also: `setSelectedFlags(boolean[])`$_{164}$

getSelectedIndex()

Declaration:
```
public int getSelectedIndex()
```

Description:
Returns the index number of an element in the `Choice` that is selected. For `Choice` types `EXCLUSIVE`, `POPUP`, and `IMPLICIT` there is at most one element selected, so this method is useful for determining the user's choice. Returns `-1` if the `Choice` has no elements (and therefore has no selected elements).

For `MULTIPLE`, this always returns `-1` because no single value can in general represent the state of such a `Choice`. To get the complete state of a `MULTIPLE` `Choice`, see `getSelectedFlags`$_{161}$.

Returns: index of selected element, or `-1` if none

See Also: `setSelectedIndex(int, boolean)`$_{164}$

getString(int)

Declaration:
```
public String getString(int elementNum)
```

Description:
Gets the `String` part of the element referenced by `elementNum`. The `elementNum` parameter must be within the range `[0..size()-1]`, inclusive.

Parameters:
`elementNum` - the index of the element to be queried

Returns: the string part of the element

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

See Also: `getImage(int)`$_{161}$

insert(int, String, Image)

Declaration:
```
public void insert(int elementNum, String stringPart,
            javax.microedition.lcdui.Image imagePart)
```

Description:
Inserts an element into the `Choice` just prior to the element specified. The size of the `Choice` grows by one. The `elementNum` parameter must be within the range `[0..size()]`, inclusive. The index of the last element is `size()-1`, and so there is actually no element whose index is `size()`. If this value is used for `elementNum`, the new element is inserted immediately after the last element. In this case, the effect is identical to `append()`$_{159}$.

Parameters:
`elementNum` - the index of the element where insertion is to occur

`stringPart` - the string part of the element to be inserted

`imagePart` - the image part of the element to be inserted, or `null` if there is no image part

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

`NullPointerException` - if `stringPart` is `null`

isSelected(int)

Declaration:
```
public boolean isSelected(int elementNum)
```

Description:
Gets a boolean value indicating whether this element is selected. The `elementNum` parameter must be within the range `[0..size()-1]`, inclusive.

Parameters:
`elementNum` - the index of the element to be queried

Returns: selection state of the element

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

set(int, String, Image)

Declaration:
```
public void set(int elementNum, String stringPart,
            javax.microedition.lcdui.Image 270 imagePart)
```

Description:
Sets the `String` and `Image` parts of the element referenced by `elementNum`, replacing the previous contents of the element. The `elementNum` parameter must be within the range `[0..size()-1]`, inclusive. The font attribute of the element is left unchanged.

Parameters:
`elementNum` - the index of the element to be set

`stringPart` - the string part of the new element

`imagePart` - the image part of the element, or `null` if there is no image part

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

`NullPointerException` - if `stringPart` is `null`

setFitPolicy(int)

Declaration:
```
public void setFitPolicy(int fitPolicy)
```

Description:
Sets the application's preferred policy for fitting `Choice` element contents to the available screen space. The set policy applies for all elements of the `Choice` object. Valid values are `TEXT_WRAP_DEFAULT` 158, `TEXT_WRAP_ON` 159, and `TEXT_WRAP_OFF` 159. Fit policy is a hint, and the implementation may disregard the application's preferred policy.

Parameters:
`fitPolicy` - preferred content fit policy for choice elements

Throws:
`IllegalArgumentException` - if `fitPolicy` is invalid

Since: MIDP 2.0

See Also: `getFitPolicy()` 160

setFont(int, Font)

Declaration:
```
public void setFont(int elementNum, javax.microedition.lcdui.Font font)
```

Description:
Sets the application's preferred font for rendering the specified element of this `Choice`. An element's font is a hint, and the implementation may disregard the application's preferred font.

The `elementNum` parameter must be within the range `[0..size()-1]`, inclusive.

The `font` parameter must be a valid `Font` object or `null`. If the `font` parameter is `null`, the implementation must use its default font to render the element.

Parameters:
`elementNum` - the index of the element, starting from zero

`font` - the preferred font to use to render the element

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

Since: MIDP 2.0

See Also: `getFont(int)`

setSelectedFlags(boolean[])

Declaration:
```
public void setSelectedFlags(boolean[] selectedArray)
```

Description:
Attempts to set the selected state of every element in the `Choice`. The array must be at least as long as the size of the `Choice`. If the array is longer, the additional values are ignored.

For `Choice` objects of type `MULTIPLE`, this sets the selected state of every element in the `Choice`. An arbitrary number of elements may be selected.

For `Choice` objects of type `EXCLUSIVE`, `POPUP`, and `IMPLICIT`, exactly one array element must have the value `true`. If no element is `true`, the first element in the `Choice` will be selected. If two or more elements are `true`, the implementation will choose the first `true` element and select it.

Parameters:
`selectedArray` - an array in which the method collect the selection status

Throws:
`IllegalArgumentException` - if `selectedArray` is shorter than the size of the `Choice`

`NullPointerException` - if `selectedArray` is `null`

See Also: `getSelectedFlags(boolean[])`

setSelectedIndex(int, boolean)

Declaration:
```
public void setSelectedIndex(int elementNum, boolean selected)
```

Description:
For `MULTIPLE`, this simply sets an individual element's selected state.

For `EXCLUSIVE` and `POPUP`, this can be used only to select any element, that is, the `selected` parameter must be `true`. When an element is selected, the previously selected element is deselected. If `selected` is `false`, this call is ignored. If element was already selected, the call has no effect.

For `IMPLICIT`, this can be used only to select any element, that is, the `selected` parameter must be `true`. When an element is selected, the previously selected element is deselected. If `selected` is `false`, this call is ignored. If element was already selected, the call has no effect.

The call to `setSelectedIndex` does not cause implicit activation of any `Command`.

For all list types, the `elementNum` parameter must be within the range `[0..size()-1]`, inclusive.

Parameters:

`elementNum` - the index of the element, starting from zero

`selected` - the state of the element, where `true` means selected and `false` means not selected

Throws:

`IndexOutOfBoundsException` - if `elementNum` is invalid

See Also: `getSelectedIndex()` *162*

size()

Declaration:

```
public int size()
```

Description:

Gets the number of elements present.

Returns: the number of elements in the `Choice`

javax.microedition.lcdui
ChoiceGroup

Declaration

```
public class ChoiceGroup extends Item287 implements Choice155

Object
  |
  +--javax.microedition.lcdui.Item287
        |
        +--javax.microedition.lcdui.ChoiceGroup
```

All Implemented Interfaces: Choice155

Description

A `ChoiceGroup` is a group of selectable elements intended to be placed within a `Form`231 . The group may be created with a mode that requires a single choice to be made or that allows multiple choices. The implementation is responsible for providing the graphical representation of these modes and must provide visually different graphics for different modes. For example, it might use “radio buttons” for the single choice mode and “check boxes” for the multiple choice mode.

Note: most of the essential methods have been specified in the `Choice`155 interface.

Since: MIDP 1.0

Member Summary	
Constructors	
	`ChoiceGroup(String label, int choiceType)`167
	`ChoiceGroup(String label, int choiceType, String stringElements, Image imageElements)`168
Methods	
`int`	`append(String stringPart, Image imagePart)`168
`void`	`delete(int elementNum)`169
`void`	`deleteAll()`169
`int`	`getFitPolicy()`169
`Font`	`getFont(int elementNum)`169
`Image`	`getImage(int elementNum)`170
`int`	`getSelectedFlags(boolean[] selectedArray_return)`170
`int`	`getSelectedIndex()`171
`java.lang.String`	`getString(int elementNum)`171
`void`	`insert(int elementNum, String stringPart, Image imagePart)`171
`boolean`	`isSelected(int elementNum)`172
`void`	`set(int elementNum, String stringPart, Image imagePart)`172
`void`	`setFitPolicy(int fitPolicy)`172
`void`	`setFont(int elementNum, Font font)`173
`void`	`setSelectedFlags(boolean[] selectedArray)`173
`void`	`setSelectedIndex(int elementNum, boolean selected)`174

Member Summary	
int	size()[174]

Inherited Member Summary
Fields inherited from interface Choice[155]
EXCLUSIVE[158], IMPLICIT[158], MULTIPLE[158], POPUP[158], TEXT_WRAP_DEFAULT[158], TEXT_WRAP_OFF[159], TEXT_WRAP_ON[159]
Fields inherited from class Item[287]
BUTTON[291], HYPERLINK[292], LAYOUT_2[292], LAYOUT_BOTTOM[292], LAYOUT_CENTER[292], LAYOUT_DEFAULT[292], LAYOUT_EXPAND[293], LAYOUT_LEFT[293], LAYOUT_NEWLINE_AFTER[293], LAYOUT_NEWLINE_BEFORE[293], LAYOUT_RIGHT[293], LAYOUT_SHRINK[294], LAYOUT_TOP[294], LAYOUT_VCENTER[294], LAYOUT_VEXPAND[294], LAYOUT_VSHRINK[294], PLAIN[295]
Methods inherited from class Item[287]
addCommand(Command)[295], getLabel()[295], getLayout()[295], getMinimumHeight()[296], getMinimumWidth()[296], getPreferredHeight()[296], getPreferredWidth()[296], notifyStateChanged()[297], removeCommand(Command)[297], setDefaultCommand(Command)[298], setItemCommandListener(ItemCommandListener)[298], setLabel(String)[298], setLayout(int)[299], setPreferredSize(int, int)[299]
Methods inherited from class Object
equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Constructors

ChoiceGroup(String, int)

Declaration:
`public ChoiceGroup(String label, int choiceType)`

Description:
Creates a new, empty `ChoiceGroup`, specifying its title and its type. The type must be one of `EXCLUSIVE`, `MULTIPLE`, or `POPUP`. The `IMPLICIT` choice type is not allowed within a `ChoiceGroup`.

Parameters:
`label` - the item's label (see `Item`[287])

`choiceType` - `EXCLUSIVE`, `MULTIPLE`, or `POPUP`

Throws:
`IllegalArgumentException` - if `choiceType` is not one of `EXCLUSIVE`, `MULTIPLE`, or `POPUP`

See Also: `Choice.EXCLUSIVE`[158], `Choice.MULTIPLE`[158], `Choice.IMPLICIT`[158], `Choice.POPUP`[158]

ChoiceGroup(String, int, String[], Image[])

Declaration:

```
public ChoiceGroup(String label, int choiceType, String[] stringElements,
            javax.microedition.lcdui.Image[]270 imageElements)
```

Description:

Creates a new `ChoiceGroup`, specifying its title, the type of the `ChoiceGroup`, and an array of `Strings` and `Images` to be used as its initial contents.

The type must be one of `EXCLUSIVE`, `MULTIPLE`, or `POPUP`. The `IMPLICIT` type is not allowed for `ChoiceGroup`.

The `stringElements` array must be non-null and every array element must also be non-null. The length of the `stringElements` array determines the number of elements in the `ChoiceGroup`. The `imageElements` array may be `null` to indicate that the `ChoiceGroup` elements have no images. If the `imageElements` array is non-null, it must be the same length as the `stringElements` array. Individual elements of the `imageElements` array may be `null` in order to indicate the absence of an image for the corresponding `ChoiceGroup` element. Non-null elements of the `imageElements` array may refer to mutable or immutable images.

Parameters:

`label` - the item's label (see `Item`[287])

`choiceType` - `EXCLUSIVE`, `MULTIPLE`, or `POPUP`

`stringElements` - set of strings specifying the string parts of the `ChoiceGroup` elements

`imageElements` - set of images specifying the image parts of the `ChoiceGroup` elements

Throws:

`NullPointerException` - if `stringElements` is `null`

`NullPointerException` - if the `stringElements` array contains any `null` elements

`IllegalArgumentException` - if the `imageElements` array is non-null and has a different length from the `stringElements` array

`IllegalArgumentException` - if `choiceType` is not one of `EXCLUSIVE`, `MULTIPLE`, or `POPUP`

See Also: `Choice.EXCLUSIVE`[158], `Choice.MULTIPLE`[158], `Choice.IMPLICIT`[158], `Choice.POPUP`[158]

Methods

append(String, Image)

Declaration:

```
public int append(String stringPart, javax.microedition.lcdui.Image270 imagePart)
```

Description:

Appends an element to the `ChoiceGroup`.

Specified By: `append`[159] in interface `Choice`[155]

Parameters:

`stringPart` - the string part of the element to be added

`imagePart` - the image part of the element to be added, or `null` if there is no image part

Returns: the assigned index of the element

Throws:
`NullPointerException` - if `stringPart` is `null`

delete(int)

Declaration:
`public void delete(int elementNum)`

Description:
Deletes the element referenced by `elementNum`.

Specified By: `delete`[160] in interface `Choice`[155]

Parameters:
`elementNum` - the index of the element to be deleted

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

deleteAll()

Declaration:
`public void deleteAll()`

Description:
Deletes all elements from this `ChoiceGroup`.

Specified By: `deleteAll`[160] in interface `Choice`[155]

getFitPolicy()

Declaration:
`public int getFitPolicy()`

Description:
Gets the application's preferred policy for fitting `Choice` element contents to the available screen space. The value returned is the policy that had been set by the application, even if that value had been disregarded by the implementation.

Specified By: `getFitPolicy`[160] in interface `Choice`[155]

Returns: one of `Choice.TEXT_WRAP_DEFAULT`[158], `Choice.TEXT_WRAP_ON`[159], or `Choice.TEXT_WRAP_OFF`[159]

Since: MIDP 2.0

See Also: `setFitPolicy(int)`[172]

getFont(int)

Declaration:
`public javax.microedition.lcdui.Font`[223] `getFont(int elementNum)`

Description:
Gets the application's preferred font for rendering the specified element of this `Choice`. The value returned is the font that had been set by the application, even if that value had been disregarded by the implementation. If no font had been set by the application, or if the application explicitly set the font to `null`, the value is the default font chosen by the implementation.

The `elementNum` parameter must be within the range `[0..size()-1]`, inclusive.

Specified By: `getFont`$_{160}$ in interface `Choice`$_{155}$

Parameters:
`elementNum` - the index of the element, starting from zero

Returns: the preferred font to use to render the element

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

Since: MIDP 2.0

See Also: `setFont(int, Font)`$_{173}$

getImage(int)

Declaration:
`public javax.microedition.lcdui.Image`$_{270}$ **`getImage`**`(int elementNum)`

Description:
Gets the `Image` part of the element referenced by `elementNum`.

Specified By: `getImage`$_{161}$ in interface `Choice`$_{155}$

Parameters:
`elementNum` - the number of the element to be queried

Returns: the image part of the element, or null if there is no image

Throws:
`IndexOutOfBoundsException` - if elementNum is invalid

See Also: `getString(int)`$_{171}$

getSelectedFlags(boolean[])

Declaration:
`public int` **`getSelectedFlags`**`(boolean[] selectedArray_return)`

Description:
Queries the state of a `ChoiceGroup` and returns the state of all elements in the boolean array `selectedArray_return`. **Note:** this is a result parameter. It must be at least as long as the size of the `ChoiceGroup` as returned by `size()`. If the array is longer, the extra elements are set to `false`.

For `ChoiceGroup` objects of type `MULTIPLE`, any number of elements may be selected and set to true in the result array. For `ChoiceGroup` objects of type `EXCLUSIVE` and `POPUP`, exactly one element will be selected, unless there are zero elements in the `ChoiceGroup`.

Specified By: `getSelectedFlags`$_{161}$ in interface `Choice`$_{155}$

Parameters:
`selectedArray_return` - array to contain the results

Returns: the number of selected elements in the `ChoiceGroup`

Throws:
`IllegalArgumentException` - if `selectedArray_return` is shorter than the size of the `ChoiceGroup`

`NullPointerException` - if `selectedArray_return` is null

See Also: `setSelectedFlags(boolean[])`$_{173}$

getSelectedIndex()

Declaration:
```
public int getSelectedIndex()
```

Description:
Returns the index number of an element in the `ChoiceGroup` that is selected. For `ChoiceGroup` objects of type `EXCLUSIVE` and `POPUP` there is at most one element selected, so this method is useful for determining the user's choice. Returns `-1` if there are no elements in the `ChoiceGroup`.

For `ChoiceGroup` objects of type `MULTIPLE`, this always returns `-1` because no single value can in general represent the state of such a `ChoiceGroup`. To get the complete state of a `MULTIPLE` `Choice`, see `getSelectedFlags`$_{170}$.

Specified By: `getSelectedIndex`$_{162}$ in interface `Choice`$_{155}$

Returns: index of selected element, or `-1` if none

See Also: `setSelectedIndex(int, boolean)`$_{174}$

getString(int)

Declaration:
```
public String getString(int elementNum)
```

Description:
Gets the `String` part of the element referenced by `elementNum`.

Specified By: `getString`$_{162}$ in interface `Choice`$_{155}$

Parameters:
`elementNum` - the index of the element to be queried

Returns: the string part of the element

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

See Also: `getImage(int)`$_{170}$

insert(int, String, Image)

Declaration:
```
public void insert(int elementNum, String stringPart,
            javax.microedition.lcdui.Image_270 imagePart)
```

Description:
Inserts an element into the `ChoiceGroup` just prior to the element specified.

Specified By: `insert`$_{162}$ in interface `Choice`$_{155}$

Parameters:
`elementNum` - the index of the element where insertion is to occur

`stringPart` - the string part of the element to be inserted

`imagePart` - the image part of the element to be inserted, or `null` if there is no image part

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

`NullPointerException` - if `stringPart` is `null`

isSelected(int)

Declaration:
```
public boolean isSelected(int elementNum)
```

Description:
Gets a boolean value indicating whether this element is selected.

Specified By: `isSelected`[163] in interface `Choice`[155]

Parameters:
`elementNum` - the index of the element to be queried

Returns: selection state of the element

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

set(int, String, Image)

Declaration:
```
public void set(int elementNum, String stringPart,
            javax.microedition.lcdui.Image[270] imagePart)
```

Description:
Sets the `String` and `Image` parts of the element referenced by `elementNum`, replacing the previous contents of the element.

Specified By: `set`[163] in interface `Choice`[155]

Parameters:
`elementNum` - the index of the element to be set

`stringPart` - the string part of the new element

`imagePart` - the image part of the element, or `null` if there is no image part

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

`NullPointerException` - if `stringPart` is `null`

setFitPolicy(int)

Declaration:
```
public void setFitPolicy(int fitPolicy)
```

Description:
Sets the application's preferred policy for fitting `Choice` element contents to the available screen space. The set policy applies for all elements of the `Choice` object. Valid values are `Choice.TEXT_WRAP_DEFAULT`[158], `Choice.TEXT_WRAP_ON`[159], and `Choice.TEXT_WRAP_OFF`[159]. Fit policy is a hint, and the implementation may disregard the application's preferred policy.

Specified By: `setFitPolicy`[163] in interface `Choice`[155]

Parameters:
`fitPolicy` - preferred content fit policy for choice elements

Throws:
`IllegalArgumentException` - if `fitPolicy` is invalid

Since: MIDP 2.0

See Also: `getFitPolicy()`[169]

setFont(int, Font)

Declaration:
```
public void setFont(int elementNum, javax.microedition.lcdui.Font[223] font)
```

Description:
Sets the application's preferred font for rendering the specified element of this `Choice`. An element's font is a hint, and the implementation may disregard the application's preferred font.

The `elementNum` parameter must be within the range `[0..size()-1]`, inclusive.

The `font` parameter must be a valid `Font` object or `null`. If the `font` parameter is `null`, the implementation must use its default font to render the element.

Specified By: `setFont`[164] in interface `Choice`[155]

Parameters:
`elementNum` - the index of the element, starting from zero

`font` - the preferred font to use to render the element

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

Since: MIDP 2.0

See Also: `getFont(int)`[169]

setSelectedFlags(boolean[])

Declaration:
```
public void setSelectedFlags(boolean[] selectedArray)
```

Description:
Attempts to set the selected state of every element in the `ChoiceGroup`. The array must be at least as long as the size of the `ChoiceGroup`. If the array is longer, the additional values are ignored.

For `ChoiceGroup` objects of type `MULTIPLE`, this sets the selected state of every element in the `Choice`. An arbitrary number of elements may be selected.

For `ChoiceGroup` objects of type `EXCLUSIVE` and `POPUP`, exactly one array element must have the value `true`. If no element is `true`, the first element in the `Choice` will be selected. If two or more elements are `true`, the implementation will choose the first `true` element and select it.

Specified By: `setSelectedFlags`[164] in interface `Choice`[155]

Parameters:
`selectedArray` - an array in which the method collect the selection status

Throws:
`IllegalArgumentException` - if `selectedArray` is shorter than the size of the `ChoiceGroup`

`NullPointerException` - if the `selectedArray` is `null`

See Also: `getSelectedFlags(boolean[])`[170]

setSelectedIndex(int, boolean)

Declaration:
```
public void setSelectedIndex(int elementNum, boolean selected)
```

Description:
For `ChoiceGroup` objects of type `MULTIPLE`, this simply sets an individual element's selected state.

For `ChoiceGroup` objects of type `EXCLUSIVE` and `POPUP`, this can be used only to select an element. That is, the `selected` parameter must be `true`. When an element is selected, the previously selected element is deselected. If `selected` is `false`, this call is ignored.

For both list types, the `elementNum` parameter must be within the range `[0..size()-1]`, inclusive.

Specified By: `setSelectedIndex` 164 in interface `Choice` 155

Parameters:
`elementNum` - the number of the element. Indexing of the elements is zero-based

`selected` - the new state of the element `true=selected`, `false=not` selected

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

See Also: `getSelectedIndex()` 171

size()

Declaration:
```
public int size()
```

Description:
Returns the number of elements in the `ChoiceGroup`.

Specified By: `size` 165 in interface `Choice` 155

Returns: the number of elements in the `ChoiceGroup`

javax.microedition.lcdui
Command

Declaration

```
public class Command

Object
  |
  +--javax.microedition.lcdui.Command
```

Description

The `Command` class is a construct that encapsulates the semantic information of an action. The behavior that the command activates is not encapsulated in this object. This means that command contains only information about "command" not the actual action that happens when command is activated. The action is defined in a `CommandListener` 183 associated with the `Displayable`. `Command` objects are *presented* in the user interface and the way they are presented may depend on the semantic information contained within the command.

`Commands` may be implemented in any user interface construct that has semantics for activating a single action. This, for example, can be a soft button, item in a menu, or some other direct user interface construct. For example, a speech interface may present these commands as voice tags.

The mapping to concrete user interface constructs may also depend on the total number of the commands. For example, if an application asks for more abstract commands than can be mapped onto the available physical buttons on a device, then the device may use an alternate human interface such as a menu. For example, the abstract commands that cannot be mapped onto physical buttons are placed in a menu and the label "Menu" is mapped onto one of the programmable buttons.

A command contains four pieces of information: a *short label*, an optional *long label*, a *type*, and a *priority.* One of the labels is used for the visual representation of the command, whereas the type and the priority indicate the semantics of the command.

Labels

Each command includes one or two label strings. The label strings are what the application requests to be shown to the user to represent this command. For example, one of these strings may appear next to a soft button on the device or as an element in a menu. For command types other than `SCREEN`, the labels provided may be overridden by a system-specific label that is more appropriate for this command on this device. The contents of the label strings are otherwise not interpreted by the implementation.

All commands have a short label. The long label is optional. If the long label is not present on a command, the short label is always used.

The short label string should be as short as possible so that it consumes a minimum of screen real estate. The long label can be longer and more descriptive, but it should be no longer than a few words. For example, a command's short label might be "Play", and its long label might be "Play Sound Clip".

The implementation chooses one of the labels to be presented in the user interface based on the context and the amount of space available. For example, the implementation might use the short label if the command appears on a soft button, and it might use the long label if the command appears on a menu, but only if there is room on the menu for the long label. The implementation may use the short labels of some commands and the long labels of other commands, and it is allowed to switch between using the short and long label at will. The application cannot determine which label is being used at any given time.

size()

Type

The application uses the command type to specify the intent of this command. For example, if the application specifies that the command is of type BACK, and if the device has a standard of placing the “back” operation on a certain soft-button, the implementation can follow the style of the device by using the semantic information as a guide. The defined types are BACK[178], CANCEL[178], EXIT[179], HELP[179], ITEM[179], OK[179], SCREEN[180], and STOP[180].

Priority

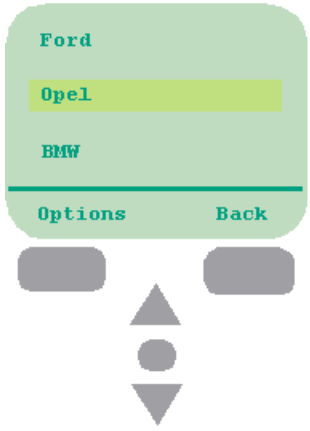
The application uses the priority value to describe the importance of this command relative to other commands on the same screen. Priority values are integers, where a lower number indicates greater importance. The actual values are chosen by the application. A priority value of one might indicate the most important command, priority values of two, three, four, and so on indicate commands of lesser importance.

Typically, the implementation first chooses the placement of a command based on the type of command and then places similar commands based on a priority order. This could mean that the command with the highest priority is placed so that user can trigger it directly and that commands with lower priority are placed on a menu. It is not an error for there to be commands on the same screen with the same priorities and types. If this occurs, the implementation will choose the order in which they are presented.

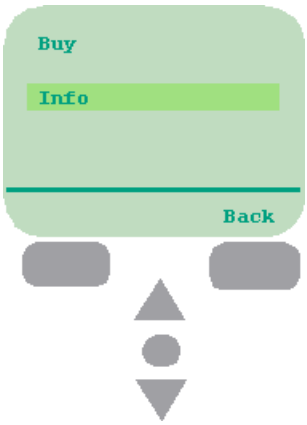
For example, if the application has the following set of commands:

```
new Command(“Buy”, Command.ITEM, 1);
new Command(“Info”, Command.ITEM, 1);
new Command(“Back”, Command.BACK, 1);
```

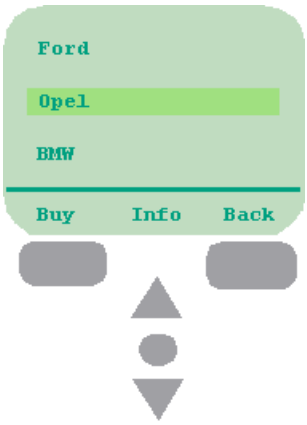
An implementation with two soft buttons may map the BACK command to the right soft button and create an “Options” menu on the left soft button to contain the other commands.



When user presses the left soft button, a menu with the two remaining Commands appears:



If the application had three soft buttons, all commands can be mapped to soft buttons:



The application is always responsible for providing the means for the user to progress through different screens. An application may set up a screen that has no commands. This is allowed by the API but is generally not useful; if this occurs the user would have no means to move to another screen. Such program would simply considered to be in error. A typical device should provide a means for the user to direct the application manager to kill the erroneous application.

Since: MIDP 1.0

Member Summary
Fields
static int BACK[178]
static int CANCEL[178]
static int EXIT[179]
static int HELP[179]
static int ITEM[179]
static int OK[179]
static int SCREEN[180]
static int STOP[180]

Member Summary

Constructors

`Command(String label, int commandType, int priority)`[180]
`Command(String shortLabel, String longLabel, int commandType, int priority)`[181]

Methods

int	`getCommandType()`[181]
java.lang.String	`getLabel()`[181]
java.lang.String	`getLongLabel()`[181]
int	`getPriority()`[182]

Inherited Member Summary

Methods inherited from class `Object`

`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Fields

BACK

Declaration:
`public static final int BACK`

Description:
A navigation command that returns the user to the logically previous screen. The jump to the previous screen is not done automatically by the implementation but by the `commandAction`[183] provided by the application. Note that the application defines the actual action since the strictly previous screen may not be logically correct.

Value 2 is assigned to BACK.

See Also: `CANCEL`[178], `STOP`[180]

CANCEL

Declaration:
`public static final int CANCEL`

Description:
A command that is a standard negative answer to a dialog implemented by current screen. Nothing is cancelled automatically by the implementation; cancellation is implemented by the `commandAction`[183] provided by the application.

With this command type, the application hints to the implementation that the user wants to dismiss the current screen without taking any action on anything that has been entered into the current screen, and usually that the user wants to return to the prior screen. In many cases CANCEL is interchangeable with BACK, but BACK is mainly used for navigation as in a browser-oriented applications.

Value 3 is assigned to CANCEL.

See Also: BACK[178], STOP[180]

EXIT

Declaration:
```
public static final int EXIT
```

Description:
A command used for exiting from the application. When the user invokes this command, the implementation does not exit automatically. The application's commandAction[183] will be called, and it should exit the application if it is appropriate to do so.

Value 7 is assigned to EXIT.

HELP

Declaration:
```
public static final int HELP
```

Description:
This command specifies a request for on-line help. No help information is shown automatically by the implementation. The commandAction[183] provided by the application is responsible for showing the help information.

Value 5 is assigned to HELP.

ITEM

Declaration:
```
public static final int ITEM
```

Description:
With this command type the application can hint to the implementation that the command is specific to the items of the Screen or the elements of a Choice. Normally this means that command relates to the focused item or element. For example, an implementation of List can use this information for creating context sensitive menus.

Value 8 is assigned to ITEM.

OK

Declaration:
```
public static final int OK
```

Description:
A command that is a standard positive answer to a dialog implemented by current screen. Nothing is done automatically by the implementation; any action taken is implemented by the commandAction[183] provided by the application.

With this command type the application hints to the implementation that the user will use this command to ask the application to confirm the data that has been entered in the current screen and to proceed to the next logical screen.

CANCEL is often used together with OK.

Value 4 is assigned to OK.

SCREEN

See Also: CANCEL[178]

SCREEN

Declaration:
`public static final int SCREEN`

Description:
Specifies an application-defined command that pertains to the current screen. Examples could be “Load” and “Save”. A `SCREEN` command generally applies to the entire screen’s contents or to navigation among screens. This is in constrast to the `ITEM` type, which applies to the currently activated or focused item or element contained within this screen.

Value `1` is assigned to `SCREEN`.

STOP

Declaration:
`public static final int STOP`

Description:
A command that will stop some currently running process, operation, etc. Nothing is stopped automatically by the implementation. The cessation must be performed by the `commandAction`[183] provided by the application.

With this command type the application hints to the implementation that the user will use this command to stop any currently running process visible to the user on the current screen. Examples of running processes might include downloading or sending of data. Use of the `STOP` command does not necessarily imply a switch to another screen.

Value `6` is assigned to `STOP`.

See Also: BACK[178], CANCEL[178]

Constructors

Command(String, int, int)

Declaration:
`public Command(String label, int commandType, int priority)`

Description:
Creates a new command object with the given short label, type, and priority. The newly created command has no long label. This constructor is identical to `Command(label, null, commandType, priority)`.

Parameters:

`label` - the command’s short label

`commandType` - the command’s type

`priority` - the command’s priority value

Throws:

`NullPointerException` - if label is `null`

`IllegalArgumentException` - if the `commandType` is an invalid type

See Also: `Command(String, String, int, int)`[181]

Command(String, String, int, int)

Declaration:
```
public Command(String shortLabel, String longLabel, int commandType, int priority)
```

Description:
Creates a new command object with the given labels, type, and priority.

The short label is required and must not be `null`. The long label is optional and may be `null` if the command is to have no long label.

Parameters:

`shortLabel` - the command's short label

`longLabel` - the command's long label, or `null` if none

`commandType` - the command's type

`priority` - the command's priority value

Throws:

`NullPointerException` - if `shortLabel` is `null`

`IllegalArgumentException` - if the `commandType` is an invalid type

Since: MIDP 2.0

Methods

getCommandType()

Declaration:
```
public int getCommandType()
```

Description:
Gets the type of the command.

Returns: type of the `Command`

getLabel()

Declaration:
```
public String getLabel()
```

Description:
Gets the short label of the command.

Returns: the `Command's` short label

getLongLabel()

Declaration:
```
public String getLongLabel()
```

Description:
Gets the long label of the command.

Returns: the `Command's` long label, or `null` if the `Command` has no long label

getPriority()

Since: MIDP 2.0

getPriority()

Declaration:
`public int getPriority()`

Description:
Gets the priority of the command.

Returns: priority of the `Command`

javax.microedition.lcdui
CommandListener

Declaration
```
public interface CommandListener
```

Description
This interface is used by applications which need to receive high-level events from the implementation. An application will provide an implementation of a `CommandListener` (typically by using a nested class or an inner class) and will then provide the instance to the `addCommand` method on a `Displayable` in order to receive high-level events on that screen.

The specification does not require the platform to create several threads for the event delivery. Thus, if a `CommandListener` method does not return or the return is not delayed, the system may be blocked. So, there is the following note to application developers:

- *the CommandListener method should return immediately.*

Since: MIDP 1.0

See Also: `Displayable.setCommandListener(CommandListener)`[221]

Member Summary	
Methods	
`void`	`commandAction(Command c, Displayable d)`[183]

Methods

commandAction(Command, Displayable)

Declaration:
```
public void commandAction(javax.microedition.lcdui.Command[175] c,
            javax.microedition.lcdui.Displayable[218] d)
```

Description:
Indicates that a command event has occurred on `Displayable d`.

Parameters:

`c` - a `Command` object identifying the command. This is either one of the applications have been added to `Displayable` with `addCommand(Command)`[219] or is the implicit `SELECT_COMMAND`[306] of `List`.

`d` - the `Displayable` on which this event has occurred

javax.microedition.lcdui
CustomItem

Declaration

```
public abstract class CustomItem extends Item287

Object
  |
  +--javax.microedition.lcdui.Item287
        |
        +--javax.microedition.lcdui.CustomItem
```

Description

A CustomItem is customizable by subclassing to introduce new visual and interactive elements into `Forms`. Subclasses are responsible for their visual appearance including sizing and rendering and choice of colors, fonts and graphics. Subclasses are responsible for the user interaction mode by responding to events generated by keys, pointer actions, and traversal actions. Finally, subclasses are responsible for calling `Item.notifyStateChanged()`$_{297}$ to trigger notification of listeners that the `CustomItem's` value has changed.

Like other `Items`, `CustomItems` have the concept of *minimum* and *preferred* sizes. These pertain to the total area of the `Item`, which includes space for the content, label, borders, etc. See Item Sizes for a full discussion of the areas and sizes of `Items`.

`CustomItem` subclasses also have the concept of the *content* size, which is the size of only the content area of the `CustomItem`. The content area is a rectangular area inside the total area occupied by the `CustomItem`. The content area is the area within which the `CustomItem` subclass paints and receives input events. It does not include space consumed by labels and borders. The implementation is responsible for laying out, painting, and handling input events within the area of the `Item` that is outside the content area.

All coordinates passed between the implementation and the `CustomItem` subclass are relative to the item's content area, with the upper-left corner of this area being located at `(0,0)`. Size information passed between the implementation and the `CustomItem` subclass with the `getMinContentHeight`$_{191}$, `getMinContentWidth`$_{191}$, `getPrefContentHeight`$_{191}$, `getPrefContentWidth`$_{192}$, and `sizeChanged`$_{196}$ methods all refer to the size of the content area. The implementation is responsible for computing and maintaining the difference between the size of the content area and the size of the total area of the `Item` as reported by the Item size methods `Item.getMinimumHeight`$_{296}$, `Item.getMinimumWidth`$_{296}$, `Item.getPreferredHeight`$_{296}$, and `Item.getPreferredWidth`$_{296}$.

The implementation may disregard sizing information returned from a `CustomItem` if it exceeds limits imposed by the implementation's user interface policy. In this case, the implementation must always report the actual size granted to the `CustomItem` via the `sizeChanged`$_{196}$ and the `paint`$_{193}$ methods. For example, this situation may occur if the implementation prohibits an `Item` from becoming wider than the screen. If the `CustomItem` subclass code returns a value from `getMinContentWidth` that would result in the `CustomItem` being wider than the screen, the implementation may assign a width smaller than the minimum width returned by `getMinContentWidth`.

The implementation is allowed to call the `CustomItem's` content size methods `getMinContentHeight`$_{191}$, `getMinContentWidth`$_{191}$, `getPrefContentHeight`$_{191}$, and `getPrefContentWidth`$_{192}$, in any order with respect to other `CustomItem` methods. For all of these methods, the `CustomItem` subclass code must return values that are consistent with the current contents of the

`CustomItem`. If the contents changes, it is not sufficient for the `CustomItem` subclass code simply to begin returning different values from the content size methods. Instead, the subclass code must call the `invalidate`$_{192}$ method whenever its contents changes. This indicates to the implementation that it may need to perform its layout computation, which will call the content size methods to get new values based on the `CustomItem's` new contents.

Interaction Modes

The `CustomItem` class is intended to allow edit-in-place on many items, but it does not allow every conceivable interaction. Desire for flexibility has been balanced against a requirement that these APIs be simple enough to master easily, along with a need to allow for platform-specific variations in look-and-feel, all without sacrificing interoperability.

The general idea is that there are multiple interaction "modes" and that the `Form` implementation can convey which ones it supports. The `CustomItem` can then choose to support one or more interaction modes. There is no requirement for a `CustomItem` to implement all combinations of all interaction modes. Typically, a `CustomItem` will implement an approach (such as the separate editing screen technique discussed below) that works on all platforms, in addition to a highly interactive approach that relies on a particular interaction mode. At run time, the `CustomItem` code can query the system to determine whether this interaction mode is supported. If it is, the `CustomItem` can use it; otherwise, it will fall back to the approach that works on all platforms.

`CustomItem` can always use item commands to invoke a separate editing screen, although components with a small number of discrete states could simply respond by changing the state and then causing an `notifyStateChanged` notification. A technique for using a separate editing screen would be to load the value into another `Displayable` object (such as a List) and then to call `Display.setCurrent(Displayable)`$_{214}$ on it. When the user issues a command (such as "OK") to indicate that editing of this value is complete, the listener can retrieve the value from that `Displayable` object and then call `Display.setCurrentItem(Item)`$_{216}$ to return to this item.

Keypad Input

The implementation may optionally support delivery of keypad events to the `CustomItem`. The implementation indicates the level of support by setting the `KEY_PRESS`, `KEY_RELEASE`, and `KEY_REPEAT` bits in the value returned by `getInteractionModes`. Events corresponding to these bits are delivered through calls to the `keyPressed()`, `keyReleased()`, and `keyRepeated()` methods, respectively. If an implementation supports `KEY_RELEASE` events, it must also support `KEY_PRESS` events. If an implementation supports `KEY_REPEAT` events, it must also support `KEY_PRESS` and `KEY_RELEASE` events. If supported, `KEY_RELEASE` events will generally occur after a corresponding `KEY_PRESS` event is received, and `KEY_REPEAT` events will generally occur between `KEY_PRESS` and `KEY_RELEASE` events. However, it is possible for the `CustomItem` to receive `KEY_RELEASE` or `KEY_REPEAT` events without a corresponding `KEY_PRESS` if a key is down when the `CustomItem` becomes visible.

Key event methods are passed the `keyCode` indicating the key on which the event occurred. Implementations must provide means for the user to generate events with key codes `Canvas.KEY_NUM0` through `Canvas.KEY_NUM9`, `Canvas.KEY_STAR`, and `Canvas.KEY_POUND`. Implementations may also deliver key events for other keys, include device-specific keys. The set of keys available to a `CustomItem` may differ depending upon whether commands have been added to it.

The application may map key codes to game actions through use of the `getGameAction` method. If the implementation supports key events on `CustomItems`, the implementation must provide a sufficient set of key codes and a mapping to game actions such that all game actions are available to `CustomItems`.

The set of keys and the key events available to a `CustomItem` may differ from what is available on a `Canvas`. In particular, on a system that supports traversal, the system might use directional keys for traversal and elect not to deliver these keys to `CustomItems`. The mapping between key codes and game actions in a `CustomItem` may differ from the mapping in a `Canvas`. See Key Events and Game Actions on class `Canvas` for further information about key codes and game actions.

Pointer Input

The implementation may optionally support delivery of pointer events (such as taps with a stylus) to the `CustomItem`. The implementation indicates the level of support by setting the `POINTER_PRESS`, `POINTER_RELEASE`, and `POINTER_DRAG` bits in the value returned by `getInteractionModes`. Events corresponding to these bits are delivered through calls to the `pointerPressed()`, `pointerReleased()`, and `pointerDragged()` methods, respectively. If an implementation supports `POINTER_RELEASE` events, it must also support `POINTER_PRESS` events. If an implementation supports `POINTER_DRAG` events, it must also support `POINTER_PRESS` and `POINTER_RELEASE` events. If supported, `POINTER_RELEASE` events will generally occur after a corresponding `POINTER_PRESS` event is received, and `POINTER_DRAG` events will generally occur between `POINTER_PRESS` and `POINTER_RELEASE` events. However, it is possible for the `CustomItem` to receive `POINTER_RELEASE` or `POINTER_DRAG` events without a corresponding `POINTER_PRESS` if the pointer is down when the `CustomItem` becomes visible.

The `(x,y)` location of the pointer event is reported with every pointer event. This location is expressed in the coordinate system of the `CustomItem`, where `(0,0)` is the upper-left corner of the `CustomItem`. Under certain circumstances, pointer events may occur outside the bounds of the item.

Traversal

An implementation may support traversal *internal* to a `CustomItem`, that is, the implementation may temporarily delegate the responsibility for traversal to the item itself. Even if there is only one traversal location inside the `CustomItem`, the item may want support the internal traversal protocol so that it can perform specialized highlighting, animation, etc. when the user has traversed into it.

The implementation indicates its support for traversal internal to a `CustomItem` by setting one or both of the `TRAVERSE_HORIZONTAL` or `TRAVERSE_VERTICAL` bits in the value returned by `getInteractionModes()`. If neither of these bits is set, the implementation is unwilling to let `CustomItems` traverse internally, or the implementation does not support traversal at all. If the implementation does support traversal but has declined to permit traversal internal to `CustomItems`, the implementation will supply its own highlighting outside the `CustomItem's` content area.

The `CustomItem` need not support internal traversal at all. It can do this by returning `false` to the initial call to the `traverse` method. (This is the default behavior if this method hasn't been overridden by the `CustomItem`.) If this occurs, the system must arrange for the user to be able to traverse onto and past this item. The system must also arrange for proper scrolling to take place, particularly if the item exceeds the height of the screen, regardless of whether internal traversal is occurring.

An implementation may provide support for delivering keypad or pointer events to `CustomItems` even if it has declined to support delivering traverse events to `CustomItems`. If an implementation provides support for delivering keypad or pointer events to `CustomItems`, it must provide a means to do so for every `CustomItem`, even for those that have refused internal traversal by returning `false` to the initial `traverse()` call. This implies that such implementations must still support some notion of focus for an item, even if that item is not supporting internal traversal.

See the documentation for the `traverse`*196* method for a full specification of the behavior and responsibilities required for the item to perform internal traversal.

Item Appearance

The visual appearance of each item consists of a label (handled by the implementation) and its contents (handled by the subclass).

Labels are the responsibility of the implementation, not the item. The screen area that is allocated to the `CustomItem` for its contents is separate from the area that the implementation uses to display the `CustomItem's` label. The implementation controls the rendering of the label and its layout with respect to the content area.

The `CustomItem` is responsible for painting its contents whenever the `paint` method is called.

The colors for foreground, background, highlighted foreground, highlighted background, border, and highlighted border should be retrieved from `Display.getColor(int)`[212] . This will allow `CustomItems` to match the color scheme of other items provided with the device. The `CustomItem` is responsible for keeping track of its own highlighted and unhighlighted state.

The fonts used should be retrieved from `Font.getFont(int)`[228] . This will allow them to match the fonts used by other items on the device for a consistent visual appearance.

Since: MIDP 2.0

Member Summary

Fields

```
   protected static int  KEY_PRESS[188]
   protected static int  KEY_RELEASE[188]
   protected static int  KEY_REPEAT[189]
   protected static int  NONE[189]
   protected static int  POINTER_DRAG[189]
   protected static int  POINTER_PRESS[189]
   protected static int  POINTER_RELEASE[189]
   protected static int  TRAVERSE_HORIZONTAL[190]
   protected static int  TRAVERSE_VERTICAL[190]
```

Constructors

```
              protected  CustomItem(String label)[190]
```

Methods

```
                    int  getGameAction(int keyCode)[190]
          protected int  getInteractionModes()[191]
protected abstract int  getMinContentHeight()[191]
protected abstract int  getMinContentWidth()[191]
protected abstract int  getPrefContentHeight(int width)[191]
protected abstract int  getPrefContentWidth(int height)[192]
         protected void  hideNotify()[192]
         protected void  invalidate()[192]
         protected void  keyPressed(int keyCode)[193]
         protected void  keyReleased(int keyCode)[193]
         protected void  keyRepeated(int keyCode)[193]
    protected abstract   paint(Graphics g, int w, int h)[193]
                  void
         protected void  pointerDragged(int x, int y)[194]
         protected void  pointerPressed(int x, int y)[194]
         protected void  pointerReleased(int x, int y)[195]
         protected void  repaint()[195]
```

Member Summary	
protected void	repaint(int x, int y, int w, int h)[195]
protected void	showNotify()[195]
protected void	sizeChanged(int w, int h)[196]
protected boolean	traverse(int dir, int viewportWidth, int viewportHeight, int[] visRect_inout)[196]
protected void	traverseOut()[199]

Inherited Member Summary

Fields inherited from class Item[287]

BUTTON[291], HYPERLINK[292], LAYOUT_2[292], LAYOUT_BOTTOM[292], LAYOUT_CENTER[292], LAYOUT_DEFAULT[292], LAYOUT_EXPAND[293], LAYOUT_LEFT[293], LAYOUT_NEWLINE_AFTER[293], LAYOUT_NEWLINE_BEFORE[293], LAYOUT_RIGHT[293], LAYOUT_SHRINK[294], LAYOUT_TOP[294], LAYOUT_VCENTER[294], LAYOUT_VEXPAND[294], LAYOUT_VSHRINK[294], PLAIN[295]

Methods inherited from class Item[287]

addCommand(Command)[295], getLabel()[295], getLayout()[295], getMinimumHeight()[296], getMinimumWidth()[296], getPreferredHeight()[296], getPreferredWidth()[296], notifyStateChanged()[297], removeCommand(Command)[297], setDefaultCommand(Command)[298], setItemCommandListener(ItemCommandListener)[298], setLabel(String)[298], setLayout(int)[299], setPreferredSize(int, int)[299]

Methods inherited from class Object

equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Fields

KEY_PRESS

Declaration:
`protected static final int KEY_PRESS`

Description:
Interaction mode bit indicating support for key pressed events.

KEY_PRESS has the value 4.

See Also: getInteractionModes()[191], keyPressed(int)[193]

KEY_RELEASE

Declaration:
`protected static final int KEY_RELEASE`

Description:
Interaction mode bit indicating support for key released events.

KEY_RELEASE has the value 8.

See Also: `getInteractionModes()`[191], `keyReleased(int)`[193]

KEY_REPEAT

Declaration:
`protected static final int KEY_REPEAT`

Description:
Interaction mode bit indicating support for key repeated events.

`KEY_REPEAT` has the value `0x10`.

See Also: `getInteractionModes()`[191], `keyRepeated(int)`[193]

NONE

Declaration:
`protected static final int NONE`

Description:
A value for traversal direction that indicates that traversal has entered or has changed location within this item, but that no specific direction is associated with this traversal event.

`NONE` has the value `0`.

See Also: `traverse(int, int, int, int[])`[196]

POINTER_DRAG

Declaration:
`protected static final int POINTER_DRAG`

Description:
Interaction mode bit indicating support for point dragged events.

`POINTER_DRAG` has the value `0x80`.

See Also: `getInteractionModes()`[191], `pointerDragged(int, int)`[194]

POINTER_PRESS

Declaration:
`protected static final int POINTER_PRESS`

Description:
Interaction mode bit indicating support for point pressed events.

`POINTER_PRESS` has the value `0x20`.

See Also: `getInteractionModes()`[191], `pointerPressed(int, int)`[194]

POINTER_RELEASE

Declaration:
`protected static final int POINTER_RELEASE`

Description:
Interaction mode bit indicating support for point released events.

`POINTER_RELEASE` has the value `0x40`.

See Also: `getInteractionModes()`[191], `pointerReleased(int, int)`[195]

TRAVERSE_HORIZONTAL

Declaration:
`protected static final int TRAVERSE_HORIZONTAL`

Description:
Interaction mode bit indicating support of horizontal traversal internal to the `CustomItem`.

`TRAVERSE_HORIZONTAL` has the value `1`.

See Also: `getInteractionModes()`[191], `traverse(int, int, int, int[])`[196]

TRAVERSE_VERTICAL

Declaration:
`protected static final int TRAVERSE_VERTICAL`

Description:
Interaction mode bit indicating support for vertical traversal internal to the `CustomItem`.

`TRAVERSE_VERTICAL` has the value `2`.

See Also: `getInteractionModes()`[191], `traverse(int, int, int, int[])`[196]

Constructors

CustomItem(String)

Declaration:
`protected CustomItem(String label)`

Description:
Superclass constructor, provided so that the `CustomItem` subclass can specify its label.

Parameters:
`label` - the `CustomItem's` label

Methods

getGameAction(int)

Declaration:
`public int getGameAction(int keyCode)`

Description:
Gets the game action associated with the given key code of the device. Returns zero if no game action is associated with this key code. See the Game Actions section of class `Canvas` for further discussion of game actions.

The mapping of key codes to game actions may differ between `CustomItem` and `Canvas`.

Parameters:
`keyCode` - the key code

Returns: the game action corresponding to this key, or `0` if none

Throws:
`IllegalArgumentException` - if `keyCode` is not a valid key code

getInteractionModes()

Declaration:
```
protected final int getInteractionModes()
```

Description:
Gets the available interaction modes. This method is intended to be called by `CustomItem` subclass code in order for it to determine what kinds of input are available from this device. The modes available may be dependent upon several factors: the hardware keys on the actual device, which of these keys are needed for the system to do proper navigation, the presence of a pointing device, etc. See Interaction Modes for further discussion. If this method returns `0`, the only interaction available is through item commands.

Returns: a bitmask of the available interaction modes

getMinContentHeight()

Declaration:
```
protected abstract int getMinContentHeight()
```

Description:
Implemented by the subclass to return the minimum height of the content area, in pixels. This method is called by the implementation as part of its layout algorithm. The actual height granted is reported in the `sizeChanged`$_{196}$ and `paint`$_{193}$ methods.

Returns: the minimum content height in pixels

getMinContentWidth()

Declaration:
```
protected abstract int getMinContentWidth()
```

Description:
Implemented by the subclass to return the minimum width of the content area, in pixels. This method is called by the implementation as part of its layout algorithm. The actual width granted is reported in the `sizeChanged`$_{196}$ and `paint`$_{193}$ methods.

Returns: the minimum content width in pixels

getPrefContentHeight(int)

Declaration:
```
protected abstract int getPrefContentHeight(int width)
```

Description:
Implemented by the subclass to return the preferred height of the content area, in pixels. This method is called by the implementation as part of its layout algorithm.

The `width` parameter is the tentative width assigned to the content area. The subclass code may use this value in its computation of the preferred height. The `width` parameter will be -1 if the implementation has not assigned a tentative value for the width. Otherwise, `width` will have a specific value if the application has locked the width of the `CustomItem` or if the container's layout algorithm has already computed a tentative width at the time of this call. The subclass must not assume that the tentative width passed or the preferred height returned will be granted. The actual size granted is reported in the `sizeChanged`$_{196}$ and `paint`$_{193}$ methods.

Parameters:
`width` - the tentative content width in pixels, or -1 if a tentative width has not been computed

Returns: the preferred content height in pixels

getPrefContentWidth(int)

Declaration:
```
protected abstract int getPrefContentWidth(int height)
```

Description:
Implemented by the subclass to return the preferred width of the content area, in pixels. This method is called by the implementation as part of its layout algorithm.

The `height` parameter is the tentative height assigned to the content area. The subclass code may use this value in its computation of the preferred width. The `height` parameter will be -1 if the implementation has not assigned a tentative value for the height. Otherwise, `height` will have a specific value if the application has locked the height of the `CustomItem` or if the container's layout algorithm has already computed a tentative height at the time of this call. The subclass must not assume that the tentative height passed or the preferred width returned will be granted. The actual size granted is reported in the `sizeChanged`[196] and `paint`[193] methods.

Parameters:
`height` - the tentative content height in pixels, or -1 if a tentative height has not been computed

Returns: the preferred content width in pixels

hideNotify()

Declaration:
```
protected void hideNotify()
```

Description:
Called by the system to notify the item that it is now completely invisible, when it previously had been at least partially visible. No further `paint()` calls will be made on this item until after a `showNotify()` has been called again.

The default implementation of this method does nothing.

invalidate()

Declaration:
```
protected final void invalidate()
```

Description:
Signals that the `CustomItem's` size and traversal location need to be updated. This method is intended to be called by `CustomItem` subclass code to inform the implementation that the size of the `CustomItem's` content area or the internal traversal location might need to change. This often occurs if the contents of the `CustomItem` are modified. A call to this method will return immediately, and it will cause the container's layout algorithm to run at some point in the future, possibly resulting in calls to `getMinContentHeight`[191], `getMinContentWidth`[191], `getPrefContentHeight`[191], `getPrefContentWidth`[192], `sizeChanged`[196], or `traverse`[196]. The `paint`[193] method may also be called if repainting is necessary as a result of the layout operation. If the content size is invalidated while the `CustomItem` is not visible, the layout operation may be deferred. The `traverse` method will be called if the `CustomItem` contains the current traversal location at the time `invalidate` is called.

keyPressed(int)

Declaration:
```
protected void keyPressed(int keyCode)
```

Description:
Called by the system when a key is pressed. The implementation indicates support for delivery of key press events by setting the KEY_PRESS bit in the value returned by the `getInteractionModes` method.

Parameters:
`keyCode` - the key code of the key that has been pressed

See Also: `getInteractionModes()`191

keyReleased(int)

Declaration:
```
protected void keyReleased(int keyCode)
```

Description:
Called by the system when a key is released. The implementation indicates support for delivery of key release events by setting the KEY_RELEASE bit in the value returned by the `getInteractionModes` method.

Parameters:
`keyCode` - the key code of the key that has been released

See Also: `getInteractionModes()`191

keyRepeated(int)

Declaration:
```
protected void keyRepeated(int keyCode)
```

Description:
Called by the system when a key is repeated. The implementation indicates support for delivery of key repeat events by setting the KEY_REPEAT bit in the value returned by the `getInteractionModes` method.

Parameters:
`keyCode` - the key code of the key that has been repeated

See Also: `getInteractionModes()`191

paint(Graphics, int, int)

Declaration:
```
protected abstract void paint(javax.microedition.lcdui.Graphics247 g, int w, int h)
```

Description:
Implemented by the subclass to render the item within its container. At the time of the call, the `Graphics` context's destination is the content area of this `CustomItem` (or back buffer for it). The Translation is set so that the upper left corner of the content area is at `(0,0)`, and the clip is set to the area to be painted. The application must paint every pixel within the given clip area. The item is allowed to modify the clip area, but the system must not allow any modification to result in drawing outside the bounds of the item's content area. The `w` and `h` passed in are the width and height of the content area of the item. These values will always be equal to the values passed with the most recent call to `sizeChanged()`; they are passed here as well for convenience.

Other values of the `Graphics` object are as follows:

- the current color is black;
- the font is the same as the font returned by `Font.getDefaultFont()`[227];
- the stroke style is `SOLID`[254];

The `paint()` method will be called only after `showNotify()` call on this item and before a subsequent `hideNotify()` call on this item, in other words, only when at least a portion of the item is actually visible on the display. In addition, the `paint()` method will be called only if the item's width and height are both greater than zero.

Parameters:

`g` - the `Graphics` object to be used for rendering the item

`w` - current width of the item in pixels

`h` - current height of the item in pixels

pointerDragged(int, int)

Declaration:

```
protected void pointerDragged(int x, int y)
```

Description:

Called by the system when a pointer drag action (for example, pen motion after a press but before a release) has occurred within the item. The `(x,y)` coordinates are relative to the origin of the item. Implementations should deliver pointer drag events to an item even if the pointer is being moved outside the item. In this case the `(x,y)` coordinates may indicate a location outside the bounds of the item. The implementation indicates support for delivery of pointer release events by setting the `POINTER_DRAG` bit in the value returned by the `getInteractionModes` method.

Parameters:

`x` - the `x` coordinate of the pointer drag

`y` - the `x` coordinate of the pointer drag

See Also: `getInteractionModes()`[191]

pointerPressed(int, int)

Declaration:

```
protected void pointerPressed(int x, int y)
```

Description:

Called by the system when a pointer down action (for example, a pen tap) has occurred within the item. The `(x,y)` coordinates are relative to the origin of the item, and they will always indicate a location within the item. The implementation indicates support for delivery of pointer press events by setting the `POINTER_PRESS` bit in the value returned by the `getInteractionModes` method.

Parameters:

`x` - the `x` coordinate of the pointer down

`y` - the `y` coordinate of the pointer down

See Also: `getInteractionModes()`[191]

pointerReleased(int, int)

Declaration:
```
protected void pointerReleased(int x, int y)
```

Description:
Called by the system when a pointer up action (for example, a pen lift) has occurred after a pointer down action had occurred within the item. The `(x,y)` coordinates are relative to the origin of the item. Implementations should deliver a pointer release event to an item even if the pointer has moved outside the item when the release occurs. In this case the `(x,y)` coordinates may indicate a location outside the bounds of the item. The implementation indicates support for delivery of pointer release events by setting the `POINTER_RELEASE` bit in the value returned by the `getInteractionModes` method.

Parameters:
`x` - the x coordinate of the pointer up

`y` - the x coordinate of the pointer up

See Also: `getInteractionModes()` *191*

repaint()

Declaration:
```
protected final void repaint()
```

Description:
Called by subclass code to request that the item be repainted. If this item is visible on the display, this will result in a call to `paint()` the next time the `CustomItem` is to be displayed. The `CustomItem` subclass should call this method when the item's internal state has been updated such that its visual representation needs to be updated.

repaint(int, int, int, int)

Declaration:
```
protected final void repaint(int x, int y, int w, int h)
```

Description:
Called by subclass code to request that the specified rectangular area of the item be repainted. If that area is visible on the display, this will result in call to `paint` with graphics set to include the specified rectangular area. The area is specified relative to the `CustomItem's` content area. The `CustomItem` should call this method when the item's internal state has been updated and only part of the visual representation needs to be updated.

Parameters:
`x` - the x coordinate of the rectangular area to be updated

`y` - the y coordinate of the rectangular area to be updated

`w` - the width of the rectangular area to be updated

`h` - the height of the rectangular area to be updated

showNotify()

Declaration:
```
protected void showNotify()
```

Description:
Called by the system to notify the item that it is now at least partially visible, when it previously had been completely invisible. The item may receive `paint()` calls after `showNotify()` has been called.

The default implementation of this method does nothing.

sizeChanged(int, int)

Declaration:
```
protected void sizeChanged(int w, int h)
```

Description:
Implemented by the subclass in order to handle size change events. This method is called by the system when the size of the content area of this `CustomItem` has changed.

If the size of a `CustomItem` changes while it is visible on the display, it may trigger an automatic repaint request. If this occurs, the call to `sizeChanged` will occur prior to the call to `paint`. If the `CustomItem` has become smaller, the implementation may choose not to trigger a repaint request if the remaining contents of the `CustomItem` have been preserved. Similarly, if the `CustomItem` has become larger, the implementation may choose to trigger a repaint only for the new region. In both cases, the preserved contents must remain stationary with respect to the origin of the `CustomItem`. If the size change is significant to the contents of the `CustomItem`, the application must explicitly issue a repaint request for the changed areas. Note that the application's repaint request should not cause multiple repaints, since it can be coalesced with repaint requests that are already pending.

If the size of the item's content area changes while it is not visible, calls to this method may be deferred. If the size had changed while the item was not visible, `sizeChanged` will be called at least once before the item becomes visible once again.

The default implementation of this method does nothing.

Parameters:
`w` - the new width of the item's content area

`h` - the new height of the item's content area

traverse(int, int, int, int[])

Declaration:
```
protected boolean traverse(int dir, int viewportWidth, int viewportHeight,
            int[] visRect_inout)
```

Description:
Called by the system when traversal has entered the item or has occurred within the item. The direction of traversal and the item's visible rectangle are passed into the method. The method must do one of the following: it must either update its state information pertaining to its internal traversal location, set the return rectangle to indicate a region associated with this location, and return `true`; or, it must return `false` to indicate that this item does not support internal traversal, or that that internal traversal has reached the edge of the item and that traversal should proceed to the next item if possible.

The implementation indicates support for internal traversal within a `CustomItem` by setting one or both of the `TRAVERSE_HORIZONTAL` or `TRAVERSE_VERTICAL` bits in the value returned by the `getInteractionModes` method. The `dir` parameter indicates the direction of traversal by using `Canvas` game actions `Canvas.UP`, `Canvas.DOWN`, `Canvas.LEFT`, and `Canvas.RIGHT`, or the value `NONE`, which indicates that there is no specific direction associated with this traversal event. If the `TRAVERSE_HORIZONTAL` bit is set, this indicates that the `Canvas.LEFT` and `Canvas.RIGHT` values

will be used to indicate the traversal direction. If the TRAVERSE_VERTICAL bit is set, this indicates that the Canvas.UP and Canvas.DOWN values will be used to indicate the traversal direction. If both bits are set, all four direction values may be used for the traversal direction, indicating that the item should perform two-dimensional traversal. The dir parameter may have the value NONE under any combination of the TRAVERSE_VERTICAL and TRAVERSE_HORIZONTAL bits.

Although Canvas game actions are used to indicate the traversal direction, this does not imply that the keys mapped to these game actions are being used for traversal, nor that that keys are being used for traversal at all.

The viewportWidth and viewportHeight parameters indicate the size of the viewable area the item's container has granted to its items. This represents the largest area of the item that is likely to be visible at any given time.

The visRect_inout parameter is used both for passing information into this method and for returning information from this method. It must be an int[4] array. The information in this array is a rectangle of the form [x,y,w,h] where (x,y) is the location of the upper-left corner of the rectangle relative to the item's origin, and (w,h) are the width and height of the rectangle. The return values placed into this array are significant only when the traverse() method returns true. The values are ignored if the traverse() method returns false.

When this method is called, the visRect_inout array contains a rectangle representing the region of the item that is currently visible. This region might have zero area if no part of the item is visible, for example, if it is scrolled offscreen. The semantics of the rectangle returned are discussed below.

The CustomItem must maintain state that tracks whether traversal is within this item, and if it is, it must also record the current internal location. Initially, traversal is outside the item. The first call to the traverse() method indicates that traversal has entered the item. Subsequent calls to this method indicate that traversal is occurring within this item. Traversal remains within the item until the traverseOut method is called. The CustomItem must keep track of its traversal state so that it can distinguish traversal *entering* the item from traversal *within* the item.

When traversal enters the item, the traversal code should initialize its internal traversal location to the "first" location appropriate for the item's structure and the traversal direction. As an example of the latter policy, if the traversal direction is DOWN, the initial location should be the topmost internal element of the item. Similarly, if the traversal direction is UP, the initial location should be the bottommost element of the item. The CustomItem should still choose the "first" location appropriately even if its primary axis is orthogonal to the axis of traversal. For example, suppose the traversal mode supported is TRAVERSE_VERTICAL but the CustomItem is structured as a horizontal row of elements. If the initial traversal direction is DOWN, the initial location might be the leftmost element, and if the initial traversal direction is UP, the initial location might be the rightmost element.

Traversal may enter the item without any specific direction, in which case the traversal direction will be NONE. This may occur if the user selects the item directly (e.g., with a pointing device), or if the item gains the focus because its containing Form has become current. The CustomItem should choose a default traversal location. If the CustomItem had been traversed to previously, and if it is appropriate for the user interface of the CustomItem, the previous traversal location should be restored.

When traversal occurs within the item, the internal traversal location must be moved to the next appropriate region in the direction of traversal. The item must report its updated internal traversal location in the visRect_inout return parameter as described below and return true. The item will typically provide a highlight to display the internal traversal location to the user. Thus, the item will typically also request repaints of the old and new traversal locations after each traversal event. There is no requirement that the area the item requests to be repainted is the same as the area returned in the visRect_inout rectangle.

The system will combine any repaint requests with any additional repainting that may occur as a result of scrolling.

The `traverse()` method may be called with a direction of `NONE` when the traversal is already within the `CustomItem`. This will occur in response to the `CustomItem` subclass code having called the `invalidate()` method. In this case, the `CustomItem` should simply return its current notion of the traversal location. This mechanism is useful if the `CustomItem` needs to update the traversal location spontaneously (that is, not in response to a traversal event), for example, because of a change in its contents.

If the internal traversal location is such that the traversal event would logically cause traversal to proceed out of the item, the item should return `false` from the `traverse()` method. For example, if the current traversal location is the bottommost internal element of the item, and the traversal direction is `DOWN`, the `traverse()` method should simply return `false`. In this case the method need not update the values in the `visRect_inout` array. The item must leave its internal traversal location unchanged, and it should not request a repaint to update its highlighting. It should defer these actions until the `traverseOut()` method is called. The system will call the `traverseOut()` method when traversal actually leaves the item. The system might not call the `traverseOut()` method, even if `traverse()` has returned `false`, if this item is at the edge of the `Form` or there is no other item beyond to accept the traversal. Even if the `traverse()` method returns `false`, the traversal location is still within this item. It remains within this item until `traverseOut()` is called.

Note the subtle distinction here between the initial `traverse()` call signifying *entry* into the item and subsequent calls signifying traversal *within* the item. A return value of `false` to the initial call indicates that this item performs no internal traversal at all, whereas a return of `false` to subsequent calls indicates that traversal is within this item and may now exit.

The width and height of the rectangle returned in the `visRect_inout` array are used by the `Form` for scrolling and painting purposes. The `Form` must always position the item so that the upper left corner of this rectangle, as specified by the `(x,y)` position, is visible. In addition, the item may also specify a width and height, in which case the `Form` will attempt to position the item so that as much of this rectangle as possible is visible. If the width and height are larger than the size of the viewport, the bottom and right portions of this rectangle will most likely not be visible to the user. The rectangle thus returned will typically denote the size and location of one of the item's internal elements, and it will also typically (though not necessarily) correspond to where the element's highlight will be painted. Width and height values of zero are legal and are not treated specially. Negative values of width and height are treated as if they were zero.

There is no requirement on the location of the rectangle returned in the `visRect_inout` array with respect to the traversal direction. For example, if the `CustomItem` implements internal scrolling, a traversal direction of `DOWN` may cause the item's contents to scroll upwards far enough so that the rectangle returned may be above its old location. `CustomItem` subclasses must ensure that continued traversal in one direction will eventually reach the edge of the item and then traverse out by returning `false` from this method. `CustomItems` must not implement "wraparound" behavior (for example, traversing downwards from the bottommost element moves the traversal location to the topmost element) because this will trap the traversal within the item.

If the `CustomItem` consists of internal elements that are smaller than the container's viewport, the rectangle returned should be the same size as one of these elements. However, the `CustomItem` might have contents whose elements are larger than the viewport, or it might have contents having no internal structure. In either of these cases, the item should return a rectangle that best represents its idea of the content area that is important for the user to see. When traversal occurs, the item should move its traversal location by an amount based on the viewport size. For example, if the viewport is `80` pixels high, and

traversal occurs downwards, the item might move its traversal location down by `70` pixels in order to display the next screenful of content, with `10` pixels overlap for context.

All internal traversal locations must be reachable regardless of which traversal modes are provided by the implementation. This implies that, if the implementation provides one-dimensional traversal, the `CustomItem` must linearize its internal locations. For example, suppose the traversal mode is `TRAVERSE_VERTICAL` and the `CustomItem` consists of a horizontal row of elements. If the traversal direction is `DOWN` the internal traversal location should move to the right, and if the traversal direction is `UP` the internal traversal location should move to the left. (The foregoing convention is appropriate for languages that use left-to-right text. The opposite convention should be used for languages that use right-to-left text.) Consider a similar example where the traversal mode is `TRAVERSE_VERTICAL` and the `CustomItem` consists of a grid of elements. A traversal direction of `DOWN` might proceed leftwards across each row, moving to the next row downwards when the location reaches the rightmost element in a row.

If the implementation provides two-dimensional traversal but the `CustomItem` is one-dimensional, a traversal direction along the item's axis should traverse within the item, and a traversal direction orthogonal to the item's axis should cause immediate traversal out of the item by returning `false` from this method. For example, suppose a `CustomItem` is implementing a vertical stack of elements and traversal is already inside the item. If a traverse event is received with direction `UP` or `DOWN`, the `traverse()` method should move to the next element and return `true`. On the other hand, if a traverse event is received with direction `RIGHT` or `LEFT`, the `traverse()` method should always return `false` so that traversal exits the item immediately. An item that implements internal traversal should always accept entry - that is, the initial call to `traverse()` should return `true` - regardless of the axis of the traversal direction.

If the `traverse()` method returns `false` when traversal is entering the item, this indicates to the system that the item does not support internal traversal. In this case, the item should not perform any of its own highlighting, and the system will perform highlighting appropriate for the platform, external to the item.

The default implementation of the `traverse()` method always returns `false`.

Parameters:

`dir` - the direction of traversal, one of `Canvas.UP`[147], `Canvas.DOWN`[143], `Canvas.LEFT`[146], `Canvas.RIGHT`[147], or `NONE`[189].

`viewportWidth` - the width of the container's viewport

`viewportHeight` - the height of the container's viewport

`visRect_inout` - passes the visible rectangle into the method, and returns the updated traversal rectangle from the method

Returns: `true` if internal traversal had occurred, `false` if traversal should proceed out

See Also: `getInteractionModes()`[191], `traverseOut()`[199], `TRAVERSE_HORIZONTAL`[190], `TRAVERSE_VERTICAL`[190]

traverseOut()

Declaration:

```
protected void traverseOut()
```

Description:

Called by the system when traversal has occurred out of the item. This may occur in response to the `CustomItem` having returned `false` to a previous call to `traverse()`, if the user has begun interacting with another item, or if `Form` containing this item is no longer current. If the `CustomItem` is using highlighting to indicate internal traversal, the `CustomItem` should set its state to be unhighlighted and request a repaint. (Note that painting will not occur if the item is no longer visible.)

traverseOut()

See Also: `getInteractionModes()`[191], `traverse(int, int, int, int[])`[196], `TRAVERSE_HORIZONTAL`[190], `TRAVERSE_VERTICAL`[190]

javax.microedition.lcdui
DateField

Declaration

```
public class DateField extends Item287

Object
  |
  +--javax.microedition.lcdui.Item287
        |
        +--javax.microedition.lcdui.DateField
```

Description

A `DateField` is an editable component for presenting date and time (calendar) information that may be placed into a `Form`. Value for this field can be initially set or left unset. If value is not set then the UI for the field shows this clearly. The field value for "not initialized state" is not valid value and `getDate()` for this state returns `null`.

Instance of a `DateField` can be configured to accept date or time information or both of them. This input mode configuration is done by `DATE`, `TIME` or `DATE_TIME` static fields of this class. `DATE` input mode allows to set only date information and `TIME` only time information (hours, minutes). `DATE_TIME` allows to set both clock time and date values.

In `TIME` input mode the date components of `Date` object must be set to the "zero epoch" value of January 1, 1970.

Calendar calculations in this field are based on default locale and defined time zone. Because of the calculations and different input modes date object may not contain same millisecond value when set to this field and get back from this field.

Since: MIDP 1.0

Member Summary	
Fields	
static int	DATE202
static int	DATE_TIME202
static int	TIME202
Constructors	
	DateField(String label, int mode)203
	DateField(String label, int mode, java.util.TimeZone timeZone)203
Methods	
java.util.Date	getDate()203
int	getInputMode()204
void	setDate(java.util.Date date)204
void	setInputMode(int mode)204

Inherited Member Summary

Fields inherited from class `Item`[287]

`BUTTON`[291], `HYPERLINK`[292], `LAYOUT_2`[292], `LAYOUT_BOTTOM`[292], `LAYOUT_CENTER`[292], `LAYOUT_DEFAULT`[292], `LAYOUT_EXPAND`[293], `LAYOUT_LEFT`[293], `LAYOUT_NEWLINE_AFTER`[293], `LAYOUT_NEWLINE_BEFORE`[293], `LAYOUT_RIGHT`[293], `LAYOUT_SHRINK`[294], `LAYOUT_TOP`[294], `LAYOUT_VCENTER`[294], `LAYOUT_VEXPAND`[294], `LAYOUT_VSHRINK`[294], `PLAIN`[295]

Methods inherited from class `Item`[287]

`addCommand(Command)`[295], `getLabel()`[295], `getLayout()`[295], `getMinimumHeight()`[296], `getMinimumWidth()`[296], `getPreferredHeight()`[296], `getPreferredWidth()`[296], `notifyStateChanged()`[297], `removeCommand(Command)`[297], `setDefaultCommand(Command)`[298], `setItemCommandListener(ItemCommandListener)`[298], `setLabel(String)`[298], `setLayout(int)`[299], `setPreferredSize(int, int)`[299]

Methods inherited from class `Object`

`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait()`, `wait()`, `wait()`

Fields

DATE

Declaration:
`public static final int DATE`

Description:
Input mode for date information (day, month, year). With this mode this `DateField` presents and allows only to modify date value. The time information of date object is ignored.

Value `1` is assigned to `DATE`.

DATE_TIME

Declaration:
`public static final int DATE_TIME`

Description:
Input mode for date (day, month, year) and time (minutes, hours) information. With this mode this `DateField` presents and allows to modify both time and date information.

Value `3` is assigned to `DATE_TIME`.

TIME

Declaration:
`public static final int TIME`

Description:
Input mode for time information (hours and minutes). With this mode this `DateField` presents and allows only to modify time. The date components should be set to the “zero epoch” value of January 1, 1970 and should not be accessed.

Value 2 is assigned to `TIME`.

Constructors

DateField(String, int)

Declaration:
```
public DateField(String label, int mode)
```

Description:
Creates a `DateField` object with the specified label and mode. This call is identical to `DateField(label, mode, null)`.

Parameters:
`label` - item label

`mode` - the input mode, one of `DATE`, `TIME` or `DATE_TIME`

Throws:
`IllegalArgumentException` - if the input `mode's` value is invalid

DateField(String, int, TimeZone)

Declaration:
```
public DateField(String label, int mode, java.util.TimeZone timeZone)
```

Description:
Creates a date field in which calendar calculations are based on specific `TimeZone` object and the default calendaring system for the current locale. The value of the `DateField` is initially in the “uninitialized” state. If `timeZone` is `null`, the system’s default time zone is used.

Parameters:
`label` - item label

`mode` - the input mode, one of `DATE`, `TIME` or `DATE_TIME`

`timeZone` - a specific time zone, or `null` for the default time zone

Throws:
`IllegalArgumentException` - if the input `mode's` value is invalid

Methods

getDate()

Declaration:
```
public java.util.Date getDate()
```

Description:
Returns date value of this field. Returned value is `null` if field value is not initialized. The date object is constructed according the rules of locale specific calendaring system and defined time zone. In `TIME` mode field the date components are set to the “zero epoch” value of January 1, 1970. If a date object that presents time beyond one day from this “zero epoch” then this field is in “not initialized” state and this method returns `null`. In `DATE` mode field the time component of the calendar is set to zero when constructing the date object.

getInputMode()

Returns: date object representing time or date depending on input mode

See Also: `setDate(Date)`[204]

getInputMode()

Declaration:
```
public int getInputMode()
```

Description:
Gets input mode for this date field. Valid input modes are `DATE`, `TIME` and `DATE_TIME`.

Returns: input mode of this field

See Also: `setInputMode(int)`[204]

setDate(Date)

Declaration:
```
public void setDate(java.util.Date date)
```

Description:
Sets a new value for this field. `null` can be passed to set the field state to "not initialized" state. The input mode of this field defines what components of passed `Date` object is used.

In `TIME` input mode the date components must be set to the "zero epoch" value of January 1, 1970. If a date object that presents time beyond one day then this field is in "not initialized" state. In `TIME` input mode the date component of `Date` object is ignored and time component is used to precision of minutes.

In `DATE` input mode the time component of `Date` object is ignored.

In `DATE_TIME` input mode the date and time component of `Date` are used but only to precision of minutes.

Parameters:
`date` - new value for this field

See Also: `getDate()`[203]

setInputMode(int)

Declaration:
```
public void setInputMode(int mode)
```

Description:
Set input mode for this date field. Valid input modes are `DATE`, `TIME` and `DATE_TIME`.

Parameters:
`mode` - the input mode, must be one of `DATE`, `TIME` or `DATE_TIME`

Throws:
`IllegalArgumentException` - if an invalid value is specified

See Also: `getInputMode()`[204]

javax.microedition.lcdui
Display

Declaration

```
public class Display

Object
  |
  +--javax.microedition.lcdui.Display
```

Description

`Display` represents the manager of the display and input devices of the system. It includes methods for retrieving properties of the device and for requesting that objects be displayed on the device. Other methods that deal with device attributes are primarily used with `Canvas`$_{139}$ objects and are thus defined there instead of here.

There is exactly one instance of Display per `MIDlet`$_{444}$ and the application can get a reference to that instance by calling the `getDisplay()`$_{213}$ method. The application may call the `getDisplay()` method at any time during course of its execution. The `Display` object returned by all calls to `getDisplay()` will remain the same during this time.

A typical application will perform the following actions in response to calls to its `MIDlet` methods:

- **startApp** - the application is moving from the paused state to the active state. Initialization of objects needed while the application is active should be done. The application may call `setCurrent()`$_{214}$ for the first screen if that has not already been done. Note that `startApp()` can be called several times if `pauseApp()` has been called in between. This means that one-time initialization should not take place here but instead should occur within the `MIDlet's` constructor.
- **pauseApp** - the application may pause its threads. Also, if it is desirable to start with another screen when the application is re-activated, the new screen should be set with `setCurrent()`.
- **destroyApp** - the application should free resources, terminate threads, etc. The behavior of method calls on user interface objects after `destroyApp()` has returned is undefined.

The user interface objects that are shown on the display device are contained within a `Displayable`$_{218}$ object. At any time the application may have at most one `Displayable` object that it intends to be shown on the display device and through which user interaction occurs. This `Displayable` is referred to as the *current* `Displayable`.

The `Display` class has a `setCurrent()`$_{214}$ method for setting the current `Displayable` and a `getCurrent()`$_{213}$ method for retrieving the current `Displayable`. The application has control over its current `Displayable` and may call `setCurrent()` at any time. Typically, the application will change the current `Displayable` in response to some user action. This is not always the case, however. Another thread may change the current `Displayable` in response to some other stimulus. The current `Displayable` will also be changed when the timer for an `Alert`$_{128}$ elapses.

The application's current `Displayable` may not physically be drawn on the screen, nor will user events (such as keystrokes) that occur necessarily be directed to the current `Displayable`. This may occur because of the presence of other `MIDlet` applications running simultaneously on the same device.

An application is said to be in the *foreground* if its current `Displayable` is actually visible on the display device and if user input device events will be delivered to it. If the application is not in the foreground, it lacks access to both the display and input devices, and it is said to be in the *background*. The policy for allocation of

these devices to different `MIDlet` applications is outside the scope of this specification and is under the control of an external agent referred to as the *application management software*.

As mentioned above, the application still has a notion of its current `Displayable` even if it is in the background. The current `Displayable` is significant, even for background applications, because the current `Displayable` is always the one that will be shown the next time the application is brought into the foreground. The application can determine whether a `Displayable` is actually visible on the display by calling `isShown()`$_{220}$. In the case of `Canvas`, the `showNotify()`$_{154}$ and `hideNotify()`$_{149}$ methods are called when the `Canvas` is made visible and is hidden, respectively.

Each `MIDlet` application has its own current `Displayable`. This means that the `getCurrent()`$_{213}$ method returns the `MIDlet's` current `Displayable`, regardless of the `MIDlet's` foreground/background state. For example, suppose a `MIDlet` running in the foreground has current `Displayable` *F*, and a `MIDlet` running in the background has current `Displayable` *B*. When the foreground `MIDlet` calls `getCurrent()`, it will return *F*, and when the background `MIDlet` calls `getCurrent()`, it will return *B*. Furthermore, if either `MIDlet` changes its current `Displayable` by calling `setCurrent()`, this will not affect the any other `MIDlet's` current `Displayable`.

It is possible for `getCurrent()` to return `null`. This may occur at startup time, before the `MIDlet` application has called `setCurrent()` on its first screen. The `getCurrent()` method will never return a reference to a `Displayable` object that was not passed in a prior call to `setCurrent()` call by this `MIDlet`.

System Screens
Typically, the current screen of the foreground `MIDlet` will be visible on the display. However, under certain circumstances, the system may create a screen that temporarily obscures the application's current screen. These screens are referred to as *system screens*. This may occur if the system needs to show a menu of commands or if the system requires the user to edit text on a separate screen instead of within a text field inside a `Form`. Even though the system screen obscures the application's screen, the notion of the current screen does not change. In particular, while a system screen is visible, a call to `getCurrent()` will return the application's current screen, not the system screen. The value returned by `isShown()` is `false` while the current `Displayable` is obscured by a system screen.

If system screen obscures a canvas, its `hideNotify()` method is called. When the system screen is removed, restoring the canvas, its `showNotify()` method and then its `paint()` method are called. If the system screen was used by the user to issue a command, the `commandAction()` method is called after `showNotify()` is called.

This class contains methods to retrieve the prevailing foreground and background colors of the high-level user interface. These methods are useful for creating `CustomItem` objects that match the user interface of other items and for creating user interfaces within `Canvas` that match the user interface of the rest of the system. Implementations are not restricted to using foreground and background colors in their user interfaces (for example, they might use highlight and shadow colors for a beveling effect) but the colors returned are those that match reasonably well with the implementation's color scheme. An application implementing a custom item should use the background color to clear its region and then paint text and geometric graphics (lines, arcs, rectangles) in the foreground color.

Since: MIDP 1.0

Member Summary

Fields

Member Summary	
static int	ALERT[207]
static int	CHOICE_GROUP_ELEMENT[208]
static int	COLOR_BACKGROUND[208]
static int	COLOR_BORDER[208]
static int	COLOR_FOREGROUND[208]
static int	COLOR_HIGHLIGHTED_BACKGROUND[209]
static int	COLOR_HIGHLIGHTED_BORDER[209]
static int	COLOR_HIGHLIGHTED_FOREGROUND[209]
static int	LIST_ELEMENT[209]
Methods	
void	callSerially(Runnable r)[210]
boolean	flashBacklight(int duration)[210]
int	getBestImageHeight(int imageType)[211]
int	getBestImageWidth(int imageType)[212]
int	getBorderStyle(boolean highlighted)[212]
int	getColor(int colorSpecifier)[212]
Displayable	getCurrent()[213]
static Display	getDisplay(javax.microedition.midlet.MIDlet m)[213]
boolean	isColor()[213]
int	numAlphaLevels()[213]
int	numColors()[214]
void	setCurrent(Alert alert, Displayable nextDisplayable)[214]
void	setCurrent(Displayable nextDisplayable)[214]
void	setCurrentItem(Item item)[216]
boolean	vibrate(int duration)[216]

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Fields

ALERT

Declaration:
`public static final int ALERT`

Description:
Image type for `Alert` image.

The value of `ALERT` is `3`.

Since: MIDP 2.0

See Also: `getBestImageWidth(int)`[212], `getBestImageHeight(int)`[211]

CHOICE_GROUP_ELEMENT

Declaration:
`public static final int CHOICE_GROUP_ELEMENT`

Description:
Image type for `ChoiceGroup` element image.

The value of `CHOICE_GROUP_ELEMENT` is `2`.

Since: MIDP 2.0

See Also: `getBestImageWidth(int)`[212], `getBestImageHeight(int)`[211]

COLOR_BACKGROUND

Declaration:
`public static final int COLOR_BACKGROUND`

Description:
A color specifier for use with `getColor`. `COLOR_BACKGROUND` specifies the background color of the screen. The background color will always contrast with the foreground color.

`COLOR_BACKGROUND` has the value `0`.

Since: MIDP 2.0

See Also: `getColor(int)`[212]

COLOR_BORDER

Declaration:
`public static final int COLOR_BORDER`

Description:
A color specifier for use with `getColor`. `COLOR_BORDER` identifies the color for boxes and borders when the object is to be drawn in a non-highlighted state. The border color is intended to be used with the background color and will contrast with it. The application should draw its borders using the stroke style returned by `getBorderStyle()`.

`COLOR_BORDER` has the value `4`.

Since: MIDP 2.0

See Also: `getColor(int)`[212]

COLOR_FOREGROUND

Declaration:
`public static final int COLOR_FOREGROUND`

Description:
A color specifier for use with `getColor`. `COLOR_FOREGROUND` specifies the foreground color, for text characters and simple graphics on the screen. Static text or user-editable text should be drawn with the foreground color. The foreground color will always constrast with background color.

`COLOR_FOREGROUND` has the value `1`.

Since: MIDP 2.0

See Also: `getColor(int)`[212]

COLOR_HIGHLIGHTED_BACKGROUND

Declaration:
`public static final int COLOR_HIGHLIGHTED_BACKGROUND`

Description:
A color specifier for use with `getColor`. `COLOR_HIGHLIGHTED_BACKGROUND` identifies the color for the focus, or focus highlight, when it is drawn as a filled in rectangle. The highlighted background will always constrast with the highlighted foreground.

`COLOR_HIGHLIGHTED_BACKGROUND` has the value `2`.

Since: MIDP 2.0

See Also: `getColor(int)` 212

COLOR_HIGHLIGHTED_BORDER

Declaration:
`public static final int COLOR_HIGHLIGHTED_BORDER`

Description:
A color specifier for use with `getColor`. `COLOR_HIGHLIGHTED_BORDER` identifies the color for boxes and borders when the object is to be drawn in a highlighted state. The highlighted border color is intended to be used with the background color (not the highlighted background color) and will contrast with it. The application should draw its borders using the stroke style returned `by getBorderStyle()`.

`COLOR_HIGHLIGHTED_BORDER` has the value `5`.

Since: MIDP 2.0

See Also: `getColor(int)` 212

COLOR_HIGHLIGHTED_FOREGROUND

Declaration:
`public static final int COLOR_HIGHLIGHTED_FOREGROUND`

Description:
A color specifier for use with `getColor`. `COLOR_HIGHLIGHTED_FOREGROUND` identifies the color for text characters and simple graphics when they are highlighted. Highlighted foreground is the color to be used to draw the highlighted text and graphics against the highlighted background. The highlighted foreground will always constrast with the highlighted background.

`COLOR_HIGHLIGHTED_FOREGROUND` has the value `3`.

Since: MIDP 2.0

See Also: `getColor(int)` 212

LIST_ELEMENT

Declaration:
`public static final int LIST_ELEMENT`

Description:
Image type for `List` element image.

The value of `LIST_ELEMENT` is `1`.

Since: MIDP 2.0

See Also: `getBestImageWidth(int)`[212], `getBestImageHeight(int)`[211]

Methods

callSerially(Runnable)

Declaration:
```
public void callSerially(Runnable r)
```

Description:
Causes the `Runnable` object `r` to have its `run()` method called later, serialized with the event stream, soon after completion of the repaint cycle. As noted in the Event Handling section of the package summary, the methods that deliver event notifications to the application are all called serially. The call to `r.run()` will be serialized along with the event calls into the application. The `run()` method will be called exactly once for each call to `callSerially()`. Calls to `run()` will occur in the order in which they were requested by calls to `callSerially()`.

If the current `Displayable` is a `Canvas` that has a repaint pending at the time of a call to `callSerially()`, the `paint()` method of the `Canvas` will be called and will return, and a buffer switch will occur (if double buffering is in effect), before the `run()` method of the `Runnable` is called. If the current `Displayable` contains one or more `CustomItems` that have repaints pending at the time of a call to `callSerially()`, the `paint()` methods of the `CustomItems` will be called and will return before the `run()` method of the `Runnable` is called. Calls to the `run()` method will occur in a timely fashion, but they are not guaranteed to occur immediately after the repaint cycle finishes, or even before the next event is delivered.

The `callSerially()` method may be called from any thread. The call to the `run()` method will occur independently of the call to `callSerially()`. In particular, `callSerially()` will *never* block waiting for `r.run()` to return.

As with other callbacks, the call to `r.run()` must return quickly. If it is necessary to perform a long-running operation, it may be initiated from within the `run()` method. The operation itself should be performed within another thread, allowing `run()` to return.

The `callSerially()` facility may be used by applications to run an animation that is properly synchronized with the repaint cycle. A typical application will set up a frame to be displayed and then call `repaint()`. The application must then wait until the frame is actually displayed, after which the setup for the next frame may occur. The call to `run()` notifies the application that the previous frame has finished painting. The example below shows `callSerially()` being used for this purpose.

Parameters:
`r` - instance of interface `Runnable` to be called

flashBacklight(int)

Declaration:
```
public boolean flashBacklight(int duration)
```

Description:
Requests a flashing effect for the device's backlight. The flashing effect is intended to be used to attract the user's attention or as a special effect for games. Examples of flashing are cycling the backlight on and off

```
class Animation extends Canvas
   implements Runnable {
// paint the current frame
void paint(Graphics g) { ... }
  Display display; // the display for the application
  void paint(Graphics g) { ... } // paint the current frame
  void startAnimation() {
     // set up initial frame
     repaint();
     display.callSerially(this);
  }
  // called after previous repaint is finished
  void run() {
     if ( /* there are more frames */ ) {
        // set up the next frame
        repaint();
        display.callSerially(this);
     }
  }
}
```

or from dim to bright repeatedly. The return value indicates if the flashing of the backlight can be controlled by the application.

The flashing effect occurs for the requested duration, or it is switched off if the requested duration is zero. This method returns immediately; that is, it must not block the caller while the flashing effect is running.

Calls to this method are honored only if the `Display` is in the foreground. This method MUST perform no action and return `false` if the `Display` is in the background.

The device MAY limit or override the duration. For devices that do not include a controllable backlight, calls to this method return `false`.

Parameters:

`duration` - the number of milliseconds the backlight should be flashed, or zero if the flashing should be stopped

Returns: `true` if the backlight can be controlled by the application and this display is in the foreground, `false` otherwise

Throws:

`IllegalArgumentException` - if `duration` is negative

Since: MIDP 2.0

getBestImageHeight(int)

Declaration:

```
public int getBestImageHeight(int imageType)
```

Description:

Returns the best image height for a given image type. The image type must be one of `LIST_ELEMENT`[209], `CHOICE_GROUP_ELEMENT`[208], or `ALERT`[207].

Parameters:

`imageType` - the image type

Returns: the best image height for the image type, may be zero if there is no best size; must not be negative

Throws:
`IllegalArgumentException` - if `imageType` is illegal

Since: MIDP 2.0

getBestImageWidth(int)

Declaration:
```
public int getBestImageWidth(int imageType)
```

Description:
Returns the best image width for a given image type. The image type must be one of `LIST_ELEMENT`[209], `CHOICE_GROUP_ELEMENT`[208], or `ALERT`[207].

Parameters:
`imageType` - the image type

Returns: the best image width for the image type, may be zero if there is no best size; must not be negative

Throws:
`IllegalArgumentException` - if `imageType` is illegal

Since: MIDP 2.0

getBorderStyle(boolean)

Declaration:
```
public int getBorderStyle(boolean highlighted)
```

Description:
Returns the stroke style used for border drawing depending on the state of the component (highlighted/non-highlighted). For example, on a monochrome system, the border around a non-highlighted item might be drawn with a `DOTTED` stroke style while the border around a highlighted item might be drawn with a `SOLID` stroke style.

Parameters:
`highlighted` - `true` if the border style being requested is for the highlighted state, `false` if the border style being requested is for the non-highlighted state

Returns: `Graphics.DOTTED`[253] or `Graphics.SOLID`[254]

Since: MIDP 2.0

getColor(int)

Declaration:
```
public int getColor(int colorSpecifier)
```

Description:
Returns one of the colors from the high level user interface color scheme, in the form `0x00RRGGBB` based on the `colorSpecifier` passed in.

Parameters:
`colorSpecifier` - the predefined color specifier; must be one of `COLOR_BACKGROUND`[208], `COLOR_FOREGROUND`[208], `COLOR_HIGHLIGHTED_BACKGROUND`[209],

`COLOR_HIGHLIGHTED_FOREGROUND`[209], `COLOR_BORDER`[208], or `COLOR_HIGHLIGHTED_BORDER`[209]

Returns: color in the form of `0x00RRGGBB`

Throws:
`IllegalArgumentException` - if `colorSpecifier` is not a valid color specifier

Since: MIDP 2.0

getCurrent()

Declaration:
```
public javax.microedition.lcdui.Displayable[218] getCurrent()
```

Description:
Gets the current `Displayable` object for this `MIDlet`. The `Displayable` object returned may not actually be visible on the display if the `MIDlet` is running in the background, or if the `Displayable` is obscured by a system screen. The `Displayable.isShown()`[220] method may be called to determine whether the `Displayable` is actually visible on the display.

The value returned by `getCurrent()` may be `null`. This occurs after the application has been initialized but before the first call to `setCurrent()`.

Returns: the `MIDlet's` current `Displayable` object

See Also: `setCurrent(Displayable)`[214]

getDisplay(MIDlet)

Declaration:
```
public static javax.microedition.lcdui.Display[205]
            getDisplay(javax.microedition.midlet.MIDlet[444] m)
```

Description:
Gets the `Display` object that is unique to this `MIDlet`.

Parameters:
`m` - `MIDlet` of the application

Returns: the display object that application can use for its user interface

Throws:
`NullPointerException` - if `m` is `null`

isColor()

Declaration:
```
public boolean isColor()
```

Description:
Gets information about color support of the device.

Returns: `true` if the display supports color, `false` otherwise

numAlphaLevels()

Declaration:
```
public int numAlphaLevels()
```

Description:
Gets the number of alpha transparency levels supported by this implementation. The minimum legal return value is `2`, which indicates support for full transparency and full opacity and no blending. Return values greater than `2` indicate that alpha blending is supported. For further information, see Alpha Processing.

Returns: number of alpha levels supported

Since: MIDP 2.0

numColors()

Declaration:
```
public int numColors()
```

Description:
Gets the number of colors (if `isColor()` is `true`) or graylevels (if `isColor()` is `false`) that can be represented on the device.

Note that the number of colors for a black and white display is `2`.

Returns: number of colors

setCurrent(Alert, Displayable)

Declaration:
```
public void setCurrent(javax.microedition.lcdui.Alert128 alert,
            javax.microedition.lcdui.Displayable218 nextDisplayable)
```

Description:
Requests that this `Alert` be made current, and that `nextDisplayable` be made current after the `Alert` is dismissed. This call returns immediately regardless of the `Alert's` timeout value or whether it is a modal alert. The `nextDisplayable` must not be an `Alert`, and it must not be `null`.

The automatic advance to `nextDisplayable` occurs only when the `Alert's` default listener is present on the `Alert` when it is dismissed. See Alert Commands and Listeners for details.

In other respects, this method behaves identically to `setCurrent(Displayable)`214 .

Parameters:
`alert` - the alert to be shown

`nextDisplayable` - the `Displayable` to be shown after this alert is dismissed

Throws:
`NullPointerException` - if alert or `nextDisplayable` is `null`

`IllegalArgumentException` - if `nextDisplayable` is an `Alert`

See Also: `Alert`128, `getCurrent()`213

setCurrent(Displayable)

Declaration:
```
public void setCurrent(javax.microedition.lcdui.Displayable218 nextDisplayable)
```

Description:
Requests that a different `Displayable` object be made visible on the display. The change will typically not take effect immediately. It may be delayed so that it occurs between event delivery method calls, although it is not guaranteed to occur before the next event delivery method is called. The `setCurrent()` method returns immediately, without waiting for the change to take place. Because of

this delay, a call to `getCurrent()` shortly after a call to `setCurrent()` is unlikely to return the value passed to `setCurrent()`.

Calls to `setCurrent()` are not queued. A delayed request made by a `setCurrent()` call may be superseded by a subsequent call to `setCurrent()`. For example, if screen `S1` is current, then

```
d.setCurrent(S2);
d.setCurrent(S3);
```

may eventually result in `S3` being made current, bypassing `S2` entirely.

When a `MIDlet` application is first started, there is no current `Displayable` object. It is the responsibility of the application to ensure that a `Displayable` is visible and can interact with the user at all times. Therefore, the application should always call `setCurrent()` as part of its initialization.

The application may pass `null` as the argument to `setCurrent()`. This does not have the effect of setting the current `Displayable` to `null`; instead, the current `Displayable` remains unchanged. However, the application management software may interpret this call as a request from the application that it is requesting to be placed into the background. Similarly, if the application is in the background, passing a non-null reference to `setCurrent()` may be interpreted by the application management software as a request that the application is requesting to be brought to the foreground. The request should be considered to be made even if the current `Displayable` is passed to the `setCurrent()`. For example, the code

```
d.setCurrent(d.getCurrent());
```

generally will have no effect other than requesting that the application be brought to the foreground. These are only requests, and there is no requirement that the application management software comply with these requests in a timely fashion if at all.

If the `Displayable` passed to `setCurrent()` is an `Alert`*128*, the previously current `Displayable`, if any, is restored after the `Alert` has been dismissed. If there is a current `Displayable`, the effect is as if `setCurrent(Alert, getCurrent())` had been called. Note that this will result in an exception being thrown if the current `Displayable` is already an alert. If there is no current `Displayable` (which may occur at startup time) the implementation's previous state will be restored after the `Alert` has been dismissed. The automatic restoration of the previous `Displayable` or the previous state occurs only when the `Alert's` default listener is present on the `Alert` when it is dismissed. See Alert Commands and Listeners for details.

To specify the `Displayable` to be shown after an `Alert` is dismissed, the application should use the `setCurrent(Alert,`

If the application calls `setCurrent()` while a system screen is active, the effect may be delayed until after the system screen is dismissed. The implementation may choose to interpret `setCurrent()` in such a situation as a request to cancel the effect of the system screen, regardless of whether `setCurrent()` has been delayed.

Parameters:

`nextDisplayable` - the `Displayable` requested to be made current; `null` is allowed

See Also: `getCurrent()`*213*

setCurrentItem(Item)

Declaration:
```
public void setCurrentItem(javax.microedition.lcdui.Item287 item)
```

Description:
Requests that the `Displayable` that contains this `Item` be made current, scrolls the `Displayable` so that this `Item` is visible, and possibly assigns the focus to this `Item`. The containing `Displayable` is first made current as if `setCurrent(Displayable)`$_{214}$ had been called. When the containing `Displayable` becomes current, or if it is already current, it is scrolled if necessary so that the requested `Item` is made visible. Then, if the implementation supports the notion of input focus, and if the `Item` accepts the input focus, the input focus is assigned to the `Item`.

This method always returns immediately, without waiting for the switching of the `Displayable`, the scrolling, and the assignment of input focus to take place.

It is an error for the `Item` not to be contained within a container. It is also an error if the `Item` is contained within an `Alert`.

Parameters:
`item` - the item that should be made visible

Throws:
`IllegalStateException`$_{37}$ - if the item is not owned by a container

`IllegalStateException`$_{37}$ - if the item is owned by an `Alert`

`NullPointerException` - if `item` is `null`

Since: MIDP 2.0

vibrate(int)

Declaration:
```
public boolean vibrate(int duration)
```

Description:
Requests operation of the device's vibrator. The vibrator is intended to be used to attract the user's attention or as a special effect for games. The return value indicates if the vibrator can be controlled by the application.

This method switches on the vibrator for the requested duration, or switches it off if the requested duration is zero. If this method is called while the vibrator is still activated from a previous call, the request is interpreted as setting a new duration. It is not interpreted as adding additional time to the original request. This method returns immediately; that is, it must not block the caller while the vibrator is running.

Calls to this method are honored only if the `Display` is in the foreground. This method MUST perform no action and return `false` if the `Display` is in the background.

The device MAY limit or override the duration. For devices that do not include a controllable vibrator, calls to this method return `false`.

Parameters:
`duration` - the number of milliseconds the vibrator should be run, or zero if the vibrator should be turned off

Returns: `true` if the vibrator can be controlled by the application and this display is in the foreground, `false` otherwise

Throws:
`IllegalArgumentException` - if `duration` is negative

Since: MIDP 2.0

javax.microedition.lcdui
Displayable

Declaration

```
public abstract class Displayable

Object
  |
  +--javax.microedition.lcdui.Displayable
```

Direct Known Subclasses: `Canvas`[139], `Screen`[315]

Description

An object that has the capability of being placed on the display. A `Displayable` object may have a title, a ticker, zero or more commands and a listener associated with it. The contents displayed and their interaction with the user are defined by subclasses.

The title string may contain line breaks. The display of the title string must break accordingly. For example, if only a single line is available for a title and the string contains a line break then only the characters up to the line break are displayed.

Unless otherwise specified by a subclass, the default state of newly created `Displayable` objects is as follows:

- it is not visible on the `Display`;
- there is no `Ticker` associated with this `Displayable`;
- the title is `null`;
- there are no `Commands` present; and
- there is no `CommandListener` present.

Since: MIDP 1.0

Member Summary

Methods

void	addCommand(Command cmd)[219]
int	getHeight()[219]
Ticker	getTicker()[219]
java.lang.String	getTitle()[220]
int	getWidth()[220]
boolean	isShown()[220]
void	removeCommand(Command cmd)[220]
void	setCommandListener(CommandListener l)[221]
void	setTicker(Ticker ticker)[221]
void	setTitle(String s)[221]
protected void	sizeChanged(int w, int h)[222]

Inherited Member Summary
Methods inherited from class `Object` `equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Methods

addCommand(Command)

Declaration:
`public void` **`addCommand`**`(javax.microedition.lcdui.Command`$_{175}$ `cmd)`

Description:
Adds a command to the `Displayable`. The implementation may choose, for example, to add the command to any of the available soft buttons or place it in a menu. If the added command is already in the screen (tested by comparing the object references), the method has no effect. If the `Displayable` is actually visible on the display, and this call affects the set of visible commands, the implementation should update the display as soon as it is feasible to do so.

Parameters:
`cmd` - the command to be added

Throws:
`NullPointerException` - if `cmd` is `null`

getHeight()

Declaration:
`public int` **`getHeight`**`()`

Description:
Gets the height in pixels of the displayable area available to the application. The value returned is appropriate for the particular `Displayable` subclass. This value may depend on how the device uses the display and may be affected by the presence of a title, a ticker, or commands. This method returns the proper result at all times, even if the `Displayable` object has not yet been shown.

Returns: height of the area available to the application

Since: MIDP 2.0

getTicker()

Declaration:
`public` `javax.microedition.lcdui.Ticker`$_{345}$ **`getTicker`**`()`

Description:
Gets the ticker used by this `Displayable`.

Returns: ticker object used, or `null` if no ticker is present

Since: MIDP 2.0

See Also: `setTicker(Ticker)`221

getTitle()

Declaration:
```
public String getTitle()
```

Description:
Gets the title of the `Displayable`. Returns `null` if there is no title.

Returns: the title of the instance, or `null` if no title

Since: MIDP 2.0

See Also: `setTitle(String)`221

getWidth()

Declaration:
```
public int getWidth()
```

Description:
Gets the width in pixels of the displayable area available to the application. The value returned is appropriate for the particular `Displayable` subclass. This value may depend on how the device uses the display and may be affected by the presence of a title, a ticker, or commands. This method returns the proper result at all times, even if the `Displayable` object has not yet been shown.

Returns: width of the area available to the application

Since: MIDP 2.0

isShown()

Declaration:
```
public boolean isShown()
```

Description:
Checks if the `Displayable` is actually visible on the display. In order for a `Displayable` to be visible, all of the following must be true: the `Display's MIDlet` must be running in the foreground, the `Displayable` must be the `Display's` current screen, and the `Displayable` must not be obscured by a system screen.

Returns: `true` if the `Displayable` is currently visible

removeCommand(Command)

Declaration:
```
public void removeCommand(javax.microedition.lcdui.Command cmd)
```
175

Description:
Removes a command from the `Displayable`. If the command is not in the `Displayable` (tested by comparing the object references), the method has no effect. If the `Displayable` is actually visible on the display, and this call affects the set of visible commands, the implementation should update the display as soon as it is feasible to do so. If `cmd` is `null`, this method does nothing.

Parameters:
`cmd` - the command to be removed

setCommandListener(CommandListener)

Declaration:
```
public void setCommandListener(javax.microedition.lcdui.CommandListener183 l)
```

Description:
Sets a listener for Commands175 to this Displayable, replacing any previous CommandListener. A null reference is allowed and has the effect of removing any existing listener.

Parameters:
l - the new listener, or null.

setTicker(Ticker)

Declaration:
```
public void setTicker(javax.microedition.lcdui.Ticker345 ticker)
```

Description:
Sets a ticker for use with this Displayable, replacing any previous ticker. If null, removes the ticker object from this Displayable. The same ticker may be shared by several Displayable objects within an application. This is done by calling setTicker() with the same Ticker object on several different Displayable objects. If the Displayable is actually visible on the display, the implementation should update the display as soon as it is feasible to do so.

The existence of a ticker may affect the size of the area available for Displayable's contents. Addition, removal, or the setting of the ticker at runtime may dynamically change the size of the content area. This is most important to be aware of when using the Canvas class. If the available area does change, the application will be notified via a call to sizeChanged()222.

Parameters:
ticker - the ticker object used on this screen

Since: MIDP 2.0

See Also: getTicker()219

setTitle(String)

Declaration:
```
public void setTitle(String s)
```

Description:
Sets the title of the Displayable. If null is given, removes the title.

If the Displayable is actually visible on the display, the implementation should update the display as soon as it is feasible to do so.

The existence of a title may affect the size of the area available for Displayable content. Addition, removal, or the setting of the title text at runtime may dynamically change the size of the content area. This is most important to be aware of when using the Canvas class. If the available area does change, the application will be notified via a call to sizeChanged()222.

Parameters:
s - the new title, or null for no title

Since: MIDP 2.0

See Also: getTitle()220

sizeChanged(int, int)

Declaration:
```
protected void sizeChanged(int w, int h)
```

Description:
The implementation calls this method when the available area of the `Displayable` has been changed. The "available area" is the area of the display that may be occupied by the application's contents, such as `Items` in a `Form` or graphics within a `Canvas`. It does not include space occupied by a title, a ticker, command labels, scroll bars, system status area, etc. A size change can occur as a result of the addition, removal, or changed contents of any of these display features.

This method is called at least once before the `Displayable` is shown for the first time. If the size of a `Displayable` changes while it is visible, `sizeChanged` will be called. If the size of a `Displayable` changes while it is *not* visible, calls to `sizeChanged` may be deferred. If the size had changed while the `Displayable` was not visible, `sizeChanged` will be called at least once at the time the `Displayable` becomes visible once again.

The default implementation of this method in `Displayable` and its subclasses defined in this specification must be empty. This method is intended solely for being overridden by the application. This method is defined on `Displayable` even though applications are prohibited from creating direct subclasses of `Displayable`. It is defined here so that applications can override it in subclasses of `Canvas` and `Form`. This is useful for `Canvas` subclasses to tailor their graphics and for `Forms` to modify `Item` sizes and layout directives in order to fit their contents within the the available display area.

Parameters:
`w` - the new width in pixels of the available area

`h` - the new height in pixels of the available area

Since: MIDP 2.0

javax.microedition.lcdui
Font

Declaration

```
public final class Font

Object
  |
  +--javax.microedition.lcdui.Font
```

Description

The `Font` class represents fonts and font metrics. `Fonts` cannot be created by applications. Instead, applications query for fonts based on font attributes and the system will attempt to provide a font that matches the requested attributes as closely as possible.

A `Font's` attributes are style, size, and face. Values for attributes must be specified in terms of symbolic constants. Values for the style attribute may be combined using the bit-wise `OR` operator, whereas values for the other attributes may not be combined. For example, the value

`STYLE_BOLD | STYLE_ITALIC`

may be used to specify a bold-italic font; however

`SIZE_LARGE | SIZE_SMALL`

is illegal.

The values of these constants are arranged so that zero is valid for each attribute and can be used to specify a reasonable default font for the system. For clarity of programming, the following symbolic constants are provided and are defined to have values of zero:

- `STYLE_PLAIN`
- `SIZE_MEDIUM`
- `FACE_SYSTEM`

Values for other attributes are arranged to have disjoint bit patterns in order to raise errors if they are inadvertently misused (for example, using `FACE_PROPORTIONAL` where a style is required). However, the values for the different attributes are not intended to be combined with each other.

Since: MIDP 1.0

Member Summary

Fields

static int	FACE_MONOSPACE 224
static int	FACE_PROPORTIONAL 224
static int	FACE_SYSTEM 225
static int	FONT_INPUT_TEXT 225
static int	FONT_STATIC_TEXT 225
static int	SIZE_LARGE 225
static int	SIZE_MEDIUM 225
static int	SIZE_SMALL 226

Member Summary

static int	STYLE_BOLD[226]
static int	STYLE_ITALIC[226]
static int	STYLE_PLAIN[226]
static int	STYLE_UNDERLINED[226]

Methods

int	charsWidth(char[] ch, int offset, int length)[226]
int	charWidth(char ch)[227]
int	getBaselinePosition()[227]
static Font	getDefaultFont()[227]
int	getFace()[228]
static Font	getFont(int fontSpecifier)[228]
static Font	getFont(int face, int style, int size)[228]
int	getHeight()[228]
int	getSize()[229]
int	getStyle()[229]
boolean	isBold()[229]
boolean	isItalic()[229]
boolean	isPlain()[229]
boolean	isUnderlined()[230]
int	stringWidth(String str)[230]
int	substringWidth(String str, int offset, int len)[230]

Inherited Member Summary

Methods inherited from class `Object`

`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Fields

FACE_MONOSPACE

Declaration:
`public static final int FACE_MONOSPACE`

Description:
The “monospace” font face.

Value 32 is assigned to FACE_MONOSPACE.

FACE_PROPORTIONAL

Declaration:
`public static final int FACE_PROPORTIONAL`

Description:
The “proportional” font face.

Value `64` is assigned to `FACE_PROPORTIONAL`.

FACE_SYSTEM

Declaration:
`public static final int FACE_SYSTEM`

Description:
The “system” font face.

Value `0` is assigned to `FACE_SYSTEM`.

FONT_INPUT_TEXT

Declaration:
`public static final int FONT_INPUT_TEXT`

Description:
Font specifier used by the implementation to draw text input by a user. `FONT_INPUT_TEXT` has the value `1`.

Since: MIDP 2.0

See Also: `getFont(int)` *228*

FONT_STATIC_TEXT

Declaration:
`public static final int FONT_STATIC_TEXT`

Description:
Default font specifier used to draw Item and Screen contents. `FONT_STATIC_TEXT` has the value `0`.

Since: MIDP 2.0

See Also: `getFont(int)` *228*

SIZE_LARGE

Declaration:
`public static final int SIZE_LARGE`

Description:
The “large” system-dependent font size.

Value `16` is assigned to `SIZE_LARGE`.

SIZE_MEDIUM

Declaration:
`public static final int SIZE_MEDIUM`

Description:
The “medium” system-dependent font size.

Value `0` is assigned to `STYLE_MEDIUM`.

SIZE_SMALL

SIZE_SMALL

Declaration:
```
public static final int SIZE_SMALL
```

Description:
The “small” system-dependent font size.

Value `8` is assigned to `STYLE_SMALL`.

STYLE_BOLD

Declaration:
```
public static final int STYLE_BOLD
```

Description:
The bold style constant. This may be combined with the other style constants for mixed styles.

Value `1` is assigned to `STYLE_BOLD`.

STYLE_ITALIC

Declaration:
```
public static final int STYLE_ITALIC
```

Description:
The italicized style constant. This may be combined with the other style constants for mixed styles.

Value `2` is assigned to `STYLE_ITALIC`.

STYLE_PLAIN

Declaration:
```
public static final int STYLE_PLAIN
```

Description:
The plain style constant. This may be combined with the other style constants for mixed styles.

Value `0` is assigned to `STYLE_PLAIN`.

STYLE_UNDERLINED

Declaration:
```
public static final int STYLE_UNDERLINED
```

Description:
The underlined style constant. This may be combined with the other style constants for mixed styles.

Value `4` is assigned to `STYLE_UNDERLINED`.

Methods

charsWidth(char[], int, int)

Declaration:
```
public int charsWidth(char[] ch, int offset, int length)
```

Description:
Returns the advance width of the characters in `ch`, starting at the specified offset and for the specified number of characters (length). The advance width is the horizontal distance that would be occupied if the characters were to be drawn using this `Font`, including inter-character spacing following the characters necessary for proper positioning of subsequent text.

The `offset` and `length` parameters must specify a valid range of characters within the character array `ch`. The `offset` parameter must be within the range `[0..(ch.length)]`, inclusive. The `length` parameter must be a non-negative integer such that `(offset + length) <= ch.length`.

Parameters:
`ch` - the array of characters

`offset` - the index of the first character to measure

`length` - the number of characters to measure

Returns: the width of the character range

Throws:
`ArrayIndexOutOfBoundsException` - if `offset` and `length` specify an invalid range

`NullPointerException` - if `ch` is `null`

charWidth(char)

Declaration:
```
public int charWidth(char ch)
```

Description:
Gets the advance width of the specified character in this Font. The advance width is the horizontal distance that would be occupied if `ch` were to be drawn using this `Font`, including inter-character spacing following `ch` necessary for proper positioning of subsequent text.

Parameters:
`ch` - the character to be measured

Returns: the total advance width (a non-negative value)

getBaselinePosition()

Declaration:
```
public int getBaselinePosition()
```

Description:
Gets the distance in pixels from the top of the text to the text's baseline.

Returns: the distance in pixels from the top of the text to the text's baseline

getDefaultFont()

Declaration:
```
public static javax.microedition.lcdui.Font223 getDefaultFont()
```

Description:
Gets the default font of the system.

Returns: the default font

getFace()

Declaration:
`public int getFace()`

Description:
Gets the face of the font.

Returns: one of `FACE_SYSTEM`, `FACE_PROPORTIONAL`, `FACE_MONOSPACE`

getFont(int)

Declaration:
`public static javax.microedition.lcdui.Font`$_{223}$ `getFont(int fontSpecifier)`

Description:
Gets the `Font` used by the high level user interface for the `fontSpecifier` passed in. It should be used by subclasses of `CustomItem` and `Canvas` to match user interface on the device.

Parameters:
`fontSpecifier` - one of `FONT_INPUT_TEXT`, or `FONT_STATIC_TEXT`

Returns: font that corresponds to the passed in font specifier

Throws:
`IllegalArgumentException` - if `fontSpecifier` is not a valid fontSpecifier

Since: MIDP 2.0

getFont(int, int, int)

Declaration:
`public static javax.microedition.lcdui.Font`$_{223}$ `getFont(int face, int style, int size)`

Description:
Obtains an object representing a font having the specified face, style, and size. If a matching font does not exist, the system will attempt to provide the closest match. This method *always* returns a valid font object, even if it is not a close match to the request.

Parameters:
`face` - one of `FACE_SYSTEM`, `FACE_MONOSPACE`, or `FACE_PROPORTIONAL`

`style` - `STYLE_PLAIN`, or a combination of `STYLE_BOLD`, `STYLE_ITALIC`, and `STYLE_UNDERLINED`

`size` - one of `SIZE_SMALL`, `SIZE_MEDIUM`, or `SIZE_LARGE`

Returns: instance the nearest font found

Throws:
`IllegalArgumentException` - if `face`, `style`, or `size` are not legal values

getHeight()

Declaration:
`public int getHeight()`

Description:
Gets the standard height of a line of text in this font. This value includes sufficient spacing to ensure that lines of text painted this distance from anchor point to anchor point are spaced as intended by the font designer and the device. This extra space (leading) occurs below the text.

Returns: standard height of a line of text in this font (a non-negative value)

getSize()

Declaration:
`public int getSize()`

Description:
Gets the size of the font.

Returns: one of `SIZE_SMALL`, `SIZE_MEDIUM`, `SIZE_LARGE`

getStyle()

Declaration:
`public int getStyle()`

Description:
Gets the style of the font. The value is an `OR'ed` combination of `STYLE_BOLD`, `STYLE_ITALIC`, and `STYLE_UNDERLINED`; or the value is zero (`STYLE_PLAIN`).

Returns: style of the current font

See Also: `isPlain()`*229*, `isBold()`*229*, `isItalic()`*229*

isBold()

Declaration:
`public boolean isBold()`

Description:
Returns `true` if the font is bold.

Returns: `true` if font is bold

See Also: `getStyle()`*229*

isItalic()

Declaration:
`public boolean isItalic()`

Description:
Returns `true` if the font is italic.

Returns: `true` if font is italic

See Also: `getStyle()`*229*

isPlain()

Declaration:
`public boolean isPlain()`

Description:
Returns `true` if the font is plain.

Returns: `true` if font is plain

See Also: `getStyle()`*229*

isUnderlined()

Declaration:
```
public boolean isUnderlined()
```

Description:
Returns `true` if the font is underlined.

Returns: `true` if font is underlined

See Also: `getStyle()`*229*

stringWidth(String)

Declaration:
```
public int stringWidth(String str)
```

Description:
Gets the total advance width for showing the specified `String` in this `Font`. The advance width is the horizontal distance that would be occupied if `str` were to be drawn using this `Font`, including inter-character spacing following `str` necessary for proper positioning of subsequent text.

Parameters:
`str` - the `String` to be measured

Returns: the total advance width

Throws:
`NullPointerException` - if `str` is `null`

substringWidth(String, int, int)

Declaration:
```
public int substringWidth(String str, int offset, int len)
```

Description:
Gets the total advance width for showing the specified substring in this `Font`. The advance width is the horizontal distance that would be occupied if the substring were to be drawn using this `Font`, including inter-character spacing following the substring necessary for proper positioning of subsequent text.

The `offset` and `len` parameters must specify a valid range of characters within `str`. The `offset` parameter must be within the range `[0..(str.length())]`, inclusive. The `len` parameter must be a non-negative integer such that `(offset + len) <= str.length()`.

Parameters:
`str` - the `String` to be measured

`offset` - zero-based index of first character in the substring

`len` - length of the substring

Returns: the total advance width

Throws:
`StringIndexOutOfBoundsException` - if `offset` and `length` specify an invalid range

`NullPointerException` - if `str` is `null`

javax.microedition.lcdui
Form

Declaration

```
public class Form extends Screen315

Object
  |
  +--javax.microedition.lcdui.Displayable218
        |
        +--javax.microedition.lcdui.Screen315
              |
              +--javax.microedition.lcdui.Form
```

Description

A `Form` is a `Screen` that contains an arbitrary mixture of items: images, read-only text fields, editable text fields, editable date fields, gauges, choice groups, and custom items. In general, any subclass of the `Item`287 class may be contained within a form. The implementation handles layout, traversal, and scrolling. The entire contents of the `Form` scrolls together.

Item Management

The items contained within a `Form` may be edited using append, delete, insert, and set methods. `Items` within a `Form` are referred to by their indexes, which are consecutive integers in the range from zero to `size()-1`, with zero referring to the first item and `size()-1` to the last item.

An item may be placed within at most one `Form`. If the application attempts to place an item into a `Form`, and the item is already owned by this or another `Form`, an `IllegalStateException` is thrown. The application must remove the item from its currently containing `Form` before inserting it into the new `Form`.

If the `Form` is visible on the display when changes to its contents are requested by the application, updates to the display take place as soon as it is feasible for the implementation to do so. Applications need not take any special action to refresh a `Form's` display after its contents have been modified.

Layout

Layout policy in `Form` is organized around rows. Rows are typically related to the width of the screen, respective of margins, scroll bars, and such. All rows in a particular `Form` will have the same width. Rows do not vary in width based on the `Items` contained within the `Form`, although they may all change width in certain circumstances, such as when a scroll bar needs to be added or removed. `Forms` generally do not scroll horizontally.

`Forms` grow vertically and scroll vertically as necessary. The height of a `Form` varies depending upon the number of rows and the height of each row. The height of each row is determined by the items that are positioned on that row. Rows need not all have the same height. Implementations may also vary row heights to provide proper padding or vertical alignment of `Item` labels.

An implementation may choose to lay out `Items` in a left-to-right or right-to-left direction depending upon the language conventions in use. The same choice of layout direction must apply to all rows within a particular `Form`.

Prior to the start of the layout algorithm, the `Form` is considered to have one empty row at the top. The layout algorithm considers each Item in turn, starting at `Item` zero and proceeding in order through each `Item` until

the last `Item` in the `Form` has been processed. If the layout direction (as described above) is left-to-right, the beginning of the row is the left edge of the `Form`. If the layout direction is right-to-left, the beginning of the row is the right edge of the `Form`. `Items` are laid out at the beginning of each row, proceeding across each row in the chosen layout direction, packing as many `Items` onto each row as will fit, unless a condition occurs that causes the packing of a row to be terminated early. A new row is then added, and `Items` are packed onto it as described above. `Items` are packed onto rows, and new rows are added below existing rows as necessary until all `Items` have been processed by the layout algorithm.

The layout algorithm has a concept of a *current alignment*. It can have the value `LAYOUT_LEFT`, `LAYOUT_CENTER`, or `LAYOUT_RIGHT`. The value of the current alignment at the start of the layout algorithm depends upon the layout direction in effect for this `Form`. If the layout direction is left-to-right, the initial alignment value must be `LAYOUT_LEFT`. If the layout direction is right-to-left, the initial alignment value must be `LAYOUT_RIGHT`. The current alignment changes when the layout algorithm encounters an `Item` that has one of the layout directives `LAYOUT_LEFT`, `LAYOUT_CENTER`, or `LAYOUT_RIGHT`. If none of these directives is present on an `Item`, the current layout directive does not change. This rule has the effect of grouping the contents of the `Form` into sequences of consecutive `Items` sharing an alignment value. The alignment value of each `Item` is maintained internally to the `Form` and does not affect the `Items'` layout value as reported by the `Item.getLayout`$_{295}$ method.

The layout algorithm generally attempts to place an item on the same row as the previous item, unless certain conditions occur that cause a “row break.” When there is a row break, the current item will be placed at the beginning of a new row instead of being placed after the previous item, even if there is room.

A row break occurs before an item if any of the following conditions occurs:

- the previous item has a row break after it;
- it has the `LAYOUT_NEWLINE_BEFORE` directive; or
- it is a `StringItem` whose contents starts with “\n”;
- it is a `ChoiceGroup`, `DateField`, `Gauge`, or a `TextField`, and the `LAYOUT_2` directive is not set; or
- this `Item` has a `LAYOUT_LEFT`, `LAYOUT_CENTER`, or `LAYOUT_RIGHT` directive that differs from the `Form's` current alignment.

A row break occurs after an item if any of the following conditions occurs:

- it is a `StringItem` whose contents ends with “\n”; or
- it has the `LAYOUT_NEWLINE_AFTER` directive; or
- it is a `ChoiceGroup`, `DateField`, `Gauge`, or a `TextField`, and the `LAYOUT_2` directive is not set.

The presence of the `LAYOUT_NEWLINE_BEFORE` or `LAYOUT_NEWLINE_AFTER` directive does not cause an additional row break if there is one already present. For example, if a `LAYOUT_NEWLINE_BEFORE` directive appears on a `StringItem` whose contents starts with “\n”, there is only a single row break. A similar rule applies with a trailing “\n” and `LAYOUT_NEWLINE_AFTER`. Also, there is only a single row break if an item has the `LAYOUT_NEWLINE_AFTER` directive and the next item has the `LAYOUT_NEWLINE_BEFORE` directive. However, the presence of consecutive “\n” characters, either within a single `StringItem` or in adjacent `StringItems`, will cause as many row breaks as there are “\n” characters. This will cause empty rows to be present. The height of an empty row is determined by the prevailing font height of the `StringItem` within which the “\n” that ends the row occurs.

Implementations may provide additional conditions under which a row break occurs. For example, an implementation’s layout policy may lay out labels specially, implicitly causing a break before every `Item` that has a label. Or, as another example, a particular implementation’s user interface style may dictate that a

DateField item always appears on a row by itself. In this case, this implementation may cause row breaks to occur both before and after every `DateField` item.

Given two items with adjacent `Form` indexes, if none of the specified or implementation-specific conditions for a row break between them occurs, and if space permits, these items should be placed on the same row.

When packing `Items` onto a row, the width of the item is compared with the remaining space on the row. For this purpose, the width used is the `Item's` preferred width, unless the `Item` has the `LAYOUT_SHRINK` directive, in which case the `Item's` minimum width is used. If the `Item` is too wide to fit in the space remaining on the row, the row is considered to be full, a new row is added beneath this one, and the `Item` is laid out on this new row.

Once the contents of a row have been determined, the space available on the row is distributed by expanding items and by adding space between items. If any items on this row have the `LAYOUT_SHRINK` directive (that is, they are shrinkable), space is first distributed to these items. Space is distributed to each of these items proportionally to the difference between the each `Item's` preferred size and its minimum size. At this stage, no shrinkable item is expanded beyond its preferred width.

For example, consider a row that has `30` pixels of space available and that has two shrinkable items `A` and `B`. Item `A's` preferred size is `15` and its minimum size is `10`. Item `B's` preferred size is `30` and its minimum size is `20`. The difference between `A's` preferred and minimum size is `5`, and `B's` difference is `10`. The `30` pixels are distributed to these items proportionally to these differences. Therefore, `10` pixels are distributed to item `A` and `20` pixels to item `B`.

If after expanding all the shrinkable items to their preferred widths, there is still space left on the row, this remaining space is distributed equally among the Items that have the `LAYOUT_EXPAND` directive (the stretchable `Items`). The presence of any stretchable items on a row will cause the `Items` on this row to occupy the full width of the row.

If there are no stretchable items on this row, and there is still space available on this row, the `Items` are packed as tightly as possible and are placed on the row according to the alignment value shared by the `Items` on this row. (Since changing the current alignment causes a row break, all `Items` on the same row must share the same alignment value.) If the alignment value is `LAYOUT_LEFT`, the `Items` are positioned at the left end of the row and the remaining space is placed at the right end of the row. If the alignment value is `LAYOUT_RIGHT`, the `Items` are positioned at the right end of the row and the remaining space is placed at the left end of the row. If the alignment value is `LAYOUT_CENTER`, the `Items` are positioned in the middle of the row such that the remaining space on the row is divided evenly between the left and right ends of the row.

Given the set of items on a particular row, the heights of these `Items` are inspected. For each `Item`, the height that is used is the preferred height, unless the `Item` has the `LAYOUT_VSHRINK` directive, in which case the `Item's` minimum height is used. The height of the tallest `Item` determines the height of the row. `Items` that have the `LAYOUT_VSHRINK` directive are expanded to their preferred height or to the height of the row, whichever is smaller. `Items` that are still shorter than the row height and that have the `LAYOUT_VEXPAND` directive will expand to the height of the row. The `LAYOUT_VEXPAND` directive on an item will never increase the height of a row.

Remaining `Items` shorter than the row height will be positioned vertically within the row using the `LAYOUT_TOP`, `LAYOUT_BOTTOM`, and `LAYOUT_VCENTER` directives. If no vertical layout directive is specified, the item must be aligned along the bottom of the row.

`StringItems` are treated specially in the above algorithm. If the contents of a `StringItem` (its string value, exclusive of its label) contain a newline character (“\n”), the string should be split at that point and the remainder laid out starting on the next row.

If one or both dimensions of the preferred size of a `StringItem` have been locked, the `StringItem` is wrapped to fit that width and height and is treated as a rectangle whose minimum and preferred width and

height are the width and height of this rectangle. In this case, the LAYOUT_SHRINK, LAYOUT_EXPAND, and LAYOUT_VEXPAND directives are ignored.

If both dimensions of the preferred size of a StringItem are unlocked, the text from the StringItem may be wrapped across multiple rows. At the point in the layout algorithm where the width of the Item is compared to the remaining space on the row, as much text is taken from the beginning of the StringItem as will fit onto the current row. The contents of this row are then positioned according to the current alignment value. The remainder of the text in the StringItem is line-wrapped to the full width of as many new rows as are necessary to accommodate the text. Each full row is positioned according to the current alignment value. The last line of the text might leave space available on its row. If there is no row break following this StringItem, subsequent Items are packed into the remaining space and the contents of the row are positioned according to the current alignment value. This rule has the effect of displaying the contents of a StringItem as a paragraph of text set flush-left, flush-right, or centered, depending upon whether the current alignment value is LAYOUT_LEFT, LAYOUT_RIGHT, or LAYOUT_CENTER, respectively. The preferred width and height of a StringItem wrapped across multiple rows, as reported by the Item.getPreferredWidth 296 and Item.getPreferredHeight 296 methods, describe the width and height of the bounding rectangle of the wrapped text.

ImageItems are also treated specially by the above algorithm. The foregoing rules concerning the horizontal alignment value and the LAYOUT_LEFT, LAYOUT_RIGHT, and LAYOUT_CENTER directives, apply to ImageItems only when the LAYOUT_2 directive is also present on that item. If the LAYOUT_2 directive is not present on an ImageItem, the behavior of the LAYOUT_LEFT, LAYOUT_RIGHT, and LAYOUT_CENTER directives is implementation-specific.

A Form's layout is recomputed automatically as necessary. This may occur because of a change in an Item's size caused by a change in its contents or because of a request by the application to change the Item's preferred size. It may also occur if an Item's layout directives are changed by the application. The application does not need to perform any specific action to cause the Form's layout to be updated.

Line Breaks and Wrapping

For all cases where text is wrapped, line breaks must occur at each newline character ('\n' = Unicode 'U+000A'). If space does not permit the full text to be displayed it is truncated at line breaks. If there are no suitable line breaks, it is recommended that implementations break text at word boundaries. If there are no word boundaries, it is recommended that implementations break text at character boundaries.

Labels that contain line breaks may be truncated at the line break and cause the rest of the label not to be shown.

User Interaction

When a Form is present on the display the user can interact with it and its Items indefinitely (for instance, traversing from Item to Item and possibly scrolling). These traversing and scrolling operations do not cause application-visible events. The system notifies the application when the user modifies the state of an interactive Item contained within the Form. This notification is accomplished by calling the itemStateChanged() 301 method of the listener declared to the Form with the setItemStateListener() 239 method.

As with other Displayable objects, a Form can declare commands 175 and declare a command listener with the setCommandListener() 221 method. CommandListener 183 objects are distinct from ItemStateListener 301 objects, and they are declared and invoked separately.

Notes for Application Developers

- Although this class allows creation of arbitrary combination of components the application developers should keep the small screen size in mind. Form is designed to contain a *small number of closely related* UI

elements.

- If the number of items does not fit on the screen, the implementation may choose to make it scrollable or to fold some components so that a separate screen appears when the element is edited.

Since: MIDP 1.0

See Also: `Item`[287]

Member Summary

Constructors

	`Form(String title)`[235]
	`Form(String title, Item items)`[236]

Methods

int	`append(Image img)`[236]
int	`append(Item item)`[236]
int	`append(String str)`[237]
void	`delete(int itemNum)`[237]
void	`deleteAll()`[237]
Item	`get(int itemNum)`[237]
int	`getHeight()`[238]
int	`getWidth()`[238]
void	`insert(int itemNum, Item item)`[238]
void	`set(int itemNum, Item item)`[239]
void	`setItemStateListener(ItemStateListener iListener)`[239]
int	`size()`[239]

Inherited Member Summary

Methods inherited from class `Displayable`[218]

`addCommand(Command)`[219], `getTicker()`[219], `getTitle()`[220], `isShown()`[220], `removeCommand(Command)`[220], `setCommandListener(CommandListener)`[221], `setTicker(Ticker)`[221], `setTitle(String)`[221], `sizeChanged(int, int)`[222]

Methods inherited from class `Object`

`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait()`, `wait()`, `wait()`

Constructors

Form(String)

Declaration:

```
public Form(String title)
```

Form(String, Item[])

Description:
Creates a new, empty `Form`.

Parameters:
`title` - the `Form's` title, or `null` for no title

Form(String, Item[])

Declaration:
`public Form(String title, javax.microedition.lcdui.Item[]`$_{287}$ `items)`

Description:
Creates a new `Form` with the specified contents. This is identical to creating an empty `Form` and then using a set of `append` methods. The items array may be `null`, in which case the `Form` is created empty. If the items array is non-null, each element must be a valid `Item` not already contained within another `Form`.

Parameters:
`title` - the `Form's` title string

`items` - the array of items to be placed in the `Form`, or `null` if there are no items

Throws:
`IllegalStateException`$_{37}$ - if one of the items is already owned by another container

`NullPointerException` - if an element of the items array is `null`

Methods

append(Image)

Declaration:
`public int append(javax.microedition.lcdui.Image`$_{270}$ `img)`

Description:
Adds an item consisting of one `Image` to the `Form`. The effect of this method is identical to

```
append(new ImageItem(null, img, ImageItem.LAYOUT_DEFAULT, null))
```

Parameters:
`img` - the image to be added

Returns: the assigned index of the `Item`

Throws:
`NullPointerException` - if `img` is `null`

append(Item)

Declaration:
`public int append(javax.microedition.lcdui.Item`$_{287}$ `item)`

Description:
Adds an `Item` into the `Form`. The newly added `Item` becomes the last `Item` in the `Form`, and the size of the `Form` grows by one.

Parameters:
`item` - the `Item`$_{287}$ to be added.

Returns: the assigned index of the `Item`

Throws:
`IllegalStateException`[37] - if the item is already owned by a container
`NullPointerException` - if item is `null`

append(String)

Declaration:
```
public int append(String str)
```

Description:
Adds an item consisting of one `String` to the `Form`. The effect of this method is identical to

```
append(new StringItem(null, str))
```

Parameters:
`str` - the `String` to be added

Returns: the assigned index of the `Item`

Throws:
`NullPointerException` - if str is `null`

delete(int)

Declaration:
```
public void delete(int itemNum)
```

Description:
Deletes the `Item` referenced by `itemNum`. The size of the `Form` shrinks by one. It is legal to delete all items from a `Form`. The `itemNum` parameter must be within the range `[0..size()-1]`, inclusive.

Parameters:
`itemNum` - the index of the item to be deleted

Throws:
`IndexOutOfBoundsException` - if `itemNum` is invalid

deleteAll()

Declaration:
```
public void deleteAll()
```

Description:
Deletes all the items from this `Form`, leaving it with zero items. This method does nothing if the `Form` is already empty.

Since: MIDP 2.0

get(int)

Declaration:
```
public javax.microedition.lcdui.Item[287] get(int itemNum)
```

Description:
Gets the item at given position. The contents of the `Form` are left unchanged. The `itemNum` parameter must be within the range `[0..size()-1]`, inclusive.

Parameters:
`itemNum` - the index of item

Returns: the item at the given position

Throws:
`IndexOutOfBoundsException` - if `itemNum` is invalid

getHeight()

Declaration:
```
public int getHeight()
```

Description:
Returns the height in pixels of the displayable area available for items. This value is the height of the form that can be displayed without scrolling. The value may depend on how the device uses the screen and may be affected by the presence or absence of the ticker, title, or commands.

Overrides: `getHeight`[219] in class `Displayable`[218]

Returns: the height of the displayable area of the `Form` in pixels

Since: MIDP 2.0

getWidth()

Declaration:
```
public int getWidth()
```

Description:
Returns the width in pixels of the displayable area available for items. The value may depend on how the device uses the screen and may be affected by the presence or absence of the ticker, title, or commands. The `Items` of the `Form` are laid out to fit within this width.

Overrides: `getWidth`[220] in class `Displayable`[218]

Returns: the width of the `Form` in pixels

Since: MIDP 2.0

insert(int, Item)

Declaration:
```
public void insert(int itemNum, javax.microedition.lcdui.Item[287] item)
```

Description:
Inserts an item into the `Form` just prior to the item specified. The size of the `Form` grows by one. The `itemNum` parameter must be within the range `[0..size()]`, inclusive. The index of the last item is `size()-1`, and so there is actually no item whose index is `size()`. If this value is used for `itemNum`, the new item is inserted immediately after the last item. In this case, the effect is identical to `append(Item)`[236].

The semantics are otherwise identical to `append(Item)`[236].

Parameters:
`itemNum` - the index where insertion is to occur

`item` - the item to be inserted

Throws:
`IndexOutOfBoundsException` - if `itemNum` is invalid

`IllegalStateException`[37] - if the item is already owned by a container

`NullPointerException` - if `item` is `null`

set(int, Item)

Declaration:
`public void set(int itemNum, javax.microedition.lcdui.Item`[287] `item)`

Description:
Sets the item referenced by `itemNum` to the specified item, replacing the previous item. The previous item is removed from this `Form`. The `itemNum` parameter must be within the range `[0..size()-1]`, inclusive.

The end result is equal to `insert(n, item); delete(n+1);`
although the implementation may optimize the repainting and usage of the array that stores the items.

Parameters:
`itemNum` - the index of the item to be replaced

`item` - the new item to be placed in the `Form`

Throws:
`IndexOutOfBoundsException` - if `itemNum` is invalid

`IllegalStateException`[37] - if the item is already owned by a container

`NullPointerException` - if `item` is `null`

setItemStateListener(ItemStateListener)

Declaration:
`public void setItemStateListener(javax.microedition.lcdui.ItemStateListener`[301] `iListener)`

Description:
Sets the `ItemStateListener` for the `Form`, replacing any previous `ItemStateListener`. If `iListener` is `null`, simply removes the previous `ItemStateListener`.

Parameters:
`iListener` - the new listener, or `null` to remove it

size()

Declaration:
`public int size()`

Description:
Gets the number of items in the `Form`.

Returns: the number of items

javax.microedition.lcdui
Gauge

Declaration

```
public class Gauge extends Item287

Object
  |
  +--javax.microedition.lcdui.Item287
        |
        +--javax.microedition.lcdui.Gauge
```

Description

Implements a graphical display, such as a bar graph, of an integer value. The `Gauge` contains a *current value* that lies between zero and the *maximum value*, inclusive. The application can control the current value and maximum value. The range of values specified by the application may be larger than the number of distinct visual states possible on the device, so more than one value may have the same visual representation.

For example, consider a `Gauge` object that has a range of values from zero to `99`, running on a device that displays the `Gauge's` approximate value using a set of one to ten bars. The device might show one bar for values zero through nine, two bars for values ten through `19`, three bars for values `20` through `29`, and so forth.

A `Gauge` may be interactive or non-interactive. Applications may set or retrieve the `Gauge's` value at any time regardless of the interaction mode. The implementation may change the visual appearance of the bar graph depending on whether the object is created in interactive mode.

In interactive mode, the user is allowed to modify the value. The user will always have the means to change the value up or down by one and may also have the means to change the value in greater increments. The user is prohibited from moving the value outside the established range. The expected behavior is that the application sets the initial value and then allows the user to modify the value thereafter. However, the application is not prohibited from modifying the value even while the user is interacting with it.

In many cases the only means for the user to modify the value will be to press a button to increase or decrease the value by one unit at a time. Therefore, applications should specify a range of no more than a few dozen values.

In non-interactive mode, the user is prohibited from modifying the value. Non-interactive mode is used to provide feedback to the user on the state of a long-running operation. One expected use of the non-interactive mode is as a “progress indicator” or “activity indicator” to give the user some feedback during a long-running operation. The application may update the value periodically using the `setValue()` method.

A non-interactive `Gauge` can have a definite or indefinite range. If a `Gauge` has definite range, it will have an integer value between zero and the maximum value set by the application, inclusive. The implementation will provide a graphical representation of this value such as described above.

A non-interactive `Gauge` that has indefinite range will exist in one of four states: continuous-idle, incremental-idle, continuous-running, or incremental-updating. These states are intended to indicate to the user that some level of activity is occurring. With incremental-updating, progress can be indicated to the user even though there is no known endpoint to the activity. With continuous-running, there is no progress that gets reported to the user and there is no known endpoint; continuous-running is merely a busy state indicator. The implementation should use a graphical display that shows this appropriately. The implementation may use different graphics for indefinite continuous gauges and indefinite incremental gauges. Because of this, separate idle states exist for each mode. For example, the implementation might show an hourglass or spinning watch in the continuous-

running state, but show an animation with different states, like a beach ball or candy-striped bar, in the incremental-updating state.

In the continuous-idle or incremental-idle state, the `Gauge` indicates that no activity is occurring. In the incremental-updating state, the `Gauge` indicates activity, but its graphical representation should be updated only when the application requests an update with a call to `setValue()`. In the continuous-running state, the `Gauge` indicates activity by showing an animation that runs continuously, without update requests from the application.

The values `CONTINUOUS_IDLE`, `INCREMENTAL_IDLE`, `CONTINUOUS_RUNNING`, and `INCREMENTAL_UPDATING` have their special meaning only when the `Gauge` is non-interactive and has been set to have indefinite range. They are treated as ordinary values if the `Gauge` is interactive or if it has been set to have a definite range.

An application using the `Gauge` as a progress indicator should typically also attach a `STOP`$_{180}$ command to the container containing the `Gauge` to allow the user to halt the operation in progress.

Notes for Application Developers
As mentioned above, a non-interactive `Gauge` may be used to give user feedback during a long-running operation. If the application can observe the progress of the operation as it proceeds to an endpoint known in advance, then the application should use a non-interactive `Gauge` with a definite range. For example, consider an application that is downloading a file known to be `20` kilobytes in size. The application could set the `Gauge's` maximum value to be `20` and set its value to the number of kilobytes downloaded so far. The user will be presented with a `Gauge` that shows the portion of the task completed at any given time.

If, on the other hand, the application is downloading a file of unknown size, it should use a non-interactive `Gauge` with indefinite range. Ideally, the application should call `setValue(INCREMENTAL_UPDATING)` periodically, perhaps each time its input buffer has filled. This will give the user an indication of the rate at which progress is occurring.

Finally, if the application is performing an operation but has no means of detecting progress, it should set a non-interactive `Gauge` to have indefinite range and set its value to `CONTINUOUS_RUNNING` or `CONTINUOUS_IDLE` as appropriate. For example, if the application has issued a request to a network server and is about to block waiting for the server to respond, it should set the `Gauge's` state to `CONTINUOUS_RUNNING` before awaiting the response, and it should set the state to `CONTINUOUS_IDLE` after it has received the response.

Since: MIDP 1.0

Member Summary

Fields

static int	`CONTINUOUS_IDLE`$_{242}$
static int	`CONTINUOUS_RUNNING`$_{242}$
static int	`INCREMENTAL_IDLE`$_{243}$
static int	`INCREMENTAL_UPDATING`$_{243}$
static int	`INDEFINITE`$_{243}$

Constructors

	`Gauge(String label, boolean interactive, int maxValue, int initialValue)`$_{244}$

Methods

int	`getMaxValue()`$_{244}$

Member Summary	
int	getValue()[245]
boolean	isInteractive()[245]
void	setMaxValue(int maxValue)[245]
void	setValue(int value)[246]

Inherited Member Summary

Fields inherited from class Item[287]

BUTTON[291], HYPERLINK[292], LAYOUT_2[292], LAYOUT_BOTTOM[292], LAYOUT_CENTER[292], LAYOUT_DEFAULT[292], LAYOUT_EXPAND[293], LAYOUT_LEFT[293], LAYOUT_NEWLINE_AFTER[293], LAYOUT_NEWLINE_BEFORE[293], LAYOUT_RIGHT[293], LAYOUT_SHRINK[294], LAYOUT_TOP[294], LAYOUT_VCENTER[294], LAYOUT_VEXPAND[294], LAYOUT_VSHRINK[294], PLAIN[295]

Methods inherited from class Item[287]

addCommand(Command)[295], getLabel()[295], getLayout()[295], getMinimumHeight()[296], getMinimumWidth()[296], getPreferredHeight()[296], getPreferredWidth()[296], notifyStateChanged()[297], removeCommand(Command)[297], setDefaultCommand(Command)[298], setItemCommandListener(ItemCommandListener)[298], setLabel(String)[298], setLayout(int)[299], setPreferredSize(int, int)[299]

Methods inherited from class Object

equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Fields

CONTINUOUS_IDLE

Declaration:
```
public static final int CONTINUOUS_IDLE
```

Description:
The value representing the continuous-idle state of a non-interactive `Gauge` with indefinite range. In the continuous-idle state, the gauge shows a graphic indicating that no work is in progress.

This value has special meaning only for non-interactive gauges with indefinite range. It is treated as an ordinary value for interactive gauges and for non-interactive gauges with definite range.

The value of `CONTINUOUS_IDLE` is `0`.

Since: MIDP 2.0

CONTINUOUS_RUNNING

Declaration:
```
public static final int CONTINUOUS_RUNNING
```

Description:
The value representing the continuous-running state of a non-interactive `Gauge` with indefinite range. In the continuous-running state, the gauge shows a continually-updating animation sequence that indicates that work is in progress. Once the application sets a gauge into the continuous-running state, the animation should proceed without further requests from the application.

This value has special meaning only for non-interactive gauges with indefinite range. It is treated as an ordinary value for interactive gauges and for non-interactive gauges with definite range.

The value of `CONTINUOUS_RUNNING` is `2`.

Since: MIDP 2.0

INCREMENTAL_IDLE

Declaration:
```
public static final int INCREMENTAL_IDLE
```

Description:
The value representing the incremental-idle state of a non-interactive `Gauge` with indefinite range. In the incremental-idle state, the gauge shows a graphic indicating that no work is in progress.

This value has special meaning only for non-interactive gauges with indefinite range. It is treated as an ordinary value for interactive gauges and for non-interactive gauges with definite range.

The value of `INCREMENTAL_IDLE` is `1`.

Since: MIDP 2.0

INCREMENTAL_UPDATING

Declaration:
```
public static final int INCREMENTAL_UPDATING
```

Description:
The value representing the incremental-updating state of a non-interactive `Gauge` with indefinite range. In the incremental-updating state, the gauge shows a graphic indicating that work is in progress, typically one frame of an animation sequence. The graphic should be updated to the next frame in the sequence only when the application calls `setValue(INCREMENTAL_UPDATING)`.

This value has special meaning only for non-interactive gauges with indefinite range. It is treated as an ordinary value for interactive gauges and for non-interactive gauges with definite range.

The value of `INCREMENTAL_UPDATING` is `3`.

Since: MIDP 2.0

INDEFINITE

Declaration:
```
public static final int INDEFINITE
```

Description:
A special value used for the maximum value in order to indicate that the `Gauge` has indefinite range. This value may be used as the `maxValue` parameter to the constructor, the parameter passed to `setMaxValue()`, and as the return value of `getMaxValue()`.

The value of `INDEFINITE` is `-1`.

Since: MIDP 2.0

Constructors

Gauge(String, boolean, int, int)

Declaration:
```
public Gauge(String label, boolean interactive, int maxValue, int initialValue)
```

Description:
Creates a new `Gauge` object with the given label, in interactive or non-interactive mode, with the given maximum and initial values. In interactive mode (where `interactive` is `true`) the maximum value must be greater than zero, otherwise an exception is thrown. In non-interactive mode (where `interactive` is `false`) the maximum value must be greater than zero or equal to the special value `INDEFINITE`, otherwise an exception is thrown.

If the maximum value is greater than zero, the gauge has definite range. In this case the initial value must be within the range zero to `maxValue`, inclusive. If the initial value is less than zero, the value is set to zero. If the initial value is greater than `maxValue`, it is set to `maxValue`.

If `interactive` is `false` and the maximum value is `INDEFINITE`, this creates a non-interactive gauge with indefinite range. The initial value must be one of `CONTINUOUS_IDLE`, `INCREMENTAL_IDLE`, `CONTINUOUS_RUNNING`, or `INCREMENTAL_UPDATING`.

Parameters:

`label` - the `Gauge's` label

`interactive` - tells whether the user can change the value

`maxValue` - the maximum value, or `INDEFINITE`

`initialValue` - the initial value in the range `[0..maxValue]`, or one of `CONTINUOUS_IDLE`, `INCREMENTAL_IDLE`, `CONTINUOUS_RUNNING`, or `INCREMENTAL_UPDATING` if `maxValue` is `INDEFINITE`.

Throws:

`IllegalArgumentException` - if `maxValue` is not positive for interactive gauges

`IllegalArgumentException` - if `maxValue` is neither positive nor `INDEFINITE` for non-interactive gauges

`IllegalArgumentException` - if initialValue is not one of `CONTINUOUS_IDLE`, `INCREMENTAL_IDLE`, `CONTINUOUS_RUNNING`, or `INCREMENTAL_UPDATING` for a non-interactive gauge with indefinite range

See Also: `INDEFINITE`[243], `CONTINUOUS_IDLE`[242], `INCREMENTAL_IDLE`[243], `CONTINUOUS_RUNNING`[242], `INCREMENTAL_UPDATING`[243]

Methods

getMaxValue()

Declaration:
```
public int getMaxValue()
```

Description:
Gets the maximum value of this `Gauge` object.

If this gauge is interactive, the maximum value will be a positive integer. If this gauge is non-interactive, the maximum value will be a positive integer (indicating that the gauge has definite range) or the special value `INDEFINITE` (indicating that the gauge has indefinite range).

Returns: the maximum value of the `Gauge`, or `INDEFINITE`

See Also: `INDEFINITE`[243], `setMaxValue(int)`[245]

getValue()

Declaration:
```
public int getValue()
```

Description:
Gets the current value of this `Gauge` object.

If this `Gauge` object is a non-interactive gauge with indefinite range, the value returned will be one of `CONTINUOUS_IDLE`, `INCREMENTAL_IDLE`, `CONTINUOUS_RUNNING`, or `INCREMENTAL_UPDATING`. Otherwise, it will be an integer between zero and the gauge's maximum value, inclusive.

Returns: current value of the `Gauge`

See Also: `CONTINUOUS_IDLE`[242], `INCREMENTAL_IDLE`[243], `CONTINUOUS_RUNNING`[242], `INCREMENTAL_UPDATING`[243], `setValue(int)`[246]

isInteractive()

Declaration:
```
public boolean isInteractive()
```

Description:
Tells whether the user is allowed to change the value of the `Gauge`.

Returns: a boolean indicating whether the `Gauge` is interactive

setMaxValue(int)

Declaration:
```
public void setMaxValue(int maxValue)
```

Description:
Sets the maximum value of this `Gauge` object.

For interactive gauges, the new maximum value must be greater than zero, otherwise an exception is thrown. For non-interactive gauges, the new maximum value must be greater than zero or equal to the special value `INDEFINITE`, otherwise an exception is thrown.

If the new maximum value is greater than zero, this provides the gauge with a definite range. If the gauge previously had a definite range, and if the current value is greater than new maximum value, the current value is set to be equal to the new maximum value. If the gauge previously had a definite range, and if the current value is less than or equal to the new maximum value, the current value is left unchanged.

If the new maximum value is greater than zero, and if the gauge had previously had indefinite range, this new maximum value provides it with a definite range. Its graphical representation must change accordingly, the previous state of `CONTINUOUS_IDLE`, `INCREMENTAL_IDLE`, `CONTINUOUS_RUNNING`, or `INCREMENTAL_UPDATING` is ignored, and the current value is set to zero.

If this gauge is non-interactive and the new maximum value is `INDEFINITE`, this gives the gauge indefinite range. If the gauge previously had a definite range, its graphical representation must change accordingly, the previous value is ignored, and the current state is set to `CONTINUOUS_IDLE`. If the gauge previously had an indefinite range, setting the maximum value to `INDEFINITE` will have no effect.

Parameters:
`maxValue` - the new maximum value

Throws:
`IllegalArgumentException` - if `maxValue` is invalid

See Also: `INDEFINITE`[243], `getMaxValue()`[244]

setValue(int)

Declaration:
```
public void setValue(int value)
```

Description:
Sets the current value of this `Gauge` object.

If the gauge is interactive, or if it is non-interactive with definite range, the following rules apply. If the value is less than zero, zero is used. If the current value is greater than the maximum value, the current value is set to be equal to the maximum value.

If this `Gauge` object is a non-interactive gauge with indefinite range, then value must be one of `CONTINUOUS_IDLE`, `INCREMENTAL_IDLE`, `CONTINUOUS_RUNNING`, or `INCREMENTAL_UPDATING`. Other values will cause an exception to be thrown.

Parameters:
`value` - the new value

Throws:
`IllegalArgumentException` - if value is not one of `CONTINUOUS_IDLE`, `INCREMENTAL_IDLE`, `CONTINUOUS_RUNNING`, or `INCREMENTAL_UPDATING` for non-interactive gauges with indefinite range

See Also: `CONTINUOUS_IDLE`[242], `INCREMENTAL_IDLE`[243], `CONTINUOUS_RUNNING`[242], `INCREMENTAL_UPDATING`[243], `getValue()`[245]

javax.microedition.lcdui
Graphics

Declaration

```
public class Graphics

Object
  |
  +--javax.microedition.lcdui.Graphics
```

Description

Provides simple 2D geometric rendering capability.

Drawing primitives are provided for text, images, lines, rectangles, and arcs. Rectangles and arcs may also be filled with a solid color. Rectangles may also be specified with rounded corners.

A 24-bit color model is provided, with 8 bits for each of red, green, and blue components of a color. Not all devices support a full 24 bits' worth of color and thus they will map colors requested by the application into colors available on the device. Facilities are provided in the `Display` 205 class for obtaining device characteristics, such as whether color is available and how many distinct gray levels are available. Applications may also use `getDisplayColor()` 265 to obtain the actual color that would be displayed for a requested color. This enables applications to adapt their behavior to a device without compromising device independence.

For all rendering operations, source pixels are always combined with destination pixels using the *Source Over Destination* rule [Porter-Duff]. Other schemes for combining source pixels with destination pixels, such as raster-ops, are not provided.

For the text, line, rectangle, and arc drawing and filling primitives, the source pixel is a pixel representing the current color of the graphics object being used for rendering. This pixel is always considered to be fully opaque. With source pixel that is always fully opaque, the Source Over Destination rule has the effect of pixel replacement, where destination pixels are simply replaced with the source pixel from the graphics object.

The `drawImage()` 257 and `drawRegion()` 258 methods use an image as the source for rendering operations instead of the current color of the graphics object. In this context, the Source Over Destination rule has the following properties: a fully opaque pixel in the source must replace the destination pixel, a fully transparent pixel in the source must leave the destination pixel unchanged, and a semitransparent pixel in the source must be alpha blended with the destination pixel. Alpha blending of semitransparent pixels is required. If an implementation does not support alpha blending, it must remove all semitransparency from image source data at the time the image is created. See Alpha Processing for further discussion.

The destinations of all graphics rendering are considered to consist entirely of fully opaque pixels. A property of the Source Over Destination rule is that compositing any pixel with a fully opaque destination pixel always results in a fully opaque destination pixel. This has the effect of confining full and partial transparency to immutable images, which may only be used as the source for rendering operations.

Graphics may be rendered directly to the display or to an off-screen image buffer. The destination of rendered graphics depends on the provenance of the graphics object. A graphics object for rendering to the display is passed to the `Canvas` object's `paint()` 151 method. This is the only means by which a graphics object may be obtained whose destination is the display. Furthermore, applications may draw using this graphics object only for the duration of the `paint()` method.

A graphics object for rendering to an off-screen image buffer may be obtained by calling the `getGraphics()` 277 method on the desired image. A graphics object so obtained may be held indefinitely by the application, and requests may be issued on this graphics object at any time.

The default coordinate system's origin is at the upper left-hand corner of the destination. The X-axis direction is positive towards the right, and the Y-axis direction is positive downwards. Applications may assume that horizontal and vertical distances in the coordinate system represent equal distances on the actual device display, that is, pixels are square. A facility is provided for translating the origin of the coordinate system. All coordinates are specified as integers.

The coordinate system represents locations between pixels, not the pixels themselves. Therefore, the first pixel in the upper left corner of the display lies in the square bounded by coordinates `(0,0) , (1,0) , (0,1) , (1,1)`.

Under this definition, the semantics for fill operations are clear. Since coordinate grid lines lie between pixels, fill operations affect pixels that lie entirely within the region bounded by the coordinates of the operation. For example, the operation

```
g.fillRect(0, 0, 3, 2)
```

paints exactly six pixels. (In this example, and in all subsequent examples, the variable `g` is assumed to contain a reference to a `Graphics` object.)

Each character of a font contains a set of pixels that forms the shape of the character. When a character is painted, the pixels forming the character's shape are filled with the `Graphics` object's current color, and the pixels not part of the character's shape are left untouched. The text drawing calls `drawChar()`$_{256}$, `drawChars()`$_{257}$, `drawString()`$_{261}$, and `drawSubstring()`$_{262}$ all draw text in this manner.

Lines, arcs, rectangles, and rounded rectangles may be drawn with either a `SOLID` or a `DOTTED` stroke style, as set by the `setStrokeStyle()`$_{268}$ method. The stroke style does not affect fill, text, and image operations.

For the `SOLID` stroke style, drawing operations are performed with a one-pixel wide pen that fills the pixel immediately below and to the right of the specified coordinate. Drawn lines touch pixels at both endpoints. Thus, the operation

```
g.drawLine(0, 0, 0, 0);
```

paints exactly one pixel, the first pixel in the upper left corner of the display.

Drawing operations under the `DOTTED` stroke style will touch a subset of pixels that would have been touched under the `SOLID` stroke style. The frequency and length of dots is implementation-dependent. The endpoints of lines and arcs are not guaranteed to be drawn, nor are the corner points of rectangles guaranteed to be drawn. Dots are drawn by painting with the current color; spaces between dots are left untouched.

An artifact of the coordinate system is that the area affected by a fill operation differs slightly from the area affected by a draw operation given the same coordinates. For example, consider the operations

```
g.fillRect(x, y, w, h); // 1
g.drawRect(x, y, w, h); // 2
```

Statement (1) fills a rectangle `w` pixels wide and `h` pixels high. Statement (2) draws a rectangle whose left and top edges are within the area filled by statement (1). However, the bottom and right edges lie one pixel outside the filled area. This is counterintuitive, but it preserves the invariant that

```
g.drawLine(x, y, x+w, y);
g.drawLine(x+w, y, x+w, y+h);
g.drawLine(x+w, y+h, x, y+h);
g.drawLine(x, y+h, x, y);
```

has an effect identical to statement (2) above.

The exact pixels painted by `drawLine()` and `drawArc()` are not specified. Pixels touched by a fill operation must either exactly overlap or directly abut pixels touched by the corresponding draw operation. A fill operation must never leave a gap between the filled area and the pixels touched by the corresponding draw operation, nor may the fill operation touch pixels outside the area bounded by the corresponding draw operation.

Clipping

The clip is the set of pixels in the destination of the `Graphics` object that may be modified by graphics rendering operations.

There is a single clip per `Graphics` object. The only pixels modified by graphics operations are those that lie within the clip. Pixels outside the clip are not modified by any graphics operations.

Operations are provided for intersecting the current clip with a given rectangle and for setting the current clip outright. The application may specify the clip by supplying a clip rectangle using coordinates relative to the current coordinate system.

It is legal to specify a clip rectangle whose width or height is zero or negative. In this case the clip is considered to be empty, that is, no pixels are contained within it. Therefore, if any graphics operations are issued under such a clip, no pixels will be modified.

It is legal to specify a clip rectangle that extends beyond or resides entirely beyond the bounds of the destination. No pixels exist outside the bounds of the destination, and the area of the clip rectangle that is outside the destination is ignored. Only the pixels that lie both within the destination and within the specified clip rectangle are considered to be part of the clip.

Operations on the coordinate system, such as `translate()`$_{269}$, do not modify the clip. The methods `getClipX()`$_{265}$, `getClipY()`$_{265}$, `getClipWidth()`$_{264}$ and `getClipHeight()`$_{264}$ must return a rectangle that, if passed to `setClip` without an intervening change to the `Graphics` object's coordinate system, must result in the identical set of pixels in the clip. The rectangle returned from the `getClip` family of methods may differ from the clip rectangle that was requested in `setClip()`$_{267}$. This can occur if the coordinate system has been changed or if the implementation has chosen to intersect the clip rectangle with the bounds of the destination of the `Graphics` object.

If a graphics operation is affected by the clip, the pixels touched by that operation must be the same ones that would be touched as if the clip did not affect the operation. For example, consider a clip represented by the rectangle `(cx, cy, cw, ch)` and a point `(x1, y1)` that lies outside this rectangle and a point `(x2, y2)` that lies within this rectangle. In the following code fragment,

```
g.setClip(0, 0, canvas.getWidth(),
          canvas.getHeight());
g.drawLine(x1, y1, x2, y2); // 3
g.setClip(cx, cy, cw, ch);
g.drawLine(x1, y1, x2, y2); // 4
```

The pixels touched by statement (4) must be identical to the pixels within `(cx, cy, cw, ch)` touched by statement (3).

Anchor Points

The drawing of text is based on "anchor points". Anchor points are used to minimize the amount of computation required when placing text. For example, in order to center a piece of text, an application needs to call `stringWidth()` or `charWidth()` to get the width and then perform a combination of subtraction and division to compute the proper location. The method to draw text is defined as follows:

```
public void drawString(String text, int x, int y, int anchor);
```

This method draws text in the current color, using the current font with its anchor point at `(x,y)`. The definition of the anchor point must be one of the horizontal constants `(LEFT, HCENTER, RIGHT)` combined with one of the vertical constants `(TOP, BASELINE, BOTTOM)` using the bit-wise `OR` operator. Zero may also be used as the value of an anchor point. Using zero for the anchor point value gives results identical to using `TOP | LEFT`.

Vertical centering of the text is not specified since it is not considered useful, it is hard to specify, and it is burdensome to implement. Thus, the `VCENTER` value is not allowed in the anchor point parameter of text drawing calls.

The actual position of the bounding box of the text relative to the `(x, y)` location is determined by the anchor point. These anchor points occur at named locations along the outer edge of the bounding box. Thus, if `f` is `g`'s current font (as returned by `g.getFont()`, the following calls will all have identical results:

```
g.drawString(str, x, y, TOP|LEFT);
g.drawString(str, x + f.stringWidth(str)/2, y, TOP|HCENTER);
g.drawString(str, x + f.stringWidth(str), y, TOP|RIGHT);
g.drawString(str, x,
   y + f.getBaselinePosition(), BASELINE|LEFT);
g.drawString(str, x + f.stringWidth(str)/2,
   y + f.getBaselinePosition(), BASELINE|HCENTER);
g.drawString(str, x + f.stringWidth(str),
   y + f.getBaselinePosition(), BASELINE|RIGHT);
drawString(str, x,
   y + f.getHeight(), BOTTOM|LEFT);
drawString(str, x + f.stringWidth(str)/2,
   y + f.getHeight(), BOTTOM|HCENTER);
drawString(str, x + f.stringWidth(str),
   y + f.getHeight(), BOTTOM|RIGHT);
```

For text drawing, the inter-character and inter-line spacing (leading) specified by the font designer are included as part of the values returned in the `stringWidth()`[230] and `getHeight()`[228] calls of class `Font`[223]. For example, given the following code:

```
// (5)
g.drawString(string1+string2, x, y, TOP|LEFT);
// (6)
g.drawString(string1, x, y, TOP|LEFT);
g.drawString(string2, x + f.stringWidth(string1), y, TOP|LEFT);
```

Code fragments (5) and (6) behave similarly if not identically. This occurs because `f.stringWidth()` includes the inter-character spacing. The exact spacing of may differ between these calls if the system supports font kerning.

Similarly, reasonable vertical spacing may be achieved simply by adding the font height to the Y-position of subsequent lines. For example:

```
g.drawString(string1, x, y, TOP|LEFT);
g.drawString(string2, x, y + f.fontHeight(), TOP|LEFT);
```

draws `string1` and `string2` on separate lines with an appropriate amount of inter-line spacing.

The `stringWidth()` of the string and the `fontHeight()` of the font in which it is drawn define the size of the bounding box of a piece of text. As described above, this box includes inter-line and inter-character spacing. The implementation is required to put this space below and to right of the pixels actually belonging to the characters drawn. Applications that wish to position graphics closely with respect to text (for example, to paint a rectangle around a string of text) may assume that there is space below and to the right of a string and that there is *no* space above and to the left of the string.

Anchor points are also used for positioning of images. Similar to text drawing, the anchor point for an image specifies the point on the bounding rectangle of the destination that is to positioned at the `(x,y)` location given in the graphics request. Unlike text, vertical centering of images is well-defined, and thus the `VCENTER` value may be used within the anchor point parameter of image drawing requests. Because images have no notion of a baseline, the `BASELINE` value may not be used within the anchor point parameter of image drawing requests.

Reference
Porter-Duff

Porter, T., and T. Duff. “Compositing Digital Images.” *Computer Graphics V18 N3 (SIGGRAPH 1984)*, p. 253-259.

Since: MIDP 1.0

Member Summary

Fields

```
static int  BASELINE 253
static int  BOTTOM 253
static int  DOTTED 253
static int  HCENTER 253
static int  LEFT 253
static int  RIGHT 254
static int  SOLID 254
static int  TOP 254
static int  VCENTER 254
```

Methods

```
      void  clipRect(int x, int y, int width, int height) 254
      void  copyArea(int x_src, int y_src, int width, int height, int
            x_dest, int y_dest, int anchor) 255
```

Member Summary

void	drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)256
void	drawChar(char character, int x, int y, int anchor)256
void	drawChars(char[] data, int offset, int length, int x, int y, int anchor)257
void	drawImage(Image img, int x, int y, int anchor)257
void	drawLine(int x1, int y1, int x2, int y2)258
void	drawRect(int x, int y, int width, int height)258
void	drawRegion(Image src, int x_src, int y_src, int width, int height, int transform, int x_dest, int y_dest, int anchor)258
void	drawRGB(int[] rgbData, int offset, int scanlength, int x, int y, int width, int height, boolean processAlpha)260
void	drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)261
void	drawString(String str, int x, int y, int anchor)261
void	drawSubstring(String str, int offset, int len, int x, int y, int anchor)262
void	fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)262
void	fillRect(int x, int y, int width, int height)263
void	fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)263
void	fillTriangle(int x1, int y1, int x2, int y2, int x3, int y3)264
int	getBlueComponent()264
int	getClipHeight()264
int	getClipWidth()264
int	getClipX()265
int	getClipY()265
int	getColor()265
int	getDisplayColor(int color)265
Font	getFont()266
int	getGrayScale()266
int	getGreenComponent()266
int	getRedComponent()266
int	getStrokeStyle()266
int	getTranslateX()267
int	getTranslateY()267
void	setClip(int x, int y, int width, int height)267
void	setColor(int RGB)267
void	setColor(int red, int green, int blue)267
void	setFont(Font font)268
void	setGrayScale(int value)268
void	setStrokeStyle(int style)268
void	translate(int x, int y)269

Inherited Member Summary

Methods inherited from class `Object`

Inherited Member Summary
`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Fields

BASELINE

Declaration:
`public static final int BASELINE`

Description:
Constant for positioning the anchor point at the baseline of text.

Value `64` is assigned to `BASELINE`.

BOTTOM

Declaration:
`public static final int BOTTOM`

Description:
Constant for positioning the anchor point of text and images below the text or image.

Value `32` is assigned to `BOTTOM`.

DOTTED

Declaration:
`public static final int DOTTED`

Description:
Constant for the `DOTTED` stroke style.

Value `1` is assigned to `DOTTED`.

HCENTER

Declaration:
`public static final int HCENTER`

Description:
Constant for centering text and images horizontally around the anchor point

Value `1` is assigned to `HCENTER`.

LEFT

Declaration:
`public static final int LEFT`

Description:
Constant for positioning the anchor point of text and images to the left of the text or image.

Value `4` is assigned to `LEFT`.

RIGHT

Declaration:
```
public static final int RIGHT
```

Description:
Constant for positioning the anchor point of text and images to the right of the text or image.

Value `8` is assigned to `RIGHT`.

SOLID

Declaration:
```
public static final int SOLID
```

Description:
Constant for the `SOLID` stroke style.

Value `0` is assigned to `SOLID`.

TOP

Declaration:
```
public static final int TOP
```

Description:
Constant for positioning the anchor point of text and images above the text or image.

Value `16` is assigned to `TOP`.

VCENTER

Declaration:
```
public static final int VCENTER
```

Description:
Constant for centering images vertically around the anchor point.

Value `2` is assigned to `VCENTER`.

Methods

clipRect(int, int, int, int)

Declaration:
```
public void clipRect(int x, int y, int width, int height)
```

Description:
Intersects the current clip with the specified rectangle. The resulting clipping area is the intersection of the current clipping area and the specified rectangle. This method can only be used to make the current clip smaller. To set the current clip larger, use the `setClip` method. Rendering operations have no effect outside of the clipping area.

Parameters:

`x` - the x coordinate of the rectangle to intersect the clip with

`y` - the y coordinate of the rectangle to intersect the clip with

`width` - the width of the rectangle to intersect the clip with

`height` - the height of the rectangle to intersect the clip with

See Also: `setClip(int, int, int, int)`[267]

copyArea(int, int, int, int, int, int, int)

Declaration:

```
public void copyArea(int x_src, int y_src, int width, int height, int x_dest, int y_dest,
            int anchor)
```

Description:

Copies the contents of a rectangular area `(x_src, y_src, width, height)` to a destination area, whose anchor point identified by anchor is located at `(x_dest, y_dest)`. The effect must be that the destination area contains an exact copy of the contents of the source area immediately prior to the invocation of this method. This result must occur even if the source and destination areas overlap.

The points `(x_src, y_src)` and `(x_dest, y_dest)` are both specified relative to the coordinate system of the `Graphics` object. It is illegal for the source region to extend beyond the bounds of the graphic object. This requires that:

x_src + tx >= 0
y_src + ty >= 0
x_src + tx + width <= width of Graphics object's destination
y_src + ty + height <= height of Graphics object's destination

where `tx` and `ty` represent the X and Y coordinates of the translated origin of this graphics object, as returned by `getTranslateX()` and `getTranslateY()`, respectively.

However, it is legal for the destination area to extend beyond the bounds of the `Graphics` object. Pixels outside of the bounds of the `Graphics` object will not be drawn.

The `copyArea` method is allowed on all `Graphics` objects except those whose destination is the actual display device. This restriction is necessary because allowing a `copyArea` method on the display would adversely impact certain techniques for implementing double-buffering.

Like other graphics operations, the `copyArea` method uses the Source Over Destination rule for combining pixels. However, since it is defined only for mutable images, which can contain only fully opaque pixels, this is effectively the same as pixel replacement.

Parameters:

`x_src` - the x coordinate of upper left corner of source area

`y_src` - the y coordinate of upper left corner of source area

`width` - the width of the source area

`height` - the height of the source area

`x_dest` - the x coordinate of the destination anchor point

`y_dest` - the y coordinate of the destination anchor point

`anchor` - the anchor point for positioning the region within the destination image

Throws:

`IllegalStateException`[37] - if the destination of this `Graphics` object is the display device

`IllegalArgumentException` - if the region to be copied exceeds the bounds of the source image

Since: MIDP 2.0

drawArc(int, int, int, int, int, int)

Declaration:
```
public void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)
```

Description:
Draws the outline of a circular or elliptical arc covering the specified rectangle, using the current color and stroke style.

The resulting arc begins at `startAngle` and extends for `arcAngle` degrees, using the current color. Angles are interpreted such that `0` degrees is at the `3` o'clock position. A positive value indicates a counter-clockwise rotation while a negative value indicates a clockwise rotation.

The center of the arc is the center of the rectangle whose origin is (*x*, *y*) and whose size is specified by the `width` and `height` arguments.

The resulting arc covers an area `width + 1` pixels wide by `height + 1` pixels tall. If either `width` or `height` is less than zero, nothing is drawn.

The angles are specified relative to the non-square extents of the bounding rectangle such that `45` degrees always falls on the line from the center of the ellipse to the upper right corner of the bounding rectangle. As a result, if the bounding rectangle is noticeably longer in one axis than the other, the angles to the start and end of the arc segment will be skewed farther along the longer axis of the bounds.

Parameters:

`x` - the *x* coordinate of the upper-left corner of the arc to be drawn

`y` - the *y* coordinate of the upper-left corner of the arc to be drawn

`width` - the width of the arc to be drawn

`height` - the height of the arc to be drawn

`startAngle` - the beginning angle

`arcAngle` - the angular extent of the arc, relative to the start angle

See Also: `fillArc(int, int, int, int, int, int)`262

drawChar(char, int, int, int)

Declaration:
```
public void drawChar(char character, int x, int y, int anchor)
```

Description:
Draws the specified character using the current font and color.

Parameters:

`character` - the character to be drawn

`x` - the x coordinate of the anchor point

`y` - the y coordinate of the anchor point

`anchor` - the anchor point for positioning the text; see anchor points

Throws:

`IllegalArgumentException` - if `anchor` is not a legal value

See Also: `drawString(String, int, int, int)`261, `drawChars(char[], int, int, int, int, int)`257

drawChars(char[], int, int, int, int, int)

Declaration:
```
public void drawChars(char[] data, int offset, int length, int x, int y, int anchor)
```

Description:
Draws the specified characters using the current font and color.

The `offset` and `length` parameters must specify a valid range of characters within the character array `data`. The `offset` parameter must be within the range `[0..(data.length)]`, inclusive. The `length` parameter must be a non-negative integer such that `(offset + length) <= data.length`.

Parameters:

`data` - the array of characters to be drawn

`offset` - the start offset in the data

`length` - the number of characters to be drawn

`x` - the x coordinate of the anchor point

`y` - the y coordinate of the anchor point

`anchor` - the anchor point for positioning the text; see anchor points

Throws:

`ArrayIndexOutOfBoundsException` - if `offset` and `length` do not specify a valid range within the data array

`IllegalArgumentException` - if anchor is not a legal value

`NullPointerException` - if `data` is `null`

See Also: `drawString(String, int, int, int)`261

drawImage(Image, int, int, int)

Declaration:
```
public void drawImage(javax.microedition.lcdui.Image270 img, int x, int y, int anchor)
```

Description:
Draws the specified image by using the anchor point. The image can be drawn in different positions relative to the anchor point by passing the appropriate position constants. See anchor points.

If the source image contains transparent pixels, the corresponding pixels in the destination image must be left untouched. If the source image contains partially transparent pixels, a compositing operation must be performed with the destination pixels, leaving all pixels of the destination image fully opaque.

If `img` is the same as the destination of this Graphics object, the result is undefined. For copying areas within an `Image`, `copyArea`255 should be used instead.

Parameters:

`img` - the specified image to be drawn

`x` - the x coordinate of the anchor point

`y` - the y coordinate of the anchor point

`anchor` - the anchor point for positioning the image

Throws:

`IllegalArgumentException` - if `anchor` is not a legal value

`NullPointerException` - if `img` is `null`

See Also: `Image`[270]

drawLine(int, int, int, int)

Declaration:
```
public void drawLine(int x1, int y1, int x2, int y2)
```

Description:
Draws a line between the coordinates `(x1,y1)` and `(x2,y2)` using the current color and stroke style.

Parameters:
`x1` - the x coordinate of the start of the line

`y1` - the y coordinate of the start of the line

`x2` - the x coordinate of the end of the line

`y2` - the y coordinate of the end of the line

drawRect(int, int, int, int)

Declaration:
```
public void drawRect(int x, int y, int width, int height)
```

Description:
Draws the outline of the specified rectangle using the current color and stroke style. The resulting rectangle will cover an area `(width + 1)` pixels wide by `(height + 1)` pixels tall. If either width or height is less than zero, nothing is drawn.

Parameters:
`x` - the x coordinate of the rectangle to be drawn

`y` - the y coordinate of the rectangle to be drawn

`width` - the width of the rectangle to be drawn

`height` - the height of the rectangle to be drawn

See Also: `fillRect(int, int, int, int)`[263]

drawRegion(Image, int, int, int, int, int, int, int, int)

Declaration:
```
public void drawRegion(javax.microedition.lcdui.Image[270] src, int x_src, int y_src,
            int width, int height, int transform, int x_dest, int y_dest, int anchor)
```

Description:
Copies a region of the specified source image to a location within the destination, possibly transforming (rotating and reflecting) the image data using the chosen transform function.

The destination, if it is an image, must not be the same image as the source image. If it is, an exception is thrown. This restriction is present in order to avoid ill-defined behaviors that might occur if overlapped, transformed copies were permitted.

The transform function used must be one of the following, as defined in the `Sprite`[365] class:
`Sprite.TRANS_NONE` - causes the specified image region to be copied unchanged
`Sprite.TRANS_ROT90` - causes the specified image region to be rotated clockwise by 90 degrees.
`Sprite.TRANS_ROT180` - causes the specified image region to be rotated clockwise by 180 degrees.
`Sprite.TRANS_ROT270` - causes the specified image region to be rotated clockwise by 270 degrees.

`Sprite.TRANS_MIRROR` - causes the specified image region to be reflected about its vertical center.
`Sprite.TRANS_MIRROR_ROT90` - causes the specified image region to be reflected about its vertical center and then rotated clockwise by 90 degrees.
`Sprite.TRANS_MIRROR_ROT180` - causes the specified image region to be reflected about its vertical center and then rotated clockwise by 180 degrees.
`Sprite.TRANS_MIRROR_ROT270` - causes the specified image region to be reflected about its vertical center and then rotated clockwise by 270 degrees.

If the source region contains transparent pixels, the corresponding pixels in the destination region must be left untouched. If the source region contains partially transparent pixels, a compositing operation must be performed with the destination pixels, leaving all pixels of the destination region fully opaque.

The `(x_src, y_src)` coordinates are relative to the upper left corner of the source image. The `x_src`, `y_src`, `width`, and `height` parameters specify a rectangular region of the source image. It is illegal for this region to extend beyond the bounds of the source image. This requires that:

```
x_src >= 0
y_src >= 0
x_src + width <= source width
y_src + height <= source height
```

The `(x_dest, y_dest)` coordinates are relative to the coordinate system of this Graphics object. It is legal for the destination area to extend beyond the bounds of the `Graphics` object. Pixels outside of the bounds of the `Graphics` object will not be drawn.

The transform is applied to the image data from the region of the source image, and the result is rendered with its anchor point positioned at location `(x_dest, y_dest)` in the destination.

Parameters:

`src` - the source image to copy from

`x_src` - the x coordinate of the upper left corner of the region within the source image to copy

`y_src` - the y coordinate of the upper left corner of the region within the source image to copy

`width` - the width of the region to copy

`height` - the height of the region to copy

`transform` - the desired transformation for the selected region being copied

`x_dest` - the x coordinate of the anchor point in the destination drawing area

`y_dest` - the y coordinate of the anchor point in the destination drawing area

`anchor` - the anchor point for positioning the region within the destination image

Throws:

`IllegalArgumentException` - if `src` is the same image as the destination of this `Graphics` object

`NullPointerException` - if `src` is `null`

`IllegalArgumentException` - if `transform` is invalid

`IllegalArgumentException` - if `anchor` is invalid

`IllegalArgumentException` - if the region to be copied exceeds the bounds of the source image

Since: MIDP 2.0

drawRGB(int[], int, int, int, int, int, int, boolean)

Declaration:

```
public void drawRGB(int[] rgbData, int offset, int scanlength, int x, int y, int width,
            int height, boolean processAlpha)
```

Description:

Renders a series of device-independent RGB+transparency values in a specified region. The values are stored in `rgbData` in a format with `24` bits of RGB and an eight-bit alpha value (`0xAARRGGBB`), with the first value stored at the specified offset. The `scanlength` specifies the relative offset within the array between the corresponding pixels of consecutive rows. Any value for `scanlength` is acceptable (even negative values) provided that all resulting references are within the bounds of the `rgbData` array. The ARGB data is rasterized horizontally from left to right within each row. The ARGB values are rendered in the region specified by `x`, `y`, `width` and `height`, and the operation is subject to the current clip region and translation for this `Graphics` object.

Consider `P(a,b)` to be the value of the pixel located at column `a` and row `b` of the Image, where rows and columns are numbered downward from the top starting at zero, and columns are numbered rightward from the left starting at zero. This operation can then be defined as:

```
P(a, b) = rgbData[offset + (a - x) + (b - y) * scanlength]
```

for

```
x <= a < x + width
y <= b < y + height
```

This capability is provided in the `Graphics` class so that it can be used to render both to the screen and to offscreen `Image` objects. The ability to retrieve ARGB values is provided by the `Image.getRGB(int[], int, int, int, int, int, int)`[278] method.

If `processAlpha` is `true`, the high-order byte of the ARGB format specifies opacity; that is, `0x00RRGGBB` specifies a fully transparent pixel and `0xFFRRGGBB` specifies a fully opaque pixel. Intermediate alpha values specify semitransparency. If the implementation does not support alpha blending for image rendering operations, it must remove any semitransparency from the source data prior to performing any rendering. (See Alpha Processing for further discussion.) If `processAlpha` is `false`, the alpha values are ignored and all pixels must be treated as completely opaque.

The mapping from ARGB values to the device-dependent pixels is platform-specific and may require significant computation.

Parameters:

`rgbData` - an array of ARGB values in the format `0xAARRGGBB`

`offset` - the array index of the first ARGB value

`scanlength` - the relative array offset between the corresponding pixels in consecutive rows in the `rgbData` array

`x` - the horizontal location of the region to be rendered

`y` - the vertical location of the region to be rendered

`width` - the width of the region to be rendered

`height` - the height of the region to be rendered

`processAlpha` - `true` if `rgbData` has an alpha channel, false if all pixels are fully opaque

Throws:

`ArrayIndexOutOfBoundsException` - if the requested operation will attempt to access an element of `rgbData` whose index is either negative or beyond its length

`NullPointerException` - if `rgbData` is `null`

Since: MIDP 2.0

drawRoundRect(int, int, int, int, int, int)

Declaration:

```
public void drawRoundRect(int x, int y, int width, int height, int arcWidth,
            int arcHeight)
```

Description:

Draws the outline of the specified rounded corner rectangle using the current color and stroke style. The resulting rectangle will cover an area `(width + 1)` pixels wide by `(height + 1)` pixels tall. If either `width` or `height` is less than zero, nothing is drawn.

Parameters:

`x` - the x coordinate of the rectangle to be drawn

`y` - the y coordinate of the rectangle to be drawn

`width` - the width of the rectangle to be drawn

`height` - the height of the rectangle to be drawn

`arcWidth` - the horizontal diameter of the arc at the four corners

`arcHeight` - the vertical diameter of the arc at the four corners

See Also: `fillRoundRect(int, int, int, int, int, int)` *263*

drawString(String, int, int, int)

Declaration:

```
public void drawString(String str, int x, int y, int anchor)
```

Description:

Draws the specified `String` using the current font and color. The `x,y` position is the position of the anchor point. See anchor points.

Parameters:

`str` - the `String` to be drawn

`x` - the x coordinate of the anchor point

`y` - the y coordinate of the anchor point

`anchor` - the anchor point for positioning the text

Throws:

`NullPointerException` - if `str` is `null`

`IllegalArgumentException` - if anchor is not a legal value

See Also: `drawChars(char[], int, int, int, int, int)`*257*

drawSubstring(String, int, int, int, int, int)

Declaration:

```
public void drawSubstring(String str, int offset, int len, int x, int y, int anchor)
```

Description:

Draws the specified `String` using the current font and color. The `x,y` position is the position of the anchor point. See anchor points.

The `offset` and `len` parameters must specify a valid range of characters within the string `str`. The `offset` parameter must be within the range `[0..(str.length())]`, inclusive. The `len` parameter must be a non-negative integer such that `(offset + len) <= str.length()`.

Parameters:

`str` - the `String` to be drawn

`offset` - zero-based index of first character in the substring

`len` - length of the substring

`x` - the x coordinate of the anchor point

`y` - the y coordinate of the anchor point

`anchor` - the anchor point for positioning the text

Throws:

`StringIndexOutOfBoundsException` - if `offset` and `length` do not specify a valid range within the `String` `str`

`IllegalArgumentException` - if `anchor` is not a legal value

`NullPointerException` - if `str` is `null`

See Also: `drawString(String, int, int, int)`*261*

fillArc(int, int, int, int, int, int)

Declaration:

```
public void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)
```

Description:

Fills a circular or elliptical arc covering the specified rectangle.

The resulting arc begins at `startAngle` and extends for `arcAngle` degrees. Angles are interpreted such that `0` degrees is at the `3` o'clock position. A positive value indicates a counter-clockwise rotation while a negative value indicates a clockwise rotation.

The center of the arc is the center of the rectangle whose origin is (*x*, *y*) and whose size is specified by the `width` and `height` arguments.

If either `width` or `height` is zero or less, nothing is drawn.

The filled region consists of the "pie wedge" region bounded by the arc segment as if drawn by `drawArc()`, the radius extending from the center to this arc at `startAngle` degrees, and radius extending from the center to this arc at `startAngle + arcAngle` degrees.

The angles are specified relative to the non-square extents of the bounding rectangle such that `45` degrees always falls on the line from the center of the ellipse to the upper right corner of the bounding rectangle. As a result, if the bounding rectangle is noticeably longer in one axis than the other, the angles to the start and end of the arc segment will be skewed farther along the longer axis of the bounds.

Parameters:

`x` - the *x* coordinate of the upper-left corner of the arc to be filled.

`y` - the *y* coordinate of the upper-left corner of the arc to be filled.

`width` - the width of the arc to be filled

`height` - the height of the arc to be filled

`startAngle` - the beginning angle.

`arcAngle` - the angular extent of the arc, relative to the start angle.

See Also: `drawArc(int, int, int, int, int, int)` 256

fillRect(int, int, int, int)

Declaration:

```
public void fillRect(int x, int y, int width, int height)
```

Description:

Fills the specified rectangle with the current color. If either width or height is zero or less, nothing is drawn.

Parameters:

`x` - the x coordinate of the rectangle to be filled

`y` - the y coordinate of the rectangle to be filled

`width` - the width of the rectangle to be filled

`height` - the height of the rectangle to be filled

See Also: `drawRect(int, int, int, int)` 258

fillRoundRect(int, int, int, int, int, int)

Declaration:

```
public void fillRoundRect(int x, int y, int width, int height, int arcWidth,
            int arcHeight)
```

Description:

Fills the specified rounded corner rectangle with the current color. If either `width` or `height` is zero or less, nothing is drawn.

Parameters:

`x` - the x coordinate of the rectangle to be filled

`y` - the y coordinate of the rectangle to be filled

`width` - the width of the rectangle to be filled

`height` - the height of the rectangle to be filled

`arcWidth` - the horizontal diameter of the arc at the four corners

`arcHeight` - the vertical diameter of the arc at the four corners

See Also: `drawRoundRect(int, int, int, int, int, int)` 261

fillTriangle(int, int, int, int, int, int)

Declaration:
```
public void fillTriangle(int x1, int y1, int x2, int y2, int x3, int y3)
```

Description:
Fills the specified triangle will the current color. The lines connecting each pair of points are included in the filled triangle.

Parameters:
`x1` - the x coordinate of the first vertex of the triangle

`y1` - the y coordinate of the first vertex of the triangle

`x2` - the x coordinate of the second vertex of the triangle

`y2` - the y coordinate of the second vertex of the triangle

`x3` - the x coordinate of the third vertex of the triangle

`y3` - the y coordinate of the third vertex of the triangle

Since: MIDP 2.0

getBlueComponent()

Declaration:
```
public int getBlueComponent()
```

Description:
Gets the blue component of the current color.

Returns: integer value in range `0-255`

See Also: `setColor(int, int, int)`*267*

getClipHeight()

Declaration:
```
public int getClipHeight()
```

Description:
Gets the height of the current clipping area.

Returns: height of the current clipping area.

See Also: `clipRect(int, int, int, int)`*254*, `setClip(int, int, int, int)`*267*

getClipWidth()

Declaration:
```
public int getClipWidth()
```

Description:
Gets the width of the current clipping area.

Returns: width of the current clipping area.

See Also: `clipRect(int, int, int, int)`*254*, `setClip(int, int, int, int)`*267*

getClipX()

Declaration:
```
public int getClipX()
```

Description:
Gets the X offset of the current clipping area, relative to the coordinate system origin of this graphics context. Separating the `getClip` operation into two methods returning integers is more performance and memory efficient than one `getClip()` call returning an object.

Returns: X offset of the current clipping area

See Also: `clipRect(int, int, int, int)`[254], `setClip(int, int, int, int)`[267]

getClipY()

Declaration:
```
public int getClipY()
```

Description:
Gets the Y offset of the current clipping area, relative to the coordinate system origin of this graphics context. Separating the `getClip` operation into two methods returning integers is more performance and memory efficient than one `getClip()` call returning an object.

Returns: Y offset of the current clipping area

See Also: `clipRect(int, int, int, int)`[254], `setClip(int, int, int, int)`[267]

getColor()

Declaration:
```
public int getColor()
```

Description:
Gets the current color.

Returns: an integer in form `0x00RRGGBB`

See Also: `setColor(int, int, int)`[267]

getDisplayColor(int)

Declaration:
```
public int getDisplayColor(int color)
```

Description:
Gets the color that will be displayed if the specified color is requested. This method enables the developer to check the manner in which RGB values are mapped to the set of distinct colors that the device can actually display. For example, with a monochrome device, this method will return either `0xFFFFFF` (white) or `0x000000` (black) depending on the brightness of the specified color.

Parameters:
`color` - the desired color (in `0x00RRGGBB` format, the high-order byte is ignored)

Returns: the corresponding color that will be displayed on the device's screen (in `0x00RRGGBB` format)

Since: MIDP 2.0

getFont()

Declaration:
`public javax.microedition.lcdui.Font`[223] **`getFont`**`()`

Description:
Gets the current font.

Returns: current font

See Also: `Font`[223], `setFont(Font)`[268]

getGrayScale()

Declaration:
`public int` **`getGrayScale`**`()`

Description:
Gets the current grayscale value of the color being used for rendering operations. If the color was set by `setGrayScale()`, that value is simply returned. If the color was set by one of the methods that allows setting of the red, green, and blue components, the value returned is computed from the RGB color components (possibly in a device-specific fashion) that best approximates the brightness of that color.

Returns: integer value in range `0-255`

See Also: `setGrayScale(int)`[268]

getGreenComponent()

Declaration:
`public int` **`getGreenComponent`**`()`

Description:
Gets the green component of the current color.

Returns: integer value in range `0-255`

See Also: `setColor(int, int, int)`[267]

getRedComponent()

Declaration:
`public int` **`getRedComponent`**`()`

Description:
Gets the red component of the current color.

Returns: integer value in range `0-255`

See Also: `setColor(int, int, int)`[267]

getStrokeStyle()

Declaration:
`public int` **`getStrokeStyle`**`()`

Description:
Gets the stroke style used for drawing operations.

Returns: stroke style, `SOLID` or `DOTTED`

See Also: `setStrokeStyle(int)`[268]

getTranslateX()

Declaration:
```
public int getTranslateX()
```

Description:
Gets the X coordinate of the translated origin of this graphics context.

Returns: X of current origin

getTranslateY()

Declaration:
```
public int getTranslateY()
```

Description:
Gets the Y coordinate of the translated origin of this graphics context.

Returns: Y of current origin

setClip(int, int, int, int)

Declaration:
```
public void setClip(int x, int y, int width, int height)
```

Description:
Sets the current clip to the rectangle specified by the given coordinates. Rendering operations have no effect outside of the clipping area.

Parameters:
`x` - the x coordinate of the new clip rectangle

`y` - the y coordinate of the new clip rectangle

`width` - the width of the new clip rectangle

`height` - the height of the new clip rectangle

See Also: `clipRect(int, int, int, int)` *254*

setColor(int)

Declaration:
```
public void setColor(int RGB)
```

Description:
Sets the current color to the specified RGB values. All subsequent rendering operations will use this specified color. The RGB value passed in is interpreted with the least significant eight bits giving the blue component, the next eight more significant bits giving the green component, and the next eight more significant bits giving the red component. That is to say, the color component is specified in the form of `0x00RRGGBB`. The high order byte of this value is ignored.

Parameters:
`RGB` - the color being set

See Also: `getColor()` *265*

setColor(int, int, int)

Declaration:
```
public void setColor(int red, int green, int blue)
```

Description:
Sets the current color to the specified RGB values. All subsequent rendering operations will use this specified color.

Parameters:
`red` - the red component of the color being set in range `0-255`

`green` - the green component of the color being set in range `0-255`

`blue` - the blue component of the color being set in range `0-255`

Throws:
`IllegalArgumentException` - if any of the color components are outside of range `0-255`

See Also: `getColor()`[265]

setFont(Font)

Declaration:
```
public void setFont(javax.microedition.lcdui.Font[223] font)
```

Description:
Sets the font for all subsequent text rendering operations. If font is `null`, it is equivalent to `setFont(Font.getDefaultFont())`.

Parameters:
`font` - the specified font

See Also: `Font`[223], `getFont()`[266], `drawString(String, int, int, int)`[261], `drawChars(char[], int, int, int, int, int)`[257]

setGrayScale(int)

Declaration:
```
public void setGrayScale(int value)
```

Description:
Sets the current grayscale to be used for all subsequent rendering operations. For monochrome displays, the behavior is clear. For color displays, this sets the color for all subsequent drawing operations to be a gray color equivalent to the value passed in. The value must be in the range `0-255`.

Parameters:
`value` - the desired grayscale value

Throws:
`IllegalArgumentException` - if the gray value is out of range

See Also: `getGrayScale()`[266]

setStrokeStyle(int)

Declaration:
```
public void setStrokeStyle(int style)
```

Description:
Sets the stroke style used for drawing lines, arcs, rectangles, and rounded rectangles. This does not affect fill, text, and image operations.

Parameters:
`style` - can be `SOLID` or `DOTTED`

Throws:

`IllegalArgumentException` - if the `style` is illegal

See Also: `getStrokeStyle()`[266]

translate(int, int)

Declaration:

```
public void translate(int x, int y)
```

Description:

Translates the origin of the graphics context to the point `(x, y)` in the current coordinate system. All coordinates used in subsequent rendering operations on this graphics context will be relative to this new origin.

The effect of calls to `translate()` are cumulative. For example, calling `translate(1, 2)` and then `translate(3, 4)` results in a translation of `(4, 6)`.

The application can set an absolute origin `(ax, ay)` using the following technique:

```
g.translate(ax - g.getTranslateX(), ay - g.getTranslateY())
```

Parameters:

`x` - the x coordinate of the new translation origin

`y` - the y coordinate of the new translation origin

See Also: `getTranslateX()`[267], `getTranslateY()`[267]

javax.microedition.lcdui
Image

Declaration

```
public class Image

Object
  |
  +--javax.microedition.lcdui.Image
```

Description

The `Image` class is used to hold graphical image data. `Image` objects exist independently of the display device. They exist only in off-screen memory and will not be painted on the display unless an explicit command is issued by the application (such as within the `paint()` method of a `Canvas`) or when an `Image` object is placed within a `Form` screen or an `Alert` screen and that screen is made current.

Images are either *mutable* or *immutable* depending upon how they are created. Immutable images are generally created by loading image data from resource bundles, from files, or from the network. They may not be modified once created. Mutable images are created as blank images containing only white pixels. The application may render on a mutable image by calling `getGraphics()`$_{277}$ on the `Image` to obtain a `Graphics` object expressly for this purpose.

`Images` may be placed within `Alert`, `Choice`, `Form`, or `ImageItem` objects. The high-level user interface implementation may need to update the display at any time, without notifying the application. In order to provide predictable behavior, the high-level user interface objects provide snapshot semantics for the image. That is, when a mutable image is placed within an `Alert`, `Choice`, `Form`, or `ImageItem` object, the effect is as if a snapshot is taken of its current contents. This snapshot is then used for all subsequent painting of the high-level user interface component. If the application modifies the contents of the image, the application must update the component containing the image (for example, by calling `ImageItem.setImage`) in order to make the modified contents visible.

An immutable image may be created from a mutable image through the use of the `createImage`$_{274}$ method. It is possible to create a mutable copy of an immutable image using a technique similar to the following:

```
Image source; // the image to be copied
source = Image.createImage(...);
Image copy = Image
   .createImage(source.getWidth(), source.getHeight());
Graphics g = copy.getGraphics();
g.drawImage(source, 0, 0, TOP|LEFT);
```

Alpha Processing

Every pixel within a mutable image is always fully opaque. Immutable images may contain a combination of fully opaque pixels (`alpha = 2bitdepth - 1`), fully transparent pixels (`alpha = 0`), and semitransparent pixels (`0 < alpha < 2bitdepth - 1`), where *bitdepth* is the number of bits per sample in the source data.

Implementations must support storage, processing, and rendering of fully opaque pixels and fully transparent pixels in immutable images. When creating an image from source data (whether from a PNG file or from an

array of ARGB data), a fully opaque pixel in the source data must always result in a fully opaque pixel in the new image, and a fully transparent pixel in the source data must always result in a fully transparent pixel in the new image.

The required treatment of semitransparent pixel data depends upon whether the implementation supports alpha blending at rendering time. If the implementation supports alpha blending, a semitransparent pixel in the source data must result in a semitransparent pixel in the new image. The resulting alpha value may be modified to accommodate the number of levels of semitransparency supported by the platform. (See the `Display.numAlphaLevels()`$_{213}$ method.) If an implementation does not support alpha blending, any semitransparent pixels in the source data must be replaced with fully transparent pixels in the new image.

PNG Image Format

Implementations are required to support images stored in the PNG format, as specified by the *PNG (Portable Network Graphics) Specification, Version 1.0.* All conforming MIDP implementations are also conformant to the minimum set of requirements given by the *PNG Specification*. MIDP implementations also must conform to additional requirements given here with respect to handling of PNG images. Note that the requirements listed here take precedence over any conflicting recommendations given in the *PNG Specification*.

Critical Chunks

All of the 'critical' chunks specified by PNG must be supported. The paragraphs below describe these critical chunks.

The IHDR chunk. MIDP devices must handle the following values in the IHDR chunk:

- All positive values of width and height are supported; however, a very large image may not be readable because of memory constraints. The dimensions of the resulting `Image` object must match the dimensions of the PNG image. That is, the values returned by `getWidth()`$_{279}$ and `getHeight()`$_{278}$ and the rendered width and height must equal the width and height specified in the IHDR chunk.
- All color types are supported, although the appearance of the image will be dependent on the capabilities of the device's screen. Color types that include alpha channel data are supported.
- For color types `4` & `6` (grayscale with alpha and RGB with alpha, respectively) the alpha channel must be decoded. Any pixels with an alpha value of zero must be treated as transparent. Any pixels with an alpha value of `255` (for images with `8` bits per sample) or `65535` (for images with `16` bits per sample) must be treated as opaque. If rendering with alpha blending is supported, any pixels with intermediate alpha values must be carried through to the resulting image. If alpha blending is not supported, any pixels with intermediate alpha values must be replaced with fully transparent pixels.
- All bit depth values for the given color type are supported.
- Compression method `0` (deflate) is the only supported compression method. This method utilizes the "zlib" compression scheme, which is also used for jar files; thus, the decompression (inflate) code may be shared between the jar decoding and PNG decoding implementations. As noted in the PNG specification, the compressed data stream may comprised internally of both compressed and uncompressed (raw) data.
- The filter method represents a series of encoding schemes that may be used to optimize compression. The PNG spec currently defines a single filter method (method `0`) that is an adaptive filtering scheme with five basic filter types. Filtering is essential for optimal compression since it allows the deflate algorithm to exploit spatial similarities within the image. Therefore, MIDP devices must support all five filter types defined by filter method `0`.
- MIDP devices are required to read PNG images that are encoded with either interlace method `0` (None) or interlace method `1` (Adam7). Image loading in MIDP is synchronous and cannot be overlapped with image rendering, and so there is no advantage for an application to use interlace method `1`. Support for decoding interlaced images is required for compatibility with PNG and for the convenience of developers who may

already have interlaced images available.

The PLTE chunk. Palette-based images must be supported.

The IDAT chunk. Image data may be encoded using any of the `5` filter types defined by filter method `0` (None, Sub, Up, Average, Paeth).

The IEND chunk. This chunk must be found in order for the image to be considered valid.

Ancillary Chunks

PNG defines several 'ancillary' chunks that may be present in a PNG image but are not critical for image decoding.

The tRNS chunk. All implementations must support the tRNS chunk. This chunk is used to implement transparency without providing alpha channel data for each pixel. For color types `0` and `2`, a particular gray or RGB value is defined to be a transparent pixel. In this case, the implementation must treat pixels with this value as fully transparent. Pixel value comparison must be based on the actual pixel values using the original sample depth; that is, this comparison must be performed before the pixel values are resampled to reflect the display capabilities of the device. For color type `3` (indexed color), `8`-bit alpha values are potentially provided for each entry in the color palette. In this case, the implementation must treat pixels with an alpha value of `0` as fully transparent, and it must treat pixels with an alpha value of `255` as fully opaque. If rendering with alpha blending is supported, any pixels with intermediate alpha values must be carried through to the resulting image. If alpha blending is not supported, any pixels with intermediate alpha values must be replaced with fully transparent pixels.

The implementation *may* (but is not required to) support any of the other ancillary chunks. The implementation *must* silently ignore any unsupported ancillary chunks that it encounters. The currently defined optional ancillary chunks are:

```
cHRM gAMA hIST iCCP iTXt pHYs
sBIT sPLT sRGB tEXt tIME zTXt
```

Reference

PNG (Portable Network Graphics) Specification, Version 1.0. W3C Recommendation, October 1, 1996. http://www.w3.org/TR/REC-png.html. Also available as RFC 2083, http://www.ietf.org/rfc/rfc2083.txt.

Since: MIDP 1.0

Member Summary

Methods

static Image	createImage(byte[] imageData, int imageOffset, int imageLength)[273]
static Image	createImage(Image source)[274]
static Image	createImage(Image image, int x, int y, int width, int height, int transform)[274]
static Image	createImage(java.io.InputStream stream)[275]
static Image	createImage(int width, int height)[276]
static Image	createImage(String name)[276]
static Image	createRGBImage(int[] rgb, int width, int height, boolean processAlpha)[276]
Graphics	getGraphics()[277]
int	getHeight()[278]

Member Summary	
void	getRGB(int[] rgbData, int offset, int scanlength, int x, int y, int width, int height)[278]
int	getWidth()[279]
boolean	isMutable()[280]

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Methods

createImage(byte[], int, int)

Declaration:

```
public static javax.microedition.lcdui.Image[270] createImage(byte[] imageData,
             int imageOffset, int imageLength)
```

Description:

Creates an immutable image which is decoded from the data stored in the specified byte array at the specified offset and length. The data must be in a self-identifying image file format supported by the implementation, such as PNG.

The `imageoffset` and `imagelength` parameters specify a range of data within the `imageData` byte array. The `imageOffset` parameter specifies the offset into the array of the first data byte to be used. It must therefore lie within the range `[0..(imageData.length-1)]`. The `imageLength` parameter specifies the number of data bytes to be used. It must be a positive integer and it must not cause the range to extend beyond the end of the array. That is, it must be true that `imageOffset + imageLength < imageData.length`.

This method is intended for use when loading an image from a variety of sources, such as from persistent storage or from the network.

Parameters:

`imageData` - the array of image data in a supported image format

`imageOffset` - the offset of the start of the data in the array

`imageLength` - the length of the data in the array

Returns: the created image

Throws:

`ArrayIndexOutOfBoundsException` - if `imageOffset` and `imageLength` specify an invalid range

`NullPointerException` - if `imageData` is `null`

`IllegalArgumentException` - if `imageData` is incorrectly formatted or otherwise cannot be decoded

createImage(Image)

Declaration:

```
public static javax.microedition.lcdui.Image270
            createImage(javax.microedition.lcdui.Image270 source)
```

Description:

Creates an immutable image from a source image. If the source image is mutable, an immutable copy is created and returned. If the source image is immutable, the implementation may simply return it without creating a new image. If an immutable source image contains transparency information, this information is copied to the new image unchanged.

This method is useful for placing the contents of mutable images into `Choice` objects. The application can create an off-screen image using the `createImage(w, h)`$_{276}$ method, draw into it using a `Graphics` object obtained with the `getGraphics()`$_{277}$ method, and then create an immutable copy of it with this method. The immutable copy may then be placed into `Choice` objects.

Parameters:

`source` - the source image to be copied

Returns: the new, immutable image

Throws:

`NullPointerException` - if `source` is `null`

createImage(Image, int, int, int, int, int)

Declaration:

```
public static javax.microedition.lcdui.Image270
            createImage(javax.microedition.lcdui.Image270 image, int x, int y,
            int width, int height, int transform)
```

Description:

Creates an immutable image using pixel data from the specified region of a source image, transformed as specified.

The source image may be mutable or immutable. For immutable source images, transparency information, if any, is copied to the new image unchanged.

On some devices, pre-transformed images may render more quickly than images that are transformed on the fly using `drawRegion`. However, creating such images does consume additional heap space, so this technique should be applied only to images whose rendering speed is critical.

The transform function used must be one of the following, as defined in the `Sprite`$_{365}$ class:
`Sprite.TRANS_NONE` - causes the specified image region to be copied unchanged
`Sprite.TRANS_ROT90` - causes the specified image region to be rotated clockwise by 90 degrees.
`Sprite.TRANS_ROT180` - causes the specified image region to be rotated clockwise by 180 degrees.
`Sprite.TRANS_ROT270` - causes the specified image region to be rotated clockwise by 270 degrees.
`Sprite.TRANS_MIRROR` - causes the specified image region to be reflected about its vertical center.
`Sprite.TRANS_MIRROR_ROT90` - causes the specified image region to be reflected about its vertical center and then rotated clockwise by 90 degrees.
`Sprite.TRANS_MIRROR_ROT180` - causes the specified image region to be reflected about its vertical center and then rotated clockwise by 180 degrees.
`Sprite.TRANS_MIRROR_ROT270` - causes the specified image region to be reflected about its vertical center and then rotated clockwise by 270 degrees.

The size of the returned image will be the size of the specified region with the transform applied. For example, if the region is `100 x 50` pixels and the transform is `TRANS_ROT90`, the returned image will be `50 x 100` pixels.

Note: If all of the following conditions are met, this method may simply return the source `Image` without creating a new one:

- the source image is immutable;
- the region represents the entire source image; and
- the transform is `TRANS_NONE`.

Parameters:

`image` - the source image to be copied from

`x` - the horizontal location of the region to be copied

`y` - the vertical location of the region to be copied

`width` - the width of the region to be copied

`height` - the height of the region to be copied

`transform` - the transform to be applied to the region

Returns: the new, immutable image

Throws:

`NullPointerException` - if `image` is `null`

`IllegalArgumentException` - if the region to be copied exceeds the bounds of the source image

`IllegalArgumentException` - if either `width` or `height` is zero or less

`IllegalArgumentException` - if the `transform` is not valid

Since: MIDP 2.0

createImage(InputStream)

Declaration:

```
public static javax.microedition.lcdui.Image(270) createImage(java.io.InputStream stream)
            throws IOException
```

Description:

Creates an immutable image from decoded image data obtained from an `InputStream`. This method blocks until all image data has been read and decoded. After this method completes (whether by returning or by throwing an exception) the stream is left open and its current position is undefined.

Parameters:

`stream` - the name of the resource containing the image data in one of the supported image formats

Returns: the created image

Throws:

`NullPointerException` - if `stream` is `null`

`java.io.IOException` - if an I/O error occurs, if the image data cannot be loaded, or if the image data cannot be decoded

Since: MIDP 2.0

createImage(int, int)

Declaration:

```
public static javax.microedition.lcdui.Image270 createImage(int width, int height)
```

Description:

Creates a new, mutable image for off-screen drawing. Every pixel within the newly created image is white. The width and height of the image must both be greater than zero.

Parameters:

`width` - the width of the new image, in pixels

`height` - the height of the new image, in pixels

Returns: the created image

Throws:

`IllegalArgumentException` - if either `width` or `height` is zero or less

createImage(String)

Declaration:

```
public static javax.microedition.lcdui.Image270 createImage(String name)
            throws IOException
```

Description:

Creates an immutable image from decoded image data obtained from the named resource. The name parameter is a resource name as defined by `Class.getResourceAsStream(name)`. The rules for resolving resource names are defined in the Application Resource Files section of the `java.lang` package documentation.

Parameters:

`name` - the name of the resource containing the image data in one of the supported image formats

Returns: the created image

Throws:

`NullPointerException` - if `name` is `null`

`java.io.IOException` - if the resource does not exist, the data cannot be loaded, or the image data cannot be decoded

createRGBImage(int[], int, int, boolean)

Declaration:

```
public static javax.microedition.lcdui.Image270 createRGBImage(int[] rgb, int width,
            int height, boolean processAlpha)
```

Description:

Creates an immutable image from a sequence of ARGB values, specified as `0xAARRGGBB`. The ARGB data within the `rgb` array is arranged horizontally from left to right within each row, row by row from top to bottom. If `processAlpha` is `true`, the high-order byte specifies opacity; that is, `0x00RRGGBB` specifies a fully transparent pixel and `0xFFRRGGBB` specifies a fully opaque pixel. Intermediate alpha values specify semitransparency. If the implementation does not support alpha blending for image rendering operations, it must replace any semitransparent pixels with fully transparent pixels. (See Alpha Processing for further discussion.) If `processAlpha` is `false`, the alpha values are ignored and all pixels must be treated as fully opaque.

Consider `P(a,b)` to be the value of the pixel located at column `a` and row `b` of the Image, where rows and columns are numbered downward from the top starting at zero, and columns are numbered rightward from the left starting at zero. This operation can then be defined as:

```
P(a, b) = rgb[a + b * width];
```

for

```
0 <= a < width
0 <= b < height
```

Parameters:

`rgb` - an array of ARGB values that composes the image

`width` - the width of the image

`height` - the height of the image

`processAlpha` - `true` if `rgb` has an alpha channel, `false` if all pixels are fully opaque

Returns: the created image

Throws:

`NullPointerException` - if `rgb` is `null`.

`IllegalArgumentException` - if either `width` or `height` is zero or less

`ArrayIndexOutOfBoundsException` - if the length of `rgb` is less than `width * height`.

Since: MIDP 2.0

getGraphics()

Declaration:

```
public javax.microedition.lcdui.Graphics247 getGraphics()
```

Description:

Creates a new `Graphics` object that renders to this image. This image must be mutable; it is illegal to call this method on an immutable image. The mutability of an image may be tested with the `isMutable()` method.

The newly created `Graphics` object has the following properties:

- the destination is this `Image` object;
- the clip region encompasses the entire `Image`;
- the current color is black;
- the font is the same as the font returned by `Font.getDefaultFont()`227;
- the stroke style is `SOLID`254; and
- the origin of the coordinate system is located at the upper-left corner of the Image.

The lifetime of `Graphics` objects created using this method is indefinite. They may be used at any time, by any thread.

Returns: a `Graphics` object with this image as its destination

Throws:
`IllegalStateException`[37] - if the image is immutable

getHeight()

Declaration:
```
public int getHeight()
```

Description:
Gets the height of the image in pixels. The value returned must reflect the actual height of the image when rendered.

Returns: height of the image

getRGB(int[], int, int, int, int, int, int)

Declaration:
```
public void getRGB(int[] rgbData, int offset, int scanlength, int x, int y, int width,
            int height)
```

Description:
Obtains ARGB pixel data from the specified region of this image and stores it in the provided array of integers. Each pixel value is stored in `0xAARRGGBB` format, where the high-order byte contains the alpha channel and the remaining bytes contain color components for red, green and blue, respectively. The alpha channel specifies the opacity of the pixel, where a value of `0x00` represents a pixel that is fully transparent and a value of `0xFF` represents a fully opaque pixel.

The returned values are not guaranteed to be identical to values from the original source, such as from `createRGBImage` or from a PNG image. Color values may be resampled to reflect the display capabilities of the device (for example, red, green or blue pixels may all be represented by the same gray value on a grayscale device). On devices that do not support alpha blending, the alpha value will be `0xFF` for opaque pixels and `0x00` for all other pixels (see Alpha Processing for further discussion.) On devices that support alpha blending, alpha channel values may be resampled to reflect the number of levels of semitransparency supported.

The `scanlength` specifies the relative offset within the array between the corresponding pixels of consecutive rows. In order to prevent rows of stored pixels from overlapping, the absolute value of `scanlength` must be greater than or equal to `width`. Negative values of `scanlength` are allowed. In all cases, this must result in every reference being within the bounds of the `rgbData` array.

Consider `P(a,b)` to be the value of the pixel located at column `a` and row `b` of the Image, where rows and columns are numbered downward from the top starting at zero, and columns are numbered rightward from the left starting at zero. This operation can then be defined as:

```
rgbData[offset + (a - x) + (b - y) * scanlength] = P(a, b);
```

for

x <= a < x + width
y <= b < y + height

The source rectangle is required to not exceed the bounds of the image. This means:

x >= 0
y >= 0
x + width <= image width
y + height <= image height

If any of these conditions is not met an `IllegalArgumentException` is thrown. Otherwise, in cases where `width <= 0` or `height <= 0`, no exception is thrown, and no pixel data is copied to `rgbData`.

Parameters:

`rgbData` - an array of integers in which the ARGB pixel data is stored

`offset` - the index into the array where the first ARGB value is stored

`scanlength` - the relative offset in the array between corresponding pixels in consecutive rows of the region

`x` - the x-coordinate of the upper left corner of the region

`y` - the y-coordinate of the upper left corner of the region

`width` - the width of the region

`height` - the height of the region

Throws:

`ArrayIndexOutOfBoundsException` - if the requested operation would attempt to access an element in the `rgbData` array whose index is either negative or beyond its length (the contents of the array are unchanged)

`IllegalArgumentException` - if the area being retrieved exceeds the bounds of the source image

`IllegalArgumentException` - if the absolute value of `scanlength` is less than `width`

`NullPointerException` - if `rgbData` is `null`

Since: MIDP 2.0

getWidth()

Declaration:

```
public int getWidth()
```

Description:

Gets the width of the image in pixels. The value returned must reflect the actual width of the image when rendered.

Returns: width of the image

isMutable()

Declaration:
```
public boolean isMutable()
```

Description:
Check if this image is mutable. Mutable images can be modified by rendering to them through a `Graphics` object obtained from the `getGraphics()` method of this object.

Returns: `true` if the image is mutable, `false` otherwise

javax.microedition.lcdui
ImageItem

Declaration

```
public class ImageItem extends Item287

Object
  |
  +--javax.microedition.lcdui.Item287
        |
        +--javax.microedition.lcdui.ImageItem
```

Description

An item that can contain an image.

Each `ImageItem` object contains a reference to an `Image`270 object. This `Image` may be mutable or immutable. If the `Image` is mutable, the effect is as if snapshot of its contents is taken at the time the `ImageItem` is constructed with this `Image` and when `setImage` is called with an `Image`. The snapshot is used whenever the contents of the `ImageItem` are to be displayed. Even if the application subsequently draws into the `Image`, the snapshot is not modified until the next call to `setImage`. The snapshot is *not* updated when the container of the `ImageItem` becomes current or becomes visible on the display. (This is because the application does not have control over exactly when `Displayables` and Items appear and disappear from the display.)

The value `null` may be specified for the image contents of an `ImageItem`. If this occurs (and if the label is also `null`) the `ImageItem` will occupy no space on the screen.

`ImageItem` contains layout directives that were originally defined in MIDP 1.0. These layout directives have been moved to the `Item`287 class and now apply to all items. The declarations are left in `ImageItem` for source compatibility purposes.

The `altText` parameter specifies a string to be displayed in place of the image if the image exceeds the capacity of the display. The `altText` parameter may be `null`.

Since: MIDP 1.0

Member Summary

Fields

static int	LAYOUT_CENTER282
static int	LAYOUT_DEFAULT282
static int	LAYOUT_LEFT283
static int	LAYOUT_NEWLINE_AFTER283
static int	LAYOUT_NEWLINE_BEFORE283
static int	LAYOUT_RIGHT283

Constructors

ImageItem(String label, Image img, int layout, String altText)283
ImageItem(String label, Image image, int layout, String altText, int appearanceMode)284

Member Summary

Methods

java.lang.String	getAltText()[285]
int	getAppearanceMode()[285]
Image	getImage()[285]
int	getLayout()[285]
void	setAltText(String text)[285]
void	setImage(Image img)[286]
void	setLayout(int layout)[286]

Inherited Member Summary

Fields inherited from class Item[287]

BUTTON[291], HYPERLINK[292], LAYOUT_2[292], LAYOUT_BOTTOM[292], LAYOUT_EXPAND[293], LAYOUT_SHRINK[294], LAYOUT_TOP[294], LAYOUT_VCENTER[294], LAYOUT_VEXPAND[294], LAYOUT_VSHRINK[294], PLAIN[295]

Methods inherited from class Item[287]

addCommand(Command)[295], getLabel()[295], getMinimumHeight()[296], getMinimumWidth()[296], getPreferredHeight()[296], getPreferredWidth()[296], notifyStateChanged()[297], removeCommand(Command)[297], setDefaultCommand(Command)[298], setItemCommandListener(ItemCommandListener)[298], setLabel(String)[298], setPreferredSize(int, int)[299]

Methods inherited from class Object

equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Fields

LAYOUT_CENTER

Declaration:
`public static final int LAYOUT_CENTER`

Description:
See Item.LAYOUT_CENTER[292].

Value 3 is assigned to LAYOUT_CENTER.

LAYOUT_DEFAULT

Declaration:
`public static final int LAYOUT_DEFAULT`

Description:
See Item.LAYOUT_DEFAULT[292].

Value 0 is assigned to LAYOUT_DEFAULT.

LAYOUT_LEFT

Declaration:
```
public static final int LAYOUT_LEFT
```

Description:
See `Item.LAYOUT_LEFT`[293].

Value `1` is assigned to `LAYOUT_LEFT`.

LAYOUT_NEWLINE_AFTER

Declaration:
```
public static final int LAYOUT_NEWLINE_AFTER
```

Description:
See `Item.LAYOUT_NEWLINE_AFTER`[293].

Value `0x200` is assigned to `LAYOUT_NEWLINE_AFTER`.

LAYOUT_NEWLINE_BEFORE

Declaration:
```
public static final int LAYOUT_NEWLINE_BEFORE
```

Description:
See `Item.LAYOUT_NEWLINE_BEFORE`[293].

Value `0x100` is assigned to `LAYOUT_NEWLINE_BEFORE`.

LAYOUT_RIGHT

Declaration:
```
public static final int LAYOUT_RIGHT
```

Description:
See `Item.LAYOUT_RIGHT`[293].

Value `2` is assigned to `LAYOUT_RIGHT`.

Constructors

ImageItem(String, Image, int, String)

Declaration:
```
public ImageItem(String label, javax.microedition.lcdui.Image[270] img, int layout,
            String altText)
```

Description:
Creates a new `ImageItem` with the given label, image, layout directive, and alternate text string. Calling this constructor is equivalent to calling

ImageItem(label, image, layout, altText, PLAIN);

ImageItem(String, Image, int, String, int)

Parameters:

`label` - the label string

`img` - the image, can be mutable or immutable

`layout` - a combination of layout directives

`altText` - the text that may be used in place of the image

Throws:

`IllegalArgumentException` - if the `layout` value is not a legal combination of directives

See Also: `ImageItem(String, Image, int, String, int)`$_{284}$

ImageItem(String, Image, int, String, int)

Declaration:

```
public ImageItem(String label, javax.microedition.lcdui.Image image, int layout,
            String altText, int appearanceMode)
```

Description:

Creates a new `ImageItem` object with the given label, image, layout directive, alternate text string, and appearance mode. Either label or alternative text may be present or `null`.

The `appearanceMode` parameter (see Appearance Modes) is a hint to the platform of the application's intended use for this `ImageItem`. To provide hyperlink- or button-like behavior, the application should associate a default `Command` with this `ImageItem` and add an `ItemCommandListener` to this `ImageItem`.

Here is an example showing the use of an `ImageItem` as a button:

```
ImageItem imgItem =
   new ImageItem("Default: ", img,
            Item.LAYOUT_CENTER, null,
            Item.BUTTON);
imgItem.setDefaultCommand(
   new Command("Set", Command.ITEM, 1);
// icl is ItemCommandListener
imgItem.setItemCommandListener(icl);
```

Parameters:

`label` - the label string

`image` - the image, can be mutable or immutable

`layout` - a combination of layout directives

`altText` - the text that may be used in place of the image

`appearanceMode` - the appearance mode of the `ImageItem`, one of `Item.PLAIN`$_{295}$, `Item.HYPERLINK`$_{292}$, or `Item.BUTTON`$_{291}$

Throws:

`IllegalArgumentException` - if the `layout` value is not a legal combination of directives

`IllegalArgumentException` - if `appearanceMode` invalid

Since: MIDP 2.0

Methods

getAltText()

Declaration:
`public String getAltText()`

Description:
Gets the text string to be used if the image exceeds the device's capacity to display it.

Returns: the alternate text value, or `null` if none

See Also: `setAltText(String)`[285]

getAppearanceMode()

Declaration:
`public int getAppearanceMode()`

Description:
Returns the appearance mode of the `ImageItem`. See Appearance Modes.

Returns: the appearance mode value, one of `Item.PLAIN`[295], `Item.HYPERLINK`[292], or `Item.BUTTON`[291]

Since: MIDP 2.0

getImage()

Declaration:
`public javax.microedition.lcdui.Image`[270] `getImage()`

Description:
Gets the image contained within the `ImageItem`, or `null` if there is no contained image.

Returns: image used by the `ImageItem`

See Also: `setImage(Image)`[286]

getLayout()

Declaration:
`public int getLayout()`

Description:
Gets the layout directives used for placing the image.

Overrides: `getLayout`[295] in class `Item`[287]

Returns: a combination of layout directive values

See Also: `setLayout(int)`[286]

setAltText(String)

Declaration:
`public void setAltText(String text)`

Description:
Sets the alternate text of the `ImageItem`, or `null` if no alternate text is provided.

setImage(Image)

Parameters:

`text` - the new alternate text

See Also: `getAltText()`[285]

setImage(Image)

Declaration:

```
public void setImage(javax.microedition.lcdui.Image[270] img)
```

Description:

Sets the `Image` object contained within the `ImageItem`. The image may be mutable or immutable. If `img` is `null`, the `ImageItem` is set to be empty. If `img` is mutable, the effect is as if a snapshot is taken of `img's` contents immediately prior to the call to `setImage`. This snapshot is used whenever the contents of the `ImageItem` are to be displayed. If `img` is already the `Image` of this `ImageItem`, the effect is as if a new snapshot of img's contents is taken. Thus, after painting into a mutable image contained by an `ImageItem`, the application can call

```
imageItem.setImage(imageItem.getImage());
```

to refresh the `ImageItem's` snapshot of its Image.

If the `ImageItem` is visible on the display when the snapshot is updated through a call to `setImage`, the display is updated with the new snapshot as soon as it is feasible for the implementation to so do.

Parameters:

`img` - the `Image` for this `ImageItem`, or `null` if none

See Also: `getImage()`[285]

setLayout(int)

Declaration:

```
public void setLayout(int layout)
```

Description:

Sets the layout directives.

Overrides: `setLayout`[299] in class `Item`[287]

Parameters:

`layout` - a combination of layout directive values

Throws:

`IllegalArgumentException` - if the value of `layout` is not a valid combination of layout directives

See Also: `getLayout()`[285]

javax.microedition.lcdui
Item

Declaration

```
public abstract class Item

Object
  |
  +--javax.microedition.lcdui.Item
```

Direct Known Subclasses: ChoiceGroup[166], CustomItem[184], DateField[201], Gauge[240], ImageItem[281], Spacer[316], StringItem[319], TextField[330]

Description

A superclass for components that can be added to a Form[231] . All `Item` objects have a label field, which is a string that is attached to the item. The label is typically displayed near the component when it is displayed within a screen. The label should be positioned on the same horizontal row as the item or directly above the item. The implementation should attempt to distinguish label strings from other textual content, possibly by displaying the label in a different font, aligning it to a different margin, or appending a colon to it if it is placed on the same line as other string content. If the screen is scrolling, the implementation should try to keep the label visible at the same time as the `Item`.

In some cases, when the user attempts to interact with an `Item`, the system will switch to a system-generated screen where the actual interaction takes place. If this occurs, the label will generally be carried along and displayed within this new screen in order to provide the user with some context for the operation. For this reason it is recommended that applications supply a label to all interactive Item objects. However, this is not required, and a `null` value for a label is legal and specifies the absence of a label.

Item Layout

An `Item's` layout within its container is influenced through layout directives:

- `LAYOUT_DEFAULT`
- `LAYOUT_LEFT`
- `LAYOUT_RIGHT`
- `LAYOUT_CENTER`
- `LAYOUT_TOP`
- `LAYOUT_BOTTOM`
- `LAYOUT_VCENTER`
- `LAYOUT_NEWLINE_BEFORE`
- `LAYOUT_NEWLINE_AFTER`
- `LAYOUT_SHRINK`
- `LAYOUT_VSHRINK`
- `LAYOUT_EXPAND`
- `LAYOUT_VEXPAND`

- LAYOUT_2

The LAYOUT_DEFAULT directive indicates that the container's default layout policy is to be used for this item. LAYOUT_DEFAULT has the value zero and has no effect when combined with other layout directives. It is useful within programs in order to document the programmer's intent.

The LAYOUT_LEFT, LAYOUT_RIGHT, and LAYOUT_CENTER directives indicate horizontal alignment and are mutually exclusive. Similarly, the LAYOUT_TOP, LAYOUT_BOTTOM, and LAYOUT_VCENTER directives indicate vertical alignment and are mutually exclusive.

A horizontal alignment directive, a vertical alignment directive, and any combination of other layout directives may be combined using the bit-wise OR operator (|) to compose a layout directive value. Such a value is used as the parameter to the setLayout(int)[299] method and is the return value from the getLayout()[295] method.

Some directives have no defined behavior in some contexts. A layout directive is ignored if its behavior is not defined for the particular context within which the Item resides.

A complete specification of the layout of Items within a Form is given here.

Item Sizes

Items have two explicit size concepts: the *minimum* size and the *preferred* size. Both the minimum and the preferred sizes refer to the total area of the Item, which includes space for the Item's contents, the Item's label, as well as other space that is significant to the layout policy. These sizes do not include space that is not significant for layout purposes. For example, if the addition of a label to an Item would cause other Items to move in order to make room, then the space occupied by this label is significant to layout and is counted as part of the Item's minimum and preferred sizes. However, if an implementation were to place the label in a margin area reserved exclusively for labels, this would not affect the layout of neighboring Items. In this case, the space occupied by the label would not be considered part of the minimum and preferred sizes.

The minimum size is the smallest size at which the Item can function and display its contents, though perhaps not optimally. The minimum size may be recomputed whenever the Item's contents changes.

The preferred size is generally a size based on the Item's contents and is the smallest size at which no information is clipped and text wrapping (if any) is kept to a tolerable minimum. The preferred size may be recomputed whenever the Item's contents changes. The application can *lock* the preferred width or preferred height (or both) by supplying specific values for parameters to the setPreferredSize[299] method. The manner in which an Item fits its contents within an application-specified preferred size is implementation-specific. However, it is recommended that textual content be word-wrapped to fit the preferred size set by the application. The application can *unlock* either or both dimensions by supplying the value -1 for parameters to the setPreferredSize method.

When an Item is created, both the preferred width and height are unlocked. In this state, the implementation computes the preferred width and height based on the Item's contents, possibly including other relevant factors such as the Item's graphic design and the screen dimensions. After having locked either the preferred width or height, the application can restore the initial, unlocked state by calling setPreferredSize(-1, -1).

The application can lock one dimension of the preferred size and leave the other unlocked. This causes the system to compute an appropriate value for the unlocked dimension based on arranging the contents to fit the locked dimension. If the contents changes, the size on the unlocked dimension is recomputed to reflect the new contents, but the size on the locked dimension remains unchanged. For example, if the application called setPreferredSize(50, -1), the preferred width would be locked at 50 pixels and the preferred height would be computed based on the Item's contents. Similarly, if the application called setPreferredSize(-1, 60), the preferred height would be locked at 60 pixels and the preferred width

would be computed based on the `Item's` contents. This feature is particularly useful for `Items` with textual content that can be line wrapped.

The application can also lock both the preferred width and height to specific values. The `Item's` contents are truncated or padded as necessary to honor this request. For `Items` containing text, the text should be wrapped to the specified width, and any truncation should occur at the end of the text.

`Items` also have an implicit maximum size provided by the implementation. The maximum width is typically based on the width of the screen space available to a `Form`. Since `Forms` can scroll vertically, the maximum height should typically not be based on the height of the available screen space.

If the application attempts to lock a preferred size dimension to a value smaller than the minimum or larger than the maximum, the implementation may disregard the requested value and instead use either the minimum or maximum as appropriate. If this occurs, the actual values used must be visible to the application via the values returned from the `getPreferredWidth`$_{296}$ and `getPreferredHeight`$_{296}$ methods.

Commands

A `Command` is said to be present on an `Item` if the `Command` has been added to this `Item` with a prior call to `addCommand(Command)`$_{295}$ or `setDefaultCommand(Command)`$_{298}$ and if the `Command` has not been removed with a subsequent call to `removeCommand(Command)`$_{297}$. `Commands` present on an item should have a command type of `ITEM`. However, it is not an error for a command whose type is other than `ITEM` to be added to an item. For purposes of presentation and placement within its user interface, the implementation is allowed to treat a command's items as if they were of type `ITEM`.

`Items` may have a *default* `Command`. This state is controlled by the `setDefaultCommand(Command)`$_{298}$ method. The default `Command` is eligible to be bound to a special platform-dependent user gesture. The implementation chooses which gesture is the most appropriate to initiate the default command on that particular `Item`. For example, on a device that has a dedicated selection key, pressing this key might invoke the item's default command. Or, on a stylus-based device, tapping on the `Item` might invoke its default command. Even if it can be invoked through a special gesture, the default command should also be invokable in the same fashion as other item commands.

It is possible that on some devices there is no special gesture suitable for invoking the default command on an item. In this case the default command must be accessible to the user in the same fashion as other item commands. The implementation may use the state of a command being the default in deciding where to place the command in its user interface.

It is possible for an `Item` not to have a default command. In this case, the implementation may bind its special user gesture (if any) for another purpose, such as for displaying a menu of commands. The default state of an `Item` is not to have a default command. An `Item` may be set to have no default `Command` by removing it from the `Item` or by passing `null` to the `setDefaultCommand()` method.

The same command may occur on more than one `Item` and also on more than one `Displayable`. If this situation occurs, the user must be provided with distinct gestures to invoke that command on each `Item` or `Displayable` on which it occurs, while those `Items` or `Displayables` are visible on the display. When the user invokes the command, the listener (`CommandListener` or `ItemCommandListener` as appropriate) of just the object on which the command was invoked will be called.

Adding commands to an `Item` may affect its appearance, the way it is laid out, and the traversal behavior. For example, the presence of commands on an `Item` may cause row breaks to occur, or it may cause additional graphical elements (such as a menu icon) to appear. In particular, if a `StringItem` whose appearance mode is `PLAIN` (see below) is given one or more `Commands`, the implementation is allowed to treat it as if it had a different appearance mode.

Appearance Modes

The `StringItem` and `ImageItem` classes have an *appearance mode* attribute that can be set in their constructors. This attribute can have one of the values `PLAIN`[295], `HYPERLINK`[292], or `BUTTON`[291]. An appearance mode of `PLAIN` is typically used for non-interactive display of textual or graphical material. The appearance mode values do not have any side effects on the interactivity of the item. In order to be interactive, the item must have one or more `Commands` (preferably with a default command assigned), and it must have a `CommandListener` that receives notification of `Command` invocations. The appearance mode values also do not have any effect on the semantics of `Command` invocation on the item. For example, setting the appearance mode of a `StringItem` to be `HYPERLINK` requests that the implementation display the string contents as if they were a hyperlink in a browser. It is the application's responsibility to attach a `Command` and a listener to the `StringItem` that provide behaviors that the user would expect from invoking an operation on a hyperlink, such as loading the referent of the link or adding the link to the user's set of bookmarks.

Setting the appearance mode of an `Item` to be other than `PLAIN` may affect its minimum, preferred, and maximum sizes, as well as the way it is laid out. For example, a `StringItem` with an appearance mode of `BUTTON` should not be wrapped across rows. (However, a `StringItem` with an appearance mode of `HYPERLINK` should be wrapped the same way as if its appearance mode is `PLAIN`.)

A `StringItem` or `ImageItem` in `BUTTON` mode can be used to create a button-based user interface. This can easily lead to applications that are inconvenient to use. For example, in a traversal-based system, users must navigate to a button before they can invoke any commands on it. If buttons are spread across a long `Form`, users may be required to perform a considerable amount of navigation in order to discover all the available commands. Furthermore, invoking a command from a button at the other end of the `Form` can be quite cumbersome. Traversal-based systems often provide a means of invoking commands from anywhere (such as from a menu), without the need to traverse to a particular item. Instead of adding a command to a button and placing that button into a `Form`, it would often be more appropriate and convenient for users if that command were added directly to the `Form`. Buttons should be used only in cases where direct user interaction with the item's string or image contents is essential to the user's understanding of the commands that can be invoked from that item.

Default State

Unless otherwise specified by a subclass, the default state of newly created `Items` is as follows:

- the `Item` is not contained within ("owned by") any container;
- there are no `Commands` present;
- the default `Command` is `null`;
- the `ItemCommandListener` is `null`;
- the layout directive value is `LAYOUT_DEFAULT`; and
- both the preferred width and preferred height are unlocked.

Since: MIDP 1.0

Member Summary

Fields

```
static int  BUTTON[291]
static int  HYPERLINK[292]
static int  LAYOUT_2[292]
static int  LAYOUT_BOTTOM[292]
```

Member Summary	
static int	LAYOUT_CENTER[292]
static int	LAYOUT_DEFAULT[292]
static int	LAYOUT_EXPAND[293]
static int	LAYOUT_LEFT[293]
static int	LAYOUT_NEWLINE_AFTER[293]
static int	LAYOUT_NEWLINE_BEFORE[293]
static int	LAYOUT_RIGHT[293]
static int	LAYOUT_SHRINK[294]
static int	LAYOUT_TOP[294]
static int	LAYOUT_VCENTER[294]
static int	LAYOUT_VEXPAND[294]
static int	LAYOUT_VSHRINK[294]
static int	PLAIN[295]
Methods	
void	addCommand(Command cmd)[295]
java.lang.String	getLabel()[295]
int	getLayout()[295]
int	getMinimumHeight()[296]
int	getMinimumWidth()[296]
int	getPreferredHeight()[296]
int	getPreferredWidth()[296]
void	notifyStateChanged()[297]
void	removeCommand(Command cmd)[297]
void	setDefaultCommand(Command cmd)[298]
void	setItemCommandListener(ItemCommandListener l)[298]
void	setLabel(String label)[298]
void	setLayout(int layout)[299]
void	setPreferredSize(int width, int height)[299]

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Fields

BUTTON

Declaration:
`public static final int BUTTON`

Description:
An appearance mode value indicating that the `Item` is to appear as a button.

Value `2` is assigned to `BUTTON`.

Since: MIDP 2.0

HYPERLINK

Declaration:
`public static final int HYPERLINK`

Description:
An appearance mode value indicating that the `Item` is to appear as a hyperlink.

Value `1` is assigned to `HYPERLINK`.

Since: MIDP 2.0

LAYOUT_2

Declaration:
`public static final int LAYOUT_2`

Description:
A layout directive indicating that new MIDP 2.0 layout rules are in effect for this `Item`. If this bit is clear, indicates that MIDP 1.0 layout behavior applies to this `Item`.

Value `0x4000` is assigned to `LAYOUT_2`.

Since: MIDP 2.0

LAYOUT_BOTTOM

Declaration:
`public static final int LAYOUT_BOTTOM`

Description:
A layout directive indicating that this `Item` should have a bottom-aligned layout.

Value `0x20` is assigned to `LAYOUT_BOTTOM`.

Since: MIDP 2.0

LAYOUT_CENTER

Declaration:
`public static final int LAYOUT_CENTER`

Description:
A layout directive indicating that this `Item` should have a horizontally centered layout.

Value `3` is assigned to `LAYOUT_CENTER`.

Since: MIDP 2.0

LAYOUT_DEFAULT

Declaration:
`public static final int LAYOUT_DEFAULT`

Description:
A layout directive indicating that this `Item` should follow the default layout policy of its container.

Value `0` is assigned to `LAYOUT_DEFAULT`.

Since: MIDP 2.0

LAYOUT_EXPAND

Declaration:
```
public static final int LAYOUT_EXPAND
```

Description:
A layout directive indicating that this `Item's` width may be increased to fill available space.

Value `0x800` is assigned to `LAYOUT_EXPAND`.

Since: MIDP 2.0

LAYOUT_LEFT

Declaration:
```
public static final int LAYOUT_LEFT
```

Description:
A layout directive indicating that this `Item` should have a left-aligned layout.

Value `1` is assigned to `LAYOUT_LEFT`.

Since: MIDP 2.0

LAYOUT_NEWLINE_AFTER

Declaration:
```
public static final int LAYOUT_NEWLINE_AFTER
```

Description:
A layout directive indicating that this `Item` should the last on its line or row, and that the next `Item` (if any) in the container should be placed on a new line or row.

Value `0x200` is assigned to `LAYOUT_NEWLINE_AFTER`.

Since: MIDP 2.0

LAYOUT_NEWLINE_BEFORE

Declaration:
```
public static final int LAYOUT_NEWLINE_BEFORE
```

Description:
A layout directive indicating that this `Item` should be placed at the beginning of a new line or row.

Value `0x100` is assigned to `LAYOUT_NEWLINE_BEFORE`.

Since: MIDP 2.0

LAYOUT_RIGHT

Declaration:
```
public static final int LAYOUT_RIGHT
```

Description:
A layout directive indicating that this `Item` should have a right-aligned layout.

Value `2` is assigned to `LAYOUT_RIGHT`.

Since: MIDP 2.0

LAYOUT_SHRINK

Declaration:
`public static final int LAYOUT_SHRINK`

Description:
A layout directive indicating that this `Item's` width may be reduced to its minimum width.

Value `0x400` is assigned to `LAYOUT_SHRINK`

Since: MIDP 2.0

LAYOUT_TOP

Declaration:
`public static final int LAYOUT_TOP`

Description:
A layout directive indicating that this `Item` should have a top-aligned layout.

Value `0x10` is assigned to `LAYOUT_TOP`.

Since: MIDP 2.0

LAYOUT_VCENTER

Declaration:
`public static final int LAYOUT_VCENTER`

Description:
A layout directive indicating that this `Item` should have a vertically centered layout.

Value `0x30` is assigned to `LAYOUT_VCENTER`.

Since: MIDP 2.0

LAYOUT_VEXPAND

Declaration:
`public static final int LAYOUT_VEXPAND`

Description:
A layout directive indicating that this `Item's` height may be increased to fill available space.

Value `0x2000` is assigned to `LAYOUT_VEXPAND`.

Since: MIDP 2.0

LAYOUT_VSHRINK

Declaration:
`public static final int LAYOUT_VSHRINK`

Description:
A layout directive indicating that this `Item's` height may be reduced to its minimum height.

Value `0x1000` is assigned to `LAYOUT_VSHRINK`.

Since: MIDP 2.0

PLAIN

Declaration:
```
public static final int PLAIN
```

Description:
An appearance mode value indicating that the `Item` is to have a normal appearance.

Value `0` is assigned to `PLAIN`.

Since: MIDP 2.0

Methods

addCommand(Command)

Declaration:
```
public void addCommand(javax.microedition.lcdui.Command cmd)
```

Description:
Adds a context sensitive `Command` to the item. The semantic type of `Command` should be `ITEM`. The implementation will present the command only when the item is active, for example, highlighted.

If the added command is already in the item (tested by comparing the object references), the method has no effect. If the item is actually visible on the display, and this call affects the set of visible commands, the implementation should update the display as soon as it is feasible to do so.

It is illegal to call this method if this `Item` is contained within an `Alert`.

Parameters:
`cmd` - the command to be added

Throws:
`IllegalStateException` - if this `Item` is contained within an `Alert`

`NullPointerException` - if cmd is `null`

Since: MIDP 2.0

getLabel()

Declaration:
```
public String getLabel()
```

Description:
Gets the label of this `Item` object.

Returns: the label string

See Also: `setLabel(String)`

getLayout()

Declaration:
```
public int getLayout()
```

Description:
Gets the layout directives used for placing the item.

Returns: a combination of layout directive values

Since: MIDP 2.0

See Also: `setLayout(int)`$_{299}$

getMinimumHeight()

Declaration:
```
public int getMinimumHeight()
```

Description:
Gets the minimum height for this `Item`. This is a height at which the item can function and display its contents, though perhaps not optimally. See Item Sizes for a complete discussion.

Returns: the minimum height of the item

Since: MIDP 2.0

getMinimumWidth()

Declaration:
```
public int getMinimumWidth()
```

Description:
Gets the minimum width for this `Item`. This is a width at which the item can function and display its contents, though perhaps not optimally. See Item Sizes for a complete discussion.

Returns: the minimum width of the item

Since: MIDP 2.0

getPreferredHeight()

Declaration:
```
public int getPreferredHeight()
```

Description:
Gets the preferred height of this `Item`. If the application has locked the height to a specific value, this method returns that value. Otherwise, the return value is computed based on the `Item's` contents, possibly with respect to the `Item's` preferred width if it is locked. See Item Sizes for a complete discussion.

Returns: the preferred height of the `Item`

Since: MIDP 2.0

See Also: `getPreferredWidth()`$_{296}$, `setPreferredSize(int, int)`$_{299}$

getPreferredWidth()

Declaration:
```
public int getPreferredWidth()
```

Description:
Gets the preferred width of this `Item`. If the application has locked the width to a specific value, this method returns that value. Otherwise, the return value is computed based on the `Item's` contents, possibly with respect to the `Item's` preferred height if it is locked. See Item Sizes for a complete discussion.

Returns: the preferred width of the Item

Since: MIDP 2.0

See Also: `getPreferredHeight()`$_{296}$, `setPreferredSize(int, int)`$_{299}$

notifyStateChanged()

Declaration:
```
public void notifyStateChanged()
```

Description:
Causes this `Item's` containing `Form` to notify the `Item's` `ItemStateListener`[301] . The application calls this method to inform the listener on the `Item` that the `Item's` state has been changed in response to an action. Even though this method simply causes a call to another part of the application, this mechanism is useful for decoupling the implementation of an `Item` (in particular, the implementation of a `CustomItem`, though this also applies to subclasses of other items) from the consumer of the item.

If an edit was performed by invoking a separate screen, and the editor now wishes to "return" to the form which contained the selected `Item`, the preferred method is `Display.setCurrent(Item)` instead of `Display.setCurrent(Displayable)`, because it allows the `Form` to restore focus to the `Item` that initially invoked the editor.

In order to make sure that the documented behavior of `ItemStateListener` is maintained, it is up to the caller (application) to guarantee that this function is not called unless:

- the `Item's` value has actually been changed, and
- the change was the result of a user action (an "edit") and NOT as a result of state change via calls to `Item's` APIs

The call to `ItemStateListener.itemStateChanged` may be delayed in order to be serialized with the event stream. The `notifyStateChanged` method does not block awaiting the completion of the `itemStateChanged` method.

Throws:
`IllegalStateException`[37] - if the `Item` is not owned by a `Form`

Since: MIDP 2.0

removeCommand(Command)

Declaration:
```
public void removeCommand(javax.microedition.lcdui.Command[175] cmd)
```

Description:
Removes the context sensitive command from item. If the command is not in the `Item` (tested by comparing the object references), the method has no effect. If the `Item` is actually visible on the display, and this call affects the set of visible commands, the implementation should update the display as soon as it is feasible to do so. If the command to be removed happens to be the default command, the command is removed and the default command on this Item is set to `null`. The following code:

```
// Command c is the default command on Item item
item.removeCommand(c);
```

is equivalent to the following code:

```
// Command c is the default command on Item item
item.setDefaultCommand(null);
item.removeCommand(c);
```

Parameters:
`cmd` - the command to be removed

Since: MIDP 2.0

setDefaultCommand(Command)

Declaration:
`public void setDefaultCommand(javax.microedition.lcdui.Command`[175] `cmd)`

Description:
Sets default `Command` for this `Item`. If the `Item` previously had a default `Command`, that `Command` is no longer the default, but it remains present on the `Item`.

If not `null`, the `Command` object passed becomes the default `Command` for this `Item`. If the `Command` object passed is not currently present on this `Item`, it is added as if `addCommand(Command)`[295] had been called before it is made the default `Command`.

If `null` is passed, the `Item` is set to have no default `Command`. The previous default `Command`, if any, remains present on the `Item`.

It is illegal to call this method if this `Item` is contained within an `Alert`.

Parameters:
`cmd` - the command to be used as this `Item's` default `Command`, or `null` if there is to be no default command

Throws:
`IllegalStateException`[37] - if this `Item` is contained within an `Alert`

Since: MIDP 2.0

setItemCommandListener(ItemCommandListener)

Declaration:
`public void setItemCommandListener(javax.microedition.lcdui.ItemCommandListener`[300] `l)`

Description:
Sets a listener for `Commands` to this `Item`, replacing any previous `ItemCommandListener`. A `null` reference is allowed and has the effect of removing any existing listener.

It is illegal to call this method if this `Item` is contained within an `Alert`.

Parameters:
`l` - the new listener, or `null`.

Throws:
`IllegalStateException`[37] - if this `Item` is contained within an `Alert`

Since: MIDP 2.0

setLabel(String)

Declaration:
`public void setLabel(String label)`

Description:
Sets the label of the `Item`. If `label` is `null`, specifies that this item has no label.

It is illegal to call this method if this `Item` is contained within an `Alert`.

Parameters:
`label` - the label string

Throws:
`IllegalStateException`[37] - if this `Item` is contained within an `Alert`

See Also: `getLabel()`[295]

setLayout(int)

Declaration:
```
public void setLayout(int layout)
```

Description:
Sets the layout directives for this item.

It is illegal to call this method if this `Item` is contained within an `Alert`.

Parameters:
`layout` - a combination of layout directive values for this item

Throws:
`IllegalArgumentException` - if the value of layout is not a bit-wise OR combination of layout directives

`IllegalStateException`[37] - if this `Item` is contained within an `Alert`

Since: MIDP 2.0

See Also: `getLayout()`[295]

setPreferredSize(int, int)

Declaration:
```
public void setPreferredSize(int width, int height)
```

Description:
Sets the preferred width and height for this `Item`. Values for width and height less than `-1` are illegal. If the width is between zero and the minimum width, inclusive, the minimum width is used instead. If the height is between zero and the minimum height, inclusive, the minimum height is used instead.

Supplying a width or height value greater than the minimum width or height *locks* that dimension to the supplied value. The implementation may silently enforce a maximum dimension for an `Item` based on factors such as the screen size. Supplying a value of `-1` for the width or height unlocks that dimension. See Item Sizes for a complete discussion.

It is illegal to call this method if this `Item` is contained within an `Alert`.

Parameters:
`width` - the value to which the width should be locked, or `-1` to unlock

`height` - the value to which the height should be locked, or `-1` to unlock

Throws:
`IllegalArgumentException` - if width or height is less than `-1`

`IllegalStateException`[37] - if this `Item` is contained within an `Alert`

Since: MIDP 2.0

See Also: `getPreferredHeight()`[296], `getPreferredWidth()`[296]

javax.microedition.lcdui
ItemCommandListener

Declaration
`public interface` **`ItemCommandListener`**

Description
A listener type for receiving notification of commands that have been invoked on `Item`[287] objects. An `Item` can have `Commands` associated with it. When such a command is invoked, the application is notified by having the `commandAction()`[300] method called on the `ItemCommandListener` that had been set on the `Item` with a call to `setItemCommandListener()`[298].

Since: MIDP 2.0

Member Summary	
Methods	
	`void commandAction(Command c, Item item)`[300]

Methods

commandAction(Command, Item)

Declaration:
```
public void commandAction(javax.microedition.lcdui.Command[175] c,
            javax.microedition.lcdui.Item[287] item)
```

Description:
Called by the system to indicate that a command has been invoked on a particular item.

Parameters:
`c` - the `Command` that was invoked

`item` - the `Item` on which the command was invoked

javax.microedition.lcdui
ItemStateListener

Declaration
```
public interface ItemStateListener
```

Description
This interface is used by applications which need to receive events that indicate changes in the internal state of the interactive items within a `Form`[231] screen.

Since: MIDP 1.0

See Also: `Form.setItemStateListener(ItemStateListener)`[239]

Member Summary	
Methods	
`void`	`itemStateChanged(Item item)`[301]

Methods

itemStateChanged(Item)

Declaration:
```
public void itemStateChanged(javax.microedition.lcdui.Item[287] item)
```

Description:
Called when internal state of an `Item` has been changed by the user. This happens when the user:

- changes the set of selected values in a `ChoiceGroup`;
- adjusts the value of an interactive `Gauge`;
- enters or modifies the value in a `TextField`;
- enters a new date or time in a `DateField`; and
- `Item.notifyStateChanged()`[297] was called on an `Item`.

It is up to the device to decide when it considers a new value to have been entered into an `Item`. For example, implementations of text editing within a `TextField` vary greatly from device to device.

In general, it is not expected that the listener will be called after every change is made. However, if an item's value has been changed, the listener will be called to notify the application of the change before it is called for a change on another item, and before a command is delivered to the `Form's CommandListener`. For implementations that have the concept of an input focus, the listener should be called no later than when the focus moves away from an item whose state has been changed. The listener should be called only if the item's value has actually been changed.

The listener is not called if the application changes the value of an interactive item.

itemStateChanged(Item)

Parameters:

`item` - the item that was changed

javax.microedition.lcdui
List

Declaration

```
public class List extends Screen315 implements Choice155

Object
  |
  +--javax.microedition.lcdui.Displayable218
        |
        +--javax.microedition.lcdui.Screen315
              |
              +--javax.microedition.lcdui.List
```

All Implemented Interfaces: Choice155

Description

A `Screen` containing list of choices. Most of its behavior is common with class `ChoiceGroup`166, and their common API. The different `List` types in particular, are defined in interface `Choice`155. When a `List` is present on the display, the user can interact with it by selecting elements and possibly by traversing and scrolling among them. Traversing and scrolling operations do not cause application-visible events. The system notifies the application only when a `Command`175 is invoked by notifying its `CommandListener`183. The `List` class also supports a select command that may be invoked specially depending upon the capabilities of the device.

The notion of a *select* operation on a `List` element is central to the user's interaction with the `List`. On devices that have a dedicated hardware "select" or "go" key, the select operation is implemented with that key. Devices that do not have a dedicated key must provide another means to do the select operation, for example, using a soft key. The behavior of the select operation within the different types of lists is described in the following sections.

`List` objects may be created with `Choice` types of `Choice.EXCLUSIVE`158, `Choice.MULTIPLE`158, and `Choice.IMPLICIT`158. The `Choice` type `Choice.POPUP`158 is not allowed on `List` objects.

Selection in `EXCLUSIVE` and `MULTIPLE` Lists

The select operation is not associated with a `Command` object, so the application has no means of setting a label for it or being notified when the operation is performed. In `Lists` of type `EXCLUSIVE`, the select operation selects the target element and deselects the previously selected element. In `Lists` of type `MULTIPLE`, the select operation toggles the selected state of the target element, leaving the selected state of other elements unchanged. Devices that implement the select operation using a soft key will need to provide a label for it. The label should be something similar to "Select" for `Lists` of type `EXCLUSIVE`, and it should be something similar to "Mark" or "Unmark" for `Lists` of type `MULTIPLE`.

Selection in `IMPLICIT` Lists

The select operation is associated with a `Command` object referred to as the *select command*. When the user performs the select operation, the system will invoke the select command by notifying the `List's` `CommandListener`183. The default select command is the system-provided command `SELECT_COMMAND`. The select command may be modified by the application through use of the `setSelectCommand`312 method. Devices that implement the select operation using a soft key will use the label from the select command. If the select command is `SELECT_COMMAND`, the device may choose to

provide its own label instead of using the label attribute of `SELECT_COMMAND`. Applications should generally provide their own select command to replace `SELECT_COMMAND`. This allows applications to provide a meaningful label, instead of relying on the one provided by the system for `SELECT_COMMAND`. The implementation must *not* invoke the select command if there are no elements in the `List`, because if the `List` is empty the selection does not exist. In this case the implementation should remove or disable the select command if it would appear explicitly on a soft button or in a menu. Other commands can be invoked normally when the `List` is empty.

Use of `IMPLICIT` Lists

`IMPLICIT Lists` can be used to construct menus by providing operations as `List` elements. The application provides a `Command` that is used to select a `List` element and then defines this `Command` to be used as the select command. The application must also register a `CommandListener` that is called when the user selects or activates the `Command`:

```
String[] elements = { ... }; //Menu items as List elements
List menuList = new List("Menu", List.IMPLICIT, elements, null);
Command selectCommand = new Command("Open", Command.ITE
M, 1);
menuList.setSelectCommand(selectCommand);
menuList.setCommandListener(...);
```

The listener can query the `List` to determine which element is selected and then perform the corresponding action. Note that setting a command as the select command adds it to the `List` as a side effect.

The select command should be considered as a *default operation* that takes place when a select key is pressed. For example, a `List` displaying email headers might have three operations: read, reply, and delete. Read is considered to be the default operation.

```
List list = new List("Email", List.IMPLICIT, headers);
readCommand = new Command("Read", Command.ITEM, 1);
replyCommand = new Command("Reply", Command.ITEM, 2);
deleteCommand = new Command("Delete", Command.ITEM, 3);
list.setSelectCommand(readCommand);
list.addCommand(replyCommand);
list.addCommand(deleteCommand);
list.setCommandListener(...);
```

On a device with a dedicated select key, pressing this key will invoke `readCommand`. On a device without a select key, the user is still able to invoke the read command, since it is also provided as an ordinary `Command`.

It should be noted that this kind of default operation must be used carefully, and the usability of the resulting user interface must always kept in mind. The default operation should always be the most intuitive operation on a particular List.

Since: MIDP 1.0

Member Summary

Fields

static Command	SELECT_COMMAND[306]

Constructors

	List(String title, int listType)[306]
	List(String title, int listType, String stringElements, Image imageElements)[306]

Methods

int	append(String stringPart, Image imagePart)[307]
void	delete(int elementNum)[307]
void	deleteAll()[308]
int	getFitPolicy()[308]
Font	getFont(int elementNum)[308]
Image	getImage(int elementNum)[309]
int	getSelectedFlags(boolean[] selectedArray_return)[309]
int	getSelectedIndex()[309]
java.lang.String	getString(int elementNum)[310]
void	insert(int elementNum, String stringPart, Image imagePart)[310]
boolean	isSelected(int elementNum)[310]
void	removeCommand(Command cmd)[311]
void	set(int elementNum, String stringPart, Image imagePart)[311]
void	setFitPolicy(int fitPolicy)[312]
void	setFont(int elementNum, Font font)[312]
void	setSelectCommand(Command command)[312]
void	setSelectedFlags(boolean[] selectedArray)[313]
void	setSelectedIndex(int elementNum, boolean selected)[313]
int	size()[314]

Inherited Member Summary

Fields inherited from interface Choice[155]

EXCLUSIVE[158], IMPLICIT[158], MULTIPLE[158], POPUP[158], TEXT_WRAP_DEFAULT[158], TEXT_WRAP_OFF[159], TEXT_WRAP_ON[159]

Methods inherited from class Displayable[218]

addCommand(Command)[219], getHeight()[219], getTicker()[219], getTitle()[220], getWidth()[220], isShown()[220], setCommandListener(CommandListener)[221], setTicker(Ticker)[221], setTitle(String)[221], sizeChanged(int, int)[222]

Methods inherited from class Object

equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Fields

SELECT_COMMAND

Declaration:
`public static final javax.microedition.lcdui.Command`[175] **`SELECT_COMMAND`**

Description:
The default select command for `IMPLICIT` `Lists`. Applications using an `IMPLICIT` `List` should set their own select command using `setSelectCommand`[312].

The field values of `SELECT_COMMAND` are:
- `label = ""` (an empty string)
- `type = SCREEN`
- `priority = 0`

(It would be more appropriate if the type were `ITEM`, but the type of `SCREEN` is retained for historical purposes.)

The application should not use these values for recognizing the `SELECT_COMMAND`. Instead, object identities of the `Command` and `Displayable` (`List`) should be used.

`SELECT_COMMAND` is treated as an ordinary `Command` if it is used with other `Displayable` types.

Constructors

List(String, int)

Declaration:
`public` **`List`**`(String title, int listType)`

Description:
Creates a new, empty `List`, specifying its title and the type of the list.

Parameters:
`title` - the screen's title (see `Displayable`[218])

`listType` - one of `IMPLICIT`, `EXCLUSIVE`, or `MULTIPLE`

Throws:
`IllegalArgumentException` - if `listType` is not one of `IMPLICIT`, `EXCLUSIVE`, or `MULTIPLE`

See Also: `Choice`[155]

List(String, int, String[], Image[])

Declaration:
`public` **`List`**`(String title, int listType, String[] stringElements,`
`javax.microedition.lcdui.Image[]`[270] `imageElements)`

Description:
Creates a new `List`, specifying its title, the type of the `List`, and an array of `Strings` and `Images` to be used as its initial contents.

The `stringElements` array must be non-null and every array element must also be non-null. The length of the `stringElements` array determines the number of elements in the `List`. The `imageElements` array may be `null` to indicate that the `List` elements have no images. If the `imageElements` array is non-null, it must be the same length as the `stringElements` array. Individual elements of the `imageElements` array may be `null` in order to indicate the absence of an image for the corresponding `List` element. Non-null elements of the `imageElements` array may refer to mutable or immutable images.

Parameters:

`title` - the screen's title (see `Displayable`$_{218}$)

`listType` - one of `IMPLICIT`, `EXCLUSIVE`, or `MULTIPLE`

`stringElements` - set of strings specifying the string parts of the `List` elements

`imageElements` - set of images specifying the image parts of the `List` elements

Throws:

`NullPointerException` - if `stringElements` is `null`

`NullPointerException` - if the `stringElements` array contains any null elements

`IllegalArgumentException` - if the `imageElements` array is non-null and has a different length from the `stringElements` array

`IllegalArgumentException` - if `listType` is not one of `IMPLICIT`, `EXCLUSIVE`, or `MULTIPLE`

See Also: `Choice.EXCLUSIVE`$_{158}$, `Choice.MULTIPLE`$_{158}$, `Choice.IMPLICIT`$_{158}$

Methods

append(String, Image)

Declaration:

```
public int append(String stringPart, javax.microedition.lcdui.Image_270 imagePart)
```

Description:

Appends an element to the `List`.

Specified By: `append`$_{159}$ in interface `Choice`$_{155}$

Parameters:

`stringPart` - the string part of the element to be added

`imagePart` - the image part of the element to be added, or `null` if there is no image part

Returns: the assigned index of the element

Throws:

`NullPointerException` - if `stringPart` is `null`

delete(int)

Declaration:

```
public void delete(int elementNum)
```

Description:

Deletes the element referenced by `elementNum`.

Specified By: `delete`[160] in interface `Choice`[155]

Parameters:
`elementNum` - the index of the element to be deleted

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

deleteAll()

Declaration:
```
public void deleteAll()
```

Description:
Deletes all elements from this List.

Specified By: `deleteAll`[160] in interface `Choice`[155]

getFitPolicy()

Declaration:
```
public int getFitPolicy()
```

Description:
Gets the application's preferred policy for fitting `Choice` element contents to the available screen space. The value returned is the policy that had been set by the application, even if that value had been disregarded by the implementation.

Specified By: `getFitPolicy`[160] in interface `Choice`[155]

Returns: one of `Choice.TEXT_WRAP_DEFAULT`[158], `Choice.TEXT_WRAP_ON`[159], or `Choice.TEXT_WRAP_OFF`[159]

Since: MIDP 2.0

See Also: `setFitPolicy(int)`[312]

getFont(int)

Declaration:
```
public javax.microedition.lcdui.Font[223] getFont(int elementNum)
```

Description:
Gets the application's preferred font for rendering the specified element of this `Choice`. The value returned is the font that had been set by the application, even if that value had been disregarded by the implementation. If no font had been set by the application, or if the application explicitly set the font to `null`, the value is the default font chosen by the implementation.

The `elementNum` parameter must be within the range `[0..size()-1]`, inclusive.

Specified By: `getFont`[160] in interface `Choice`[155]

Parameters:
`elementNum` - the index of the element, starting from zero

Returns: the preferred font to use to render the element

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

Since: MIDP 2.0

See Also: `setFont(int, Font)`[312]

getImage(int)

Declaration:
```
public javax.microedition.lcdui.Image[270] getImage(int elementNum)
```

Description:
Gets the `Image` part of the element referenced by `elementNum`.

Specified By: `getImage`[161] in interface `Choice`[155]

Parameters:
`elementNum` - the number of the element to be queried

Returns: the image part of the element, or `null` if there is no image

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

See Also: `getString(int)`[310]

getSelectedFlags(boolean[])

Declaration:
```
public int getSelectedFlags(boolean[] selectedArray_return)
```

Description:
Queries the state of a `List` and returns the state of all elements in the boolean array `selectedArray_return`.

Specified By: `getSelectedFlags`[161] in interface `Choice`[155]

Parameters:
`selectedArray_return` - array to contain the results

Returns: the number of selected elements in the `Choice`

Throws:
`IllegalArgumentException` - if `selectedArray_return` is shorter than the size of the List

`NullPointerException` - if `selectedArray_return` is `null`

See Also: `setSelectedFlags(boolean[])`[313]

getSelectedIndex()

Declaration:
```
public int getSelectedIndex()
```

Description:
Returns the index number of an element in the `List` that is selected.

Specified By: `getSelectedIndex`[162] in interface `Choice`[155]

Returns: index of selected element, or `-1` if none

See Also: `setSelectedIndex(int, boolean)`[313]

getString(int)

Declaration:
```
public String getString(int elementNum)
```

Description:
Gets the `String` part of the element referenced by `elementNum`.

Specified By: `getString`[162] in interface `Choice`[155]

Parameters:
`elementNum` - the index of the element to be queried

Returns: the string part of the element

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

See Also: `getImage(int)`[309]

insert(int, String, Image)

Declaration:
```
public void insert(int elementNum, String stringPart,
            javax.microedition.lcdui.Image[270] imagePart)
```

Description:
Inserts an element into the `List` just prior to the element specified.

Specified By: `insert`[162] in interface `Choice`[155]

Parameters:
`elementNum` - the index of the element where insertion is to occur

`stringPart` - the string part of the element to be inserted

`imagePart` - the image part of the element to be inserted, or `null` if there is no image part

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

`NullPointerException` - if `stringPart` is `null`

isSelected(int)

Declaration:
```
public boolean isSelected(int elementNum)
```

Description:
Gets a boolean value indicating whether this element is selected.

Specified By: `isSelected`[163] in interface `Choice`[155]

Parameters:
`elementNum` - index to element to be queried

Returns: selection state of the element

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

removeCommand(Command)

Declaration:

```
public void removeCommand(javax.microedition.lcdui.Command175 cmd)
```

Description:

The same as `Displayable.removeCommand`[220] but with the following additional semantics.

If the command to be removed happens to be the select command, the `List` is set to have no select command, and the command is removed from the `List`.

The following code:

```
// Command c is the select command on List list
list.removeCommand(c);
```

is equivalent to the following code:

```
// Command c is the select command on List list
list.setSelectCommand(null);
list.removeCommand(c);
```

Overrides: `removeCommand`[220] in class `Displayable`[218]

Parameters:

`cmd` - the command to be removed

Since: MIDP 2.0

set(int, String, Image)

Declaration:

```
public void set(int elementNum, String stringPart,
            javax.microedition.lcdui.Image270 imagePart)
```

Description:

Sets the `String` and `Image` parts of the element referenced by `elementNum`, replacing the previous contents of the element.

Specified By: `set`[163] in interface `Choice`[155]

Parameters:

`elementNum` - the index of the element to be set

`stringPart` - the string part of the new element

`imagePart` - the image part of the element, or `null` if there is no image part

Throws:

`IndexOutOfBoundsException` - if `elementNum` is invalid

`NullPointerException` - if `stringPart` is `null`

setFitPolicy(int)

Declaration:
```
public void setFitPolicy(int fitPolicy)
```

Description:
Sets the application's preferred policy for fitting `Choice` element contents to the available screen space. The set policy applies for all elements of the `Choice` object. Valid values are `Choice.TEXT_WRAP_DEFAULT`[158], `Choice.TEXT_WRAP_ON`[159], and `Choice.TEXT_WRAP_OFF`[159]. Fit policy is a hint, and the implementation may disregard the application's preferred policy.

Specified By: `setFitPolicy`[163] in interface `Choice`[155]

Parameters:
`fitPolicy` - preferred content fit policy for choice elements

Throws:
`IllegalArgumentException` - if `fitPolicy` is invalid

Since: MIDP 2.0

See Also: `getFitPolicy()`[308]

setFont(int, Font)

Declaration:
```
public void setFont(int elementNum, javax.microedition.lcdui.Font[223] font)
```

Description:
Sets the application's preferred font for rendering the specified element of this `Choice`. An element's font is a hint, and the implementation may disregard the application's preferred font.

The `elementNum` parameter must be within the range `[0..size()-1]`, inclusive.

The `font` parameter must be a valid `Font` object or `null`. If the `font` parameter is `null`, the implementation must use its default font to render the element.

Specified By: `setFont`[164] in interface `Choice`[155]

Parameters:
`elementNum` - the index of the element, starting from zero

`font` - the preferred font to use to render the element

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

Since: MIDP 2.0

See Also: `getFont(int)`[308]

setSelectCommand(Command)

Declaration:
```
public void setSelectCommand(javax.microedition.lcdui.Command[175] command)
```

Description:
Sets the `Command` to be used for an `IMPLICIT` `List` selection action. By default, an implicit selection of a List will result in the predefined `List.SELECT_COMMAND` being used. This behavior may be overridden by calling the `List.setSelectCommand()` method with an appropriate parameter value. If

a `null` reference is passed, this indicates that no “select” action is appropriate for the contents of this `List`.

If a reference to a command object is passed, and it is not the special command `List.SELECT_COMMAND`, and it is not currently present on this `List` object, the command object is added to this `List` as if `addCommand(command)` had been called prior to the command being made the select command. This indicates that this command is to be invoked when the user performs the “select” on an element of this `List`.

The select command should have a command type of `ITEM` to indicate that it operates on the currently selected object. It is not an error if the command is of some other type. (`List.SELECT_COMMAND` has a type of `SCREEN` for historical purposes.) For purposes of presentation and placement within its user interface, the implementation is allowed to treat the select command as if it were of type `ITEM`.

If the select command is later removed from the `List` with `removeCommand()`, the `List` is set to have no select command as if `List.setSelectCommand(null)` had been called.

The default behavior can be reestablished explicitly by calling `setSelectCommand()` with an argument of `List.SELECT_COMMAND`.

This method has no effect if the type of the `List` is not `IMPLICIT`.

Parameters:

`command` - the command to be used for an `IMPLICIT` list selection action, or `null` if there is none

Since: MIDP 2.0

setSelectedFlags(boolean[])

Declaration:

```
public void setSelectedFlags(boolean[] selectedArray)
```

Description:

Sets the selected state of all elements of the `List`.

Specified By: `setSelectedFlags`$_{164}$ in interface `Choice`$_{155}$

Parameters:

`selectedArray` - an array in which the method collect the selection status

Throws:

`IllegalArgumentException` - if `selectedArray` is shorter than the size of the `List`

`NullPointerException` - if `selectedArray` is `null`

See Also: `getSelectedFlags(boolean[])`$_{309}$

setSelectedIndex(int, boolean)

Declaration:

```
public void setSelectedIndex(int elementNum, boolean selected)
```

Description:

Sets the selected state of an element.

Specified By: `setSelectedIndex`$_{164}$ in interface `Choice`$_{155}$

Parameters:

`elementNum` - the index of the element, starting from zero

`selected` - the state of the element, where `true` means selected and `false` means not selected

size()

Throws:
`IndexOutOfBoundsException` - if `elementNum` is invalid

See Also: `getSelectedIndex()`[309]

size()

Declaration:
```
public int size()
```

Description:
Gets the number of elements in the `List`.

Specified By: `size`[165] in interface `Choice`[155]

Returns: the number of elements in the `List`

javax.microedition.lcdui
Screen

Declaration

```
public abstract class Screen extends Displayable[218]

Object
  |
  +--javax.microedition.lcdui.Displayable[218]
        |
        +--javax.microedition.lcdui.Screen
```

Direct Known Subclasses: `Alert`[128], `Form`[231], `List`[303], `TextBox`[323]

Description

The common superclass of all high-level user interface classes. The contents displayed and their interaction with the user are defined by subclasses.

Using subclass-defined methods, the application may change the contents of a `Screen` object while it is shown to the user. If this occurs, and the `Screen` object is visible, the display will be updated automatically. That is, the implementation will refresh the display in a timely fashion without waiting for any further action by the application. For example, suppose a `List` object is currently displayed, and every element of the `List` is visible. If the application inserts a new element at the beginning of the `List`, it is displayed immediately, and the other elements will be rearranged appropriately. There is no need for the application to call another method to refresh the display.

It is recommended that applications change the contents of a `Screen` only while it is not visible (that is, while another `Displayable` is current). Changing the contents of a `Screen` while it is visible may result in performance problems on some devices, and it may also be confusing if the `Screen's` contents changes while the user is interacting with it.

In MIDP 2.0 the four `Screen` methods that defined read/write ticker and title properties were moved to `Displayable`, `Screen's` superclass. The semantics of these methods have not changed.

Since: MIDP 1.0

Inherited Member Summary

Methods inherited from class `Displayable`[218]

`addCommand(Command)`[219], `getHeight()`[219], `getTicker()`[219], `getTitle()`[220], `getWidth()`[220], `isShown()`[220], `removeCommand(Command)`[220], `setCommandListener(CommandListener)`[221], `setTicker(Ticker)`[221], `setTitle(String)`[221], `sizeChanged(int, int)`[222]

Methods inherited from class `Object`

`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait()`, `wait()`, `wait()`

javax.microedition.lcdui
Spacer

Declaration

```
public class Spacer extends Item287

Object
  |
  +--javax.microedition.lcdui.Item287
        |
        +--javax.microedition.lcdui.Spacer
```

Description

A blank, non-interactive item that has a settable minimum size. The minimum width is useful for allocating flexible amounts of space between `Items` within the same row of a `Form`. The minimum height is useful for enforcing a particular minimum height of a row. The application can set the minimum width or height to any non-negative value. The implementation may enforce implementation-defined maximum values for the minimum width and height.

The unlocked preferred width of a `Spacer` is the same as its current minimum width. Its unlocked preferred height is the same as its current minimum height.

Since a `Spacer's` primary purpose is to position other items, it is restricted to be non-interactive, and the application is not allowed to add `Commands` to a `Spacer`. Since the presence of a label on an `Item` may affect layout in device-specific ways, the label of a `Spacer` is restricted to always be `null`, and the application is not allowed to change it.

Since: MIDP 2.0

Member Summary	
Constructors	
	`Spacer(int minWidth, int minHeight)`317
Methods	
`void`	`addCommand(Command cmd)`317
`void`	`setDefaultCommand(Command cmd)`318
`void`	`setLabel(String label)`318
`void`	`setMinimumSize(int minWidth, int minHeight)`318

Inherited Member Summary
Fields inherited from class `Item`287

Inherited Member Summary

BUTTON[291], HYPERLINK[292], LAYOUT_2[292], LAYOUT_BOTTOM[292], LAYOUT_CENTER[292], LAYOUT_DEFAULT[292], LAYOUT_EXPAND[293], LAYOUT_LEFT[293], LAYOUT_NEWLINE_AFTER[293], LAYOUT_NEWLINE_BEFORE[293], LAYOUT_RIGHT[293], LAYOUT_SHRINK[294], LAYOUT_TOP[294], LAYOUT_VCENTER[294], LAYOUT_VEXPAND[294], LAYOUT_VSHRINK[294], PLAIN[295]

Methods inherited from class Item[287]

getLabel()[295], getLayout()[295], getMinimumHeight()[296], getMinimumWidth()[296], getPreferredHeight()[296], getPreferredWidth()[296], notifyStateChanged()[297], removeCommand(Command)[297], setItemCommandListener(ItemCommandListener)[298], setLayout(int)[299], setPreferredSize(int, int)[299]

Methods inherited from class Object

equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Constructors

Spacer(int, int)

Declaration:
```
public Spacer(int minWidth, int minHeight)
```

Description:
Creates a new `Spacer` with the given minimum size. The `Spacer's` label is `null`. The minimum size must be zero or greater. If `minWidth` is greater than the implementation-defined maximum width, the maximum width will be used instead. If `minHeight` is greater than the implementation-defined maximum height, the maximum height will be used instead.

Parameters:
`minWidth` - the minimum width in pixels

`minHeight` - the minimum height in pixels

Throws:
`IllegalArgumentException` - if either `minWidth` or `minHeight` is less than zero

Methods

addCommand(Command)

Declaration:
```
public void addCommand(javax.microedition.lcdui.Command[175] cmd)
```

Description:
`Spacers` are restricted from having `Commands`, so this method will always throw `IllegalStateException` whenever it is called.

Overrides: addCommand[295] in class Item[287]

Parameters:
`cmd` - the `Command`

Throws:
`IllegalStateException`[37] - always

setDefaultCommand(Command)

Declaration:
```
public void setDefaultCommand(javax.microedition.lcdui.Command[175] cmd)
```

Description:
Spacers are restricted from having `Commands`, so this method will always throw `IllegalStateException` whenever it is called.

Overrides: `setDefaultCommand`[298] in class `Item`[287]

Parameters:
`cmd` - the `Command`

Throws:
`IllegalStateException`[37] - always

setLabel(String)

Declaration:
```
public void setLabel(String label)
```

Description:
`Spacers` are restricted to having `null` labels, so this method will always throw `IllegalStateException` whenever it is called.

Overrides: `setLabel`[298] in class `Item`[287]

Parameters:
`label` - the label string

Throws:
`IllegalStateException`[37] - always

setMinimumSize(int, int)

Declaration:
```
public void setMinimumSize(int minWidth, int minHeight)
```

Description:
Sets the minimum size for this spacer. The `Form` will not be allowed to make the item smaller than this size. The minimum size must be zero or greater. If `minWidth` is greater than the implementation-defined maximum width, the maximum width will be used instead. If `minHeight` is greater than the implementation-defined maximum height, the maximum height will be used instead.

Parameters:
`minWidth` - the minimum width in pixels

`minHeight` - the minimum height in pixels

Throws:
`IllegalArgumentException` - if either `minWidth` or `minHeight` is less than zero

javax.microedition.lcdui
StringItem

Declaration

```
public class StringItem extends Item287

Object
  |
  +--javax.microedition.lcdui.Item287
        |
        +--javax.microedition.lcdui.StringItem
```

Description

An item that can contain a string. A `StringItem` is display-only; the user cannot edit the contents. Both the label and the textual content of a `StringItem` may be modified by the application. The visual representation of the label may differ from that of the textual contents.

Member Summary

Constructors

	StringItem(String label, String text)320
	StringItem(String label, String text, int appearanceMode)320

Methods

int	getAppearanceMode()321
Font	getFont()321
java.lang.String	getText()321
void	setFont(Font font)322
void	setText(String text)322

Inherited Member Summary

Fields inherited from class Item287

BUTTON291, HYPERLINK292, LAYOUT_2292, LAYOUT_BOTTOM292, LAYOUT_CENTER292, LAYOUT_DEFAULT292, LAYOUT_EXPAND293, LAYOUT_LEFT293, LAYOUT_NEWLINE_AFTER293, LAYOUT_NEWLINE_BEFORE293, LAYOUT_RIGHT293, LAYOUT_SHRINK294, LAYOUT_TOP294, LAYOUT_VCENTER294, LAYOUT_VEXPAND294, LAYOUT_VSHRINK294, PLAIN295

Methods inherited from class Item287

addCommand(Command)295, getLabel()295, getLayout()295, getMinimumHeight()296, getMinimumWidth()296, getPreferredHeight()296, getPreferredWidth()296, notifyStateChanged()297, removeCommand(Command)297, setDefaultCommand(Command)298, setItemCommandListener(ItemCommandListener)298, setLabel(String)298, setLayout(int)299, setPreferredSize(int, int)299

Methods inherited from class Object

Inherited Member Summary
`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Constructors

StringItem(String, String)

Declaration:
`public StringItem(String label, String text)`

Description:
Creates a new `StringItem` object. Calling this constructor is equivalent to calling

```
StringItem(label, text, PLAIN);
```

Parameters:
`label` - the `Item` label

`text` - the text contents

See Also: `StringItem(String, String, int)`[320]

StringItem(String, String, int)

Declaration:
`public StringItem(String label, String text, int appearanceMode)`

Description:
Creates a new `StringItem` object with the given label, textual content, and appearance mode. Either label or text may be present or `null`.

The `appearanceMode` parameter (see Appearance Modes) is a hint to the platform of the application's intended use for this `StringItem`. To provide hyperlink- or button-like behavior, the application should associate a default `Command` with this `StringItem` and add an `ItemCommandListener` to this `StringItem`.

Here is an example showing the use of a `StringItem` as a button:

```
StringItem strItem =
   new StringItem("Default: ", "Set",
                  Item.BUTTON);
strItem.setDefaultCommand(
   new Command("Set", Command.ITEM, 1);
// icl is ItemCommandListener
strItem.setItemCommandListener(icl);
```

Parameters:

`label` - the `StringItem's` label, or `null` if no label

`text` - the `StringItem's` text contents, or `null` if the contents are initially empty

`appearanceMode` - the appearance mode of the `StringItem`, one of `Item.PLAIN`[295], `Item.HYPERLINK`[292], or `Item.BUTTON`[291]

Throws:

`IllegalArgumentException` - if `appearanceMode` invalid

Since: MIDP 2.0

Methods

getAppearanceMode()

Declaration:
```
public int getAppearanceMode()
```

Description:
Returns the appearance mode of the `StringItem`. See Appearance Modes.

Returns: the appearance mode value, one of `Item.PLAIN`[295], `Item.HYPERLINK`[292], or `Item.BUTTON`[291]

Since: MIDP 2.0

getFont()

Declaration:
```
public javax.microedition.lcdui.Font[223] getFont()
```

Description:
Gets the application's preferred font for rendering this `StringItem`. The value returned is the font that had been set by the application, even if that value had been disregarded by the implementation. If no font had been set by the application, or if the application explicitly set the font to `null`, the value is the default font chosen by the implementation.

Returns: the preferred font to use to render this `StringItem`

Since: MIDP 2.0

See Also: `setFont(Font)`[322]

getText()

Declaration:
```
public String getText()
```

Description:
Gets the text contents of the `StringItem`, or `null` if the `StringItem` is empty.

Returns: a string with the content of the item

See Also: `setText(String)`[322]

setFont(Font)

Declaration:
```
public void setFont(javax.microedition.lcdui.Font223 font)
```

Description:
Sets the application's preferred font for rendering this `StringItem`. The font is a hint, and the implementation may disregard the application's preferred font.

The `font` parameter must be a valid `Font` object or `null`. If the `font` parameter is `null`, the implementation must use its default font to render the `StringItem`.

Parameters:
`font` - the preferred font to use to render this `StringItem`

Since: MIDP 2.0

See Also: `getFont()`321

setText(String)

Declaration:
```
public void setText(String text)
```

Description:
Sets the text contents of the `StringItem`. If text is `null`, the `StringItem` is set to be empty.

Parameters:
`text` - the new content

See Also: `getText()`321

javax.microedition.lcdui
TextBox

Declaration

```
public class TextBox extends Screen315

Object
  |
  +--javax.microedition.lcdui.Displayable218
        |
        +--javax.microedition.lcdui.Screen315
              |
              +--javax.microedition.lcdui.TextBox
```

Description

The `TextBox` class is a `Screen` that allows the user to enter and edit text.

A `TextBox` has a maximum size, which is the maximum number of characters that can be stored in the object at any time (its capacity). This limit is enforced when the `TextBox` instance is constructed, when the user is editing text within the `TextBox`, as well as when the application program calls methods on the `TextBox` that modify its contents. The maximum size is the maximum stored capacity and is unrelated to the number of characters that may be displayed at any given time. The number of characters displayed and their arrangement into rows and columns are determined by the device.

The implementation may place a boundary on the maximum size, and the maximum size actually assigned may be smaller than the application had requested. The value actually assigned will be reflected in the value returned by `getMaxSize()`326 . A defensively-written application should compare this value to the maximum size requested and be prepared to handle cases where they differ.

The text contained within a `TextBox` may be more than can be displayed at one time. If this is the case, the implementation will let the user scroll to view and edit any part of the text. This scrolling occurs transparently to the application.

If the constraints are set to `TextField.ANY`334 The text may contain line breaks. The display of the text must break accordingly and the user must be able to enter line break characters.

`TextBox` has the concept of *input constraints* that is identical to `TextField`. The `constraints` parameters of methods within the `TextBox` class use constants defined in the `TextField`330 class. See the description of input constraints in the `TextField` class for the definition of these constants. `TextBox` also has the same notions as `TextField` of the *actual contents* and the *displayed contents*, described in the same section.

`TextBox` also has the concept of *input modes* that is identical to `TextField`. See the description of input modes in the `TextField` class for more details.

Since: MIDP 1.0

Member Summary	
Constructors	
	`TextBox(String title, String text, int maxSize, int constraints)`324

Member Summary

Methods

void	delete(int offset, int length)[325]
int	getCaretPosition()[325]
int	getChars(char[] data)[325]
int	getConstraints()[326]
int	getMaxSize()[326]
java.lang.String	getString()[326]
void	insert(char[] data, int offset, int length, int position)[326]
void	insert(String src, int position)[327]
void	setChars(char[] data, int offset, int length)[328]
void	setConstraints(int constraints)[328]
void	setInitialInputMode(String characterSubset)[328]
int	setMaxSize(int maxSize)[329]
void	setString(String text)[329]
int	size()[329]

Inherited Member Summary

Methods inherited from class `Displayable`[218]

addCommand(Command)[219], getHeight()[219], getTicker()[219], getTitle()[220], getWidth()[220], isShown()[220], removeCommand(Command)[220], setCommandListener(CommandListener)[221], setTicker(Ticker)[221], setTitle(String)[221], sizeChanged(int, int)[222]

Methods inherited from class `Object`

equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Constructors

TextBox(String, String, int, int)

Declaration:
`public TextBox(String title, String text, int maxSize, int constraints)`

Description:
Creates a new `TextBox` object with the given title string, initial contents, maximum size in characters, and constraints. If the text parameter is `null`, the `TextBox` is created empty. The `maxSize` parameter must be greater than zero. An `IllegalArgumentException` is thrown if the length of the initial contents string exceeds `maxSize`. However, the implementation may assign a maximum size smaller than the application had requested. If this occurs, and if the length of the contents exceeds the newly assigned maximum size, the contents are truncated from the end in order to fit, and no exception is thrown.

Parameters:

`title` - the title text to be shown with the display

`text` - the initial contents of the text editing area, `null` may be used to indicate no initial content

`maxSize` - the maximum capacity in characters. The implementation may limit boundary maximum capacity and the actually assigned capacity may me smaller than requested. A defensive application will test the actually given capacity with `getMaxSize()`[326] .

`constraints` - see input constraints

Throws:

`IllegalArgumentException` - if `maxSize` is zero or less

`IllegalArgumentException` - if the `constraints` parameter is invalid

`IllegalArgumentException` - if `text` is illegal for the specified constraints

`IllegalArgumentException` - if the length of the string exceeds the requested maximum capacity

Methods

delete(int, int)

Declaration:
```
public void delete(int offset, int length)
```

Description:
Deletes characters from the `TextBox`.

The `offset` and `length` parameters must specify a valid range of characters within the contents of the `TextBox`. The `offset` parameter must be within the range `[0..(size())]`, inclusive. The `length` parameter must be a non-negative integer such that `(offset + length) <= size()`.

Parameters:

`offset` - the beginning of the region to be deleted

`length` - the number of characters to be deleted

Throws:

`IllegalArgumentException` - if the resulting contents would be illegal for the current input constraints

`StringIndexOutOfBoundsException` - if `offset` and `length` do not specify a valid range within the contents of the `TextBox`

getCaretPosition()

Declaration:
```
public int getCaretPosition()
```

Description:
Gets the current input position. For some UIs this may block and ask the user for the intended caret position, and on other UIs this may simply return the current caret position.

Returns: the current caret position, `0` if at the beginning

getChars(char[])

Declaration:
```
public int getChars(char[] data)
```

Description:
Copies the contents of the `TextBox` into a character array starting at index zero. Array elements beyond the characters copied are left unchanged.

Parameters:
`data` - the character array to receive the value

Returns: the number of characters copied

Throws:
`ArrayIndexOutOfBoundsException` - if the array is too short for the contents

`NullPointerException` - if `data` is `null`

See Also: `setChars(char[], int, int)`[328]

getConstraints()

Declaration:
```
public int getConstraints()
```

Description:
Gets the current input constraints of the `TextBox`.

Returns: the current constraints value (see input constraints)

See Also: `setConstraints(int)`[328]

getMaxSize()

Declaration:
```
public int getMaxSize()
```

Description:
Returns the maximum size (number of characters) that can be stored in this `TextBox`.

Returns: the maximum size in characters

See Also: `setMaxSize(int)`[329]

getString()

Declaration:
```
public String getString()
```

Description:
Gets the contents of the `TextBox` as a string value.

Returns: the current contents

See Also: `setString(String)`[329]

insert(char[], int, int, int)

Declaration:
```
public void insert(char[] data, int offset, int length, int position)
```

Description:
Inserts a subrange of an array of characters into the contents of the `TextBox`. The `offset` and `length` parameters indicate the subrange of the data array to be used for insertion. Behavior is otherwise identical to `insert(String, int)`[327].

The `offset` and `length` parameters must specify a valid range of characters within the character array `data`. The `offset` parameter must be within the range `[0..(data.length)]`, inclusive. The `length` parameter must be a non-negative integer such that `(offset + length) <= data.length`.

Parameters:

`data` - the source of the character data

`offset` - the beginning of the region of characters to copy

`length` - the number of characters to copy

`position` - the position at which insertion is to occur

Throws:

`ArrayIndexOutOfBoundsException` - if `offset` and `length` do not specify a valid range within the data array

`IllegalArgumentException` - if the resulting contents would be illegal for the current input constraints

`IllegalArgumentException` - if the insertion would exceed the current maximum capacity

`NullPointerException` - if `data` is `null`

insert(String, int)

Declaration:

```
public void insert(String src, int position)
```

Description:

Inserts a string into the contents of the `TextBox`. The string is inserted just prior to the character indicated by the `position` parameter, where zero specifies the first character of the contents of the `TextBox`. If `position` is less than or equal to zero, the insertion occurs at the beginning of the contents, thus effecting a prepend operation. If `position` is greater than or equal to the current size of the contents, the insertion occurs immediately after the end of the contents, thus effecting an append operation. For example, `text.insert(s, text.size())` always appends the string `s` to the current contents.

The current size of the contents is increased by the number of inserted characters. The resulting string must fit within the current maximum capacity.

If the application needs to simulate typing of characters it can determining the location of the current insertion point (“caret”) using the with `getCaretPosition()` 325 method. For example, `text.insert(s, text.getCaretPosition())` inserts the string `s` at the current caret position.

Parameters:

`src` - the `String` to be inserted

`position` - the position at which insertion is to occur

Throws:

`IllegalArgumentException` - if the resulting contents would be illegal for the current input constraints

`IllegalArgumentException` - if the insertion would exceed the current maximum capacity

`NullPointerException` - if `src` is `null`

setChars(char[], int, int)

Declaration:
```
public void setChars(char[] data, int offset, int length)
```

Description:
Sets the contents of the `TextBox` from a character array, replacing the previous contents. Characters are copied from the region of the `data` array starting at array index `offset` and running for `length` characters. If the data array is `null`, the `TextBox` is set to be empty and the other parameters are ignored.

The `offset` and `length` parameters must specify a valid range of characters within the character array `data`. The `offset` parameter must be within the range `[0..(data.length)]`, inclusive. The `length` parameter must be a non-negative integer such that `(offset + length) <= data.length`.

Parameters:
`data` - the source of the character data

`offset` - the beginning of the region of characters to copy

`length` - the number of characters to copy

Throws:
`ArrayIndexOutOfBoundsException` - if `offset` and `length` do not specify a valid range within the data array

`IllegalArgumentException` - if `data` is illegal for the current input constraints

`IllegalArgumentException` - if the text would exceed the current maximum capacity

See Also: `getChars(char[])` *325*

setConstraints(int)

Declaration:
```
public void setConstraints(int constraints)
```

Description:
Sets the input constraints of the `TextBox`. If the current contents of the `TextBox` do not match the new constraints, the contents are set to empty.

Parameters:
`constraints` - see input constraints

Throws:
`IllegalArgumentException` - if the value of the constraints parameter is invalid

See Also: `getConstraints()` *326*

setInitialInputMode(String)

Declaration:
```
public void setInitialInputMode(String characterSubset)
```

Description:
Sets a hint to the implementation as to the input mode that should be used when the user initiates editing of this `TextBox`. The `characterSubset` parameter names a subset of Unicode characters that is used by the implementation to choose an initial input mode. If `null` is passed, the implementation should choose a default input mode.

See Input Modes for a full explanation of input modes.

Parameters:
`characterSubset` - a string naming a Unicode character subset, or `null`

Since: MIDP 2.0

setMaxSize(int)

Declaration:
```
public int setMaxSize(int maxSize)
```

Description:
Sets the maximum size (number of characters) that can be contained in this `TextBox`. If the current contents of the `TextBox` are larger than `maxSize`, the contents are truncated to fit.

Parameters:
`maxSize` - the new maximum size

Returns: assigned maximum capacity - may be smaller than requested.

Throws:
`IllegalArgumentException` - if `maxSize` is zero or less.

`IllegalArgumentException` - if the contents after truncation would be illegal for the current input constraints

See Also: `getMaxSize()` *326*

setString(String)

Declaration:
```
public void setString(String text)
```

Description:
Sets the contents of the `TextBox` as a string value, replacing the previous contents.

Parameters:
`text` - the new value of the `TextBox`, or `null` if the `TextBox` is to be made empty

Throws:
`IllegalArgumentException` - if `text` is illegal for the current input constraints

`IllegalArgumentException` - if the text would exceed the current maximum capacity

See Also: `getString()` *326*

size()

Declaration:
```
public int size()
```

Description:
Gets the number of characters that are currently stored in this `TextBox`.

Returns: the number of characters

javax.microedition.lcdui
TextField

Declaration

```
public class TextField extends Item
```

```
Object
  |
  +--javax.microedition.lcdui.Item
        |
        +--javax.microedition.lcdui.TextField
```

Description

A `TextField` is an editable text component that may be placed into a `Form` . It can be given a piece of text that is used as the initial value.

A `TextField` has a maximum size, which is the maximum number of characters that can be stored in the object at any time (its capacity). This limit is enforced when the `TextField` instance is constructed, when the user is editing text within the `TextField`, as well as when the application program calls methods on the `TextField` that modify its contents. The maximum size is the maximum stored capacity and is unrelated to the number of characters that may be displayed at any given time. The number of characters displayed and their arrangement into rows and columns are determined by the device.

The implementation may place a boundary on the maximum size, and the maximum size actually assigned may be smaller than the application had requested. The value actually assigned will be reflected in the value returned by `getMaxSize()` . A defensively-written application should compare this value to the maximum size requested and be prepared to handle cases where they differ.

Input Constraints

The `TextField` shares the concept of *input constraints* with the `TextBox` class. The different constraints allow the application to request that the user's input be restricted in a variety of ways. The implementation is required to restrict the user's input as requested by the application. For example, if the application requests the `NUMERIC` constraint on a `TextField`, the implementation must allow only numeric characters to be entered.

The *actual contents* of the text object are set and modified by and are reported to the application through the `TextBox` and `TextField` APIs. The *displayed contents* may differ from the actual contents if the implementation has chosen to provide special formatting suitable for the text object's constraint setting. For example, a `PHONENUMBER` field might be displayed with digit separators and punctuation as appropriate for the phone number conventions in use, grouping the digits into country code, area code, prefix, etc. Any spaces or punctuation provided are not considered part of the text object's actual contents. For example, a text object with the `PHONENUMBER` constraint might display as follows:

(408) 555-1212

but the actual contents of the object visible to the application through the APIs would be the string "`4085551212`". The `size` method reflects the number of characters in the actual contents, not the number of characters that are displayed, so for this example the `size` method would return `10`.

Some constraints, such as `DECIMAL`, require the implementation to perform syntactic validation of the contents of the text object. The syntax checking is performed on the actual contents of the text object, which may differ from the displayed contents as described above. Syntax checking is performed on the initial contents passed to the constructors, and it is also enforced for all method calls that affect the contents of the text object. The methods and constructors throw `IllegalArgumentException` if they would result in the contents of the text object not conforming to the required syntax.

The value passed to the `setConstraints()`$_{342}$ method consists of a restrictive constraint setting described above, as well as a variety of flag bits that modify the behavior of text entry and display. The value of the restrictive constraint setting is in the low order `16` bits of the value, and it may be extracted by combining the constraint value with the `CONSTRAINT_MASK` constant using the bit-wise `AND` (`&`) operator. The restrictive constraint settings are as follows:

```
ANY
 EMAILADDR
 NUMERIC
 PHONENUMBER
 URL
 DECIMAL
```

The modifier flags reside in the high order `16` bits of the constraint value, that is, those in the complement of the `CONSTRAINT_MASK` constant. The modifier flags may be tested individually by combining the constraint value with a modifier flag using the bit-wise `AND` (`&`) operator. The modifier flags are as follows:

```
PASSWORD
 UNEDITABLE
 SENSITIVE
 NON_PREDICTIVE
 INITIAL_CAPS_WORD
 INITIAL_CAPS_SENTENCE
```

Input Modes

The `TextField` shares the concept of *input modes* with the `TextBox`$_{323}$ class. The application can request that the implementation use a particular input mode when the user initiates editing of a `TextField` or `TextBox`. The input mode is a concept that exists within the user interface for text entry on a particular device. The application does not request an input mode directly, since the user interface for text entry is not standardized across devices. Instead, the application can request that the entry of certain characters be made convenient. It can do this by passing the name of a Unicode character subset to the `setInitialInputMode()`$_{343}$ method. Calling this method requests that the implementation set the mode of the text entry user interface so that it is convenient for the user to enter characters in this subset. The application can also request that the input mode have certain behavioral characteristics by setting modifier flags in the constraints value.

The requested input mode should be used whenever the user initiates the editing of a `TextBox` or `TextField` object. If the user had changed input modes in a previous editing session, the application's requested input mode should take precedence over the previous input mode set by the user. However, the input mode is not restrictive, and the user is allowed to change the input mode at any time during editing. If editing is already in progress, calls to the `setInitialInputMode` method do not affect the current input mode, but instead take effect at the next time the user initiates editing of this text object.

The initial input mode is a hint to the implementation. If the implementation cannot provide an input mode that satisfies the application's request, it should use a default input mode.

The input mode that results from the application's request is not a restriction on the set of characters the user is allowed to enter. The user MUST be allowed to switch input modes to enter any character that is allowed within the current constraint setting. The constraint setting takes precedence over an input mode request, and the implementation may refuse to supply a particular input mode if it is inconsistent with the current constraint setting.

For example, if the current constraint is ANY, the call

```
setInitialInputMode("MIDP_UPPERCASE_LATIN");
```

should set the initial input mode to allow entry of uppercase Latin characters. This does not restrict input to these characters, and the user will be able to enter other characters by switching the input mode to allow entry of numerals or lowercase Latin letters. However, if the current constraint is `NUMERIC`, the implementation may ignore the request to set an initial input mode allowing `MIDP_UPPERCASE_LATIN` characters because these characters are not allowed in a `TextField` whose constraint is `NUMERIC`. In this case, the implementation may instead use an input mode that allows entry of numerals, since such an input mode is most appropriate for entry of data under the `NUMERIC` constraint.

A string is used to name the Unicode character subset passed as a parameter to the `setInitialInputMode()`[343] method. String comparison is case sensitive.

Unicode character blocks can be named by adding the prefix "`UCB_`" to the the string names of fields representing Unicode character blocks as defined in the J2SE class `java.lang.Character.UnicodeBlock`. Any Unicode character block may be named in this fashion. For convenience, the most common Unicode character blocks are listed below.

```
UCB_BASIC_LATIN
 UCB_GREEK
 UCB_CYRILLIC
 UCB_ARMENIAN
 UCB_HEBREW
 UCB_ARABIC
 UCB_DEVANAGARI
 UCB_BENGALI
 UCB_THAI
 UCB_HIRAGANA
 UCB_KATAKANA
 UCB_HANGUL_SYLLABLES
```

"Input subsets" as defined by the J2SE class `java.awt.im.InputSubset` may be named by adding the prefix "`IS_`" to the string names of fields representing input subsets as defined in that class. Any defined input subset may be used. For convenience, the names of the currently defined input subsets are listed below.

```
IS_FULLWIDTH_DIGITS
 IS_FULLWIDTH_LATIN
 IS_HALFWIDTH_KATAKANA
 IS_HANJA
 IS_KANJI
 IS_LATIN
 IS_LATIN_DIGITS
 IS_SIMPLIFIED_HANZI
```

`IS_TRADITIONAL_HANZI`

MIDP has also defined the following character subsets:

`MIDP_UPPERCASE_LATIN` - the subset of `IS_LATIN` that corresponds to uppercase Latin letters

`MIDP_LOWERCASE_LATIN` - the subset of `IS_LATIN` that corresponds to lowercase Latin letters

Finally, implementation-specific character subsets may be named with strings that have a prefix of "`X_`". In order to avoid namespace conflicts, it is recommended that implementation-specific names include the name of the defining company or organization after the initial "`X_`" prefix.

For example, a Japanese language application might have a particular `TextField` that the application intends to be used primarily for input of words that are "loaned" from languages other than Japanese. The application might request an input mode facilitating Hiragana input by issuing the following method call:

```
textfield.setInitialInputMode("UCB_HIRAGANA");
```

Implementation Note
Implementations need not compile in all the strings listed above. Instead, they need only to compile in the strings that name Unicode character subsets that they support. If the subset name passed by the application does not match a known subset name, the request should simply be ignored without error, and a default input mode should be used. This lets implementations support this feature reasonably inexpensively. However, it has the consequence that the application cannot tell whether its request has been accepted, nor whether the Unicode character subset it has requested is actually a valid subset.

Since: MIDP 1.0

Member Summary

Fields

static int	ANY[334]
static int	CONSTRAINT_MASK[335]
static int	DECIMAL[335]
static int	EMAILADDR[335]
static int	INITIAL_CAPS_SENTENCE[336]
static int	INITIAL_CAPS_WORD[336]
static int	NON_PREDICTIVE[336]
static int	NUMERIC[337]
static int	PASSWORD[337]
static int	PHONENUMBER[337]
static int	SENSITIVE[338]
static int	UNEDITABLE[338]
static int	URL[338]

Constructors

	TextField(String label, String text, int maxSize, int constraints)[338]

Methods

void	delete(int offset, int length)[339]

Member Summary	
int	getCaretPosition()[339]
int	getChars(char[] data)[340]
int	getConstraints()[340]
int	getMaxSize()[340]
java.lang.String	getString()[340]
void	insert(char[] data, int offset, int length, int position)[341]
void	insert(String src, int position)[341]
void	setChars(char[] data, int offset, int length)[342]
void	setConstraints(int constraints)[342]
void	setInitialInputMode(String characterSubset)[343]
int	setMaxSize(int maxSize)[343]
void	setString(String text)[343]
int	size()[343]

Inherited Member Summary

Fields inherited from class `Item`[287]

BUTTON[291], HYPERLINK[292], LAYOUT_2[292], LAYOUT_BOTTOM[292], LAYOUT_CENTER[292], LAYOUT_DEFAULT[292], LAYOUT_EXPAND[293], LAYOUT_LEFT[293], LAYOUT_NEWLINE_AFTER[293], LAYOUT_NEWLINE_BEFORE[293], LAYOUT_RIGHT[293], LAYOUT_SHRINK[294], LAYOUT_TOP[294], LAYOUT_VCENTER[294], LAYOUT_VEXPAND[294], LAYOUT_VSHRINK[294], PLAIN[295]

Methods inherited from class `Item`[287]

addCommand(Command)[295], getLabel()[295], getLayout()[295], getMinimumHeight()[296], getMinimumWidth()[296], getPreferredHeight()[296], getPreferredWidth()[296], notifyStateChanged()[297], removeCommand(Command)[297], setDefaultCommand(Command)[298], setItemCommandListener(ItemCommandListener)[298], setLabel(String)[298], setLayout(int)[299], setPreferredSize(int, int)[299]

Methods inherited from class `Object`

equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Fields

ANY

Declaration:
```
public static final int ANY
```

Description:
The user is allowed to enter any text. Line breaks may be entered.

Constant `0` is assigned to ANY.

CONSTRAINT_MASK

Declaration:
```
public static final int CONSTRAINT_MASK
```

Description:
The mask value for determining the constraint mode. The application should use the bit-wise `AND` operation with a value returned by `getConstraints()` and `CONSTRAINT_MASK` in order to retrieve the current constraint mode, in order to remove any modifier flags such as the `PASSWORD` flag.

Constant `0xFFFF` is assigned to `CONSTRAINT_MASK`.

DECIMAL

Declaration:
```
public static final int DECIMAL
```

Description:
The user is allowed to enter numeric values with optional decimal fractions, for example “-123”, “0.123”, or “.5”.

The implementation may display a period “.” or a comma “,” for the decimal fraction separator, depending on the conventions in use on the device. Similarly, the implementation may display other device-specific characters as part of a decimal string, such as spaces or commas for digit separators. However, the only characters allowed in the actual contents of the text object are period “.”, minus sign “-”, and the decimal digits.

The actual contents of a `DECIMAL` text object may be empty. If the actual contents are not empty, they must conform to a subset of the syntax for a `FloatingPointLiteral` as defined by the *Java Language Specification*, section 3.10.2. This subset syntax is defined as follows: the actual contents must consist of an optional minus sign “-”, followed by one or more whole-number decimal digits, followed by an optional fraction separator, followed by zero or more decimal fraction digits. The whole-number decimal digits may be omitted if the fraction separator and one or more decimal fraction digits are present.

The syntax defined above is also enforced whenever the application attempts to set or modify the contents of the text object by calling a constructor or a method.

Parsing this string value into a numeric value suitable for computation is the responsibility of the application. If the contents are not empty, the result can be parsed successfully by `Double.valueOf` and related methods if they are present in the runtime environment.

The sign and the fraction separator consume space in the text object. Applications should account for this when assigning a maximum size for the text object.

Constant `5` is assigned to `DECIMAL`.

Since: MIDP 2.0

EMAILADDR

Declaration:
```
public static final int EMAILADDR
```

Description:
The user is allowed to enter an e-mail address.

Constant `1` is assigned to `EMAILADDR`.

INITIAL_CAPS_SENTENCE

Declaration:
```
public static final int INITIAL_CAPS_SENTENCE
```

Description:
This flag is a hint to the implementation that during text editing, the initial letter of each sentence should be capitalized. This hint should be honored only on devices for which automatic capitalization is appropriate and when the character set of the text being edited has the notion of upper case and lower case letters. The definition of sentence boundaries is implementation-specific.

If the application specifies both the `INITIAL_CAPS_WORD` and the `INITIAL_CAPS_SENTENCE` flags, `INITIAL_CAPS_WORD` behavior should be used.

The `INITIAL_CAPS_SENTENCE` modifier can be combined with other input constraints by using the bit-wise `OR` operator (`|`).

Constant `0x200000` is assigned to `INITIAL_CAPS_SENTENCE`.

Since: MIDP 2.0

INITIAL_CAPS_WORD

Declaration:
```
public static final int INITIAL_CAPS_WORD
```

Description:
This flag is a hint to the implementation that during text editing, the initial letter of each word should be capitalized. This hint should be honored only on devices for which automatic capitalization is appropriate and when the character set of the text being edited has the notion of upper case and lower case letters. The definition of word boundaries is implementation-specific.

If the application specifies both the `INITIAL_CAPS_WORD` and the `INITIAL_CAPS_SENTENCE` flags, `INITIAL_CAPS_WORD` behavior should be used.

The `INITIAL_CAPS_WORD` modifier can be combined with other input constraints by using the bit-wise `OR` operator (`|`).

Constant `0x100000` is assigned to `INITIAL_CAPS_WORD`.

Since: MIDP 2.0

NON_PREDICTIVE

Declaration:
```
public static final int NON_PREDICTIVE
```

Description:
Indicates that the text entered does not consist of words that are likely to be found in dictionaries typically used by predictive input schemes. If this bit is clear, the implementation is allowed to (but is not required to) use predictive input facilities. If this bit is set, the implementation should not use any predictive input facilities, but it instead should allow character-by-character text entry.

The `NON_PREDICTIVE` modifier can be combined with other input constraints by using the bit-wise `OR` operator (`|`).

Constant `0x80000` is assigned to `NON_PREDICTIVE`.

Since: MIDP 2.0

NUMERIC

Declaration:
```
public static final int NUMERIC
```

Description:
The user is allowed to enter only an integer value. The implementation must restrict the contents either to be empty or to consist of an optional minus sign followed by a string of one or more decimal numerals. Unless the value is empty, it will be successfully parsable using `Integer.parseInt(String)`.

The minus sign consumes space in the text object. It is thus impossible to enter negative numbers into a text object whose maximum size is `1`.

Constant `2` is assigned to `NUMERIC`.

PASSWORD

Declaration:
```
public static final int PASSWORD
```

Description:
Indicates that the text entered is confidential data that should be obscured whenever possible. The contents may be visible while the user is entering data. However, the contents must never be divulged to the user. In particular, the existing contents must not be shown when the user edits the contents. The means by which the contents are obscured is implementation-dependent. For example, each character of the data might be masked with a "`*`" character. The `PASSWORD` modifier is useful for entering confidential information such as passwords or personal identification numbers (PINs).

Data entered into a `PASSWORD` field is treated similarly to `SENSITIVE` in that the implementation must never store the contents into a dictionary or table for use in predictive, auto-completing, or other accelerated input schemes. If the `PASSWORD` bit is set in a constraint value, the `SENSITIVE` and `NON_PREDICTIVE` bits are also considered to be set, regardless of their actual values. In addition, the `INITIAL_CAPS_WORD` and `INITIAL_CAPS_SENTENCE` flag bits should be ignored even if they are set.

The `PASSWORD` modifier can be combined with other input constraints by using the bit-wise `OR` operator (`|`). The `PASSWORD` modifier is not useful with some constraint values such as `EMAILADDR`, `PHONENUMBER`, and `URL`. These combinations are legal, however, and no exception is thrown if such a constraint is specified.

Constant `0x10000` is assigned to `PASSWORD`.

PHONENUMBER

Declaration:
```
public static final int PHONENUMBER
```

Description:
The user is allowed to enter a phone number. The phone number is a special case, since a phone-based implementation may be linked to the native phone dialing application. The implementation may automatically start a phone dialer application that is initialized so that pressing a single key would be enough to make a call. The call must not made automatically without requiring user's confirmation. Implementations may also provide a feature to look up the phone number in the device's phone or address database.

The exact set of characters allowed is specific to the device and to the device's network and may include non-numeric characters, such as a "+" prefix character.

Some platforms may provide the capability to initiate voice calls using the `MIDlet.platformRequest`[447] method.

Constant `3` is assigned to `PHONENUMBER`.

SENSITIVE

Declaration:
```
public static final int SENSITIVE
```

Description:
Indicates that the text entered is sensitive data that the implementation must never store into a dictionary or table for use in predictive, auto-completing, or other accelerated input schemes. A credit card number is an example of sensitive data.

The `SENSITIVE` modifier can be combined with other input constraints by using the bit-wise `OR` operator (`|`).

Constant `0x40000` is assigned to `SENSITIVE`.

Since: MIDP 2.0

UNEDITABLE

Declaration:
```
public static final int UNEDITABLE
```

Description:
Indicates that editing is currently disallowed. When this flag is set, the implementation must prevent the user from changing the text contents of this object. The implementation should also provide a visual indication that the object's text cannot be edited. The intent of this flag is that this text object has the potential to be edited, and that there are circumstances where the application will clear this flag and allow the user to edit the contents.

The `UNEDITABLE` modifier can be combined with other input constraints by using the bit-wise `OR` operator (`|`).

Constant `0x20000` is assigned to `UNEDITABLE`.

Since: MIDP 2.0

URL

Declaration:
```
public static final int URL
```

Description:
The user is allowed to enter a URL.

Constant `4` is assigned to `URL`.

Constructors

TextField(String, String, int, int)

Declaration:
```
public TextField(String label, String text, int maxSize, int constraints)
```

Description:
Creates a new `TextField` object with the given label, initial contents, maximum size in characters, and constraints. If the text parameter is `null`, the `TextField` is created empty. The `maxSize` parameter must be greater than zero. An `IllegalArgumentException` is thrown if the length of the initial contents string exceeds `maxSize`. However, the implementation may assign a maximum size smaller than the application had requested. If this occurs, and if the length of the contents exceeds the newly assigned maximum size, the contents are truncated from the end in order to fit, and no exception is thrown.

Parameters:

`label` - item label

`text` - the initial contents, or `null` if the `TextField` is to be empty

`maxSize` - the maximum capacity in characters

`constraints` - see input constraints

Throws:

`IllegalArgumentException` - if `maxSize` is zero or less

`IllegalArgumentException` - if the value of the constraints parameter is invalid

`IllegalArgumentException` - if `text` is illegal for the specified constraints

`IllegalArgumentException` - if the length of the string exceeds the requested maximum capacity

Methods

delete(int, int)

Declaration:
```
public void delete(int offset, int length)
```

Description:
Deletes characters from the `TextField`.

The `offset` and `length` parameters must specify a valid range of characters within the contents of the `TextField`. The `offset` parameter must be within the range `[0..(size())]`, inclusive. The `length` parameter must be a non-negative integer such that `(offset + length) <= size()`.

Parameters:

`offset` - the beginning of the region to be deleted

`length` - the number of characters to be deleted

Throws:

`IllegalArgumentException` - if the resulting contents would be illegal for the current input constraints

`StringIndexOutOfBoundsException` - if `offset` and `length` do not specify a valid range within the contents of the `TextField`

getCaretPosition()

Declaration:
```
public int getCaretPosition()
```

Description:
Gets the current input position. For some UIs this may block and ask the user for the intended caret position, and on other UIs this may simply return the current caret position.

Returns: the current caret position, `0` if at the beginning

getChars(char[])

Declaration:
```
public int getChars(char[] data)
```

Description:
Copies the contents of the `TextField` into a character array starting at index zero. Array elements beyond the characters copied are left unchanged.

Parameters:
`data` - the character array to receive the value

Returns: the number of characters copied

Throws:
`ArrayIndexOutOfBoundsException` - if the array is too short for the contents

`NullPointerException` - if `data` is `null`

See Also: `setChars(char[], int, int)`342

getConstraints()

Declaration:
```
public int getConstraints()
```

Description:
Gets the current input constraints of the `TextField`.

Returns: the current constraints value (see input constraints)

See Also: `setConstraints(int)`342

getMaxSize()

Declaration:
```
public int getMaxSize()
```

Description:
Returns the maximum size (number of characters) that can be stored in this `TextField`.

Returns: the maximum size in characters

See Also: `setMaxSize(int)`343

getString()

Declaration:
```
public String getString()
```

Description:
Gets the contents of the `TextField` as a string value.

Returns: the current contents

See Also: `setString(String)`343

insert(char[], int, int, int)

Declaration:
```
public void insert(char[] data, int offset, int length, int position)
```

Description:
Inserts a subrange of an array of characters into the contents of the `TextField`. The `offset` and `length` parameters indicate the subrange of the data array to be used for insertion. Behavior is otherwise identical to `insert(String, int)`[341] .

The `offset` and `length` parameters must specify a valid range of characters within the character array `data`. The `offset` parameter must be within the range `[0..(data.length)]`, inclusive. The `length` parameter must be a non-negative integer such that `(offset + length) <= data.length`.

Parameters:

`data` - the source of the character data

`offset` - the beginning of the region of characters to copy

`length` - the number of characters to copy

`position` - the position at which insertion is to occur

Throws:

`ArrayIndexOutOfBoundsException` - if `offset` and `length` do not specify a valid range within the `data` array

`IllegalArgumentException` - if the resulting contents would be illegal for the current input constraints

`IllegalArgumentException` - if the insertion would exceed the current maximum capacity

`NullPointerException` - if `data` is `null`

insert(String, int)

Declaration:
```
public void insert(String src, int position)
```

Description:
Inserts a string into the contents of the `TextField`. The string is inserted just prior to the character indicated by the `position` parameter, where zero specifies the first character of the contents of the `TextField`. If `position` is less than or equal to zero, the insertion occurs at the beginning of the contents, thus effecting a prepend operation. If `position` is greater than or equal to the current size of the contents, the insertion occurs immediately after the end of the contents, thus effecting an append operation. For example, `text.insert(s, text.size())` always appends the string `s` to the current contents.

The current size of the contents is increased by the number of inserted characters. The resulting string must fit within the current maximum capacity.

If the application needs to simulate typing of characters it can determining the location of the current insertion point (“caret”) using the with `getCaretPosition()`[339] method. For example, `text.insert(s, text.getCaretPosition())` inserts the string `s` at the current caret position.

Parameters:

`src` - the `String` to be inserted

`position` - the position at which insertion is to occur

Throws:

`IllegalArgumentException` - if the resulting contents would be illegal for the current input constraints

`IllegalArgumentException` - if the insertion would exceed the current maximum capacity

`NullPointerException` - if `src` is `null`

setChars(char[], int, int)

Declaration:
```
public void setChars(char[] data, int offset, int length)
```

Description:
Sets the contents of the `TextField` from a character array, replacing the previous contents. Characters are copied from the region of the `data` array starting at array index `offset` and running for `length` characters. If the data array is `null`, the `TextField` is set to be empty and the other parameters are ignored.

The `offset` and `length` parameters must specify a valid range of characters within the character array `data`. The `offset` parameter must be within the range `[0..(data.length)]`, inclusive. The `length` parameter must be a non-negative integer such that `(offset + length) <= data.length`.

Parameters:

`data` - the source of the character data

`offset` - the beginning of the region of characters to copy

`length` - the number of characters to copy

Throws:

`ArrayIndexOutOfBoundsException` - if `offset` and `length` do not specify a valid range within the data array

`IllegalArgumentException` - if `data` is illegal for the current input constraints

`IllegalArgumentException` - if the text would exceed the current maximum capacity

See Also: `getChars(char[])` *340*

setConstraints(int)

Declaration:
```
public void setConstraints(int constraints)
```

Description:
Sets the input constraints of the `TextField`. If the the current contents of the `TextField` do not match the new `constraints`, the contents are set to empty.

Parameters:

`constraints` - see input constraints

Throws:

`IllegalArgumentException` - if constraints is not any of the ones specified in input constraints

See Also: `getConstraints()` *340*

setInitialInputMode(String)

Declaration:
```
public void setInitialInputMode(String characterSubset)
```

Description:
Sets a hint to the implementation as to the input mode that should be used when the user initiates editing of this `TextField`. The `characterSubset` parameter names a subset of Unicode characters that is used by the implementation to choose an initial input mode. If `null` is passed, the implementation should choose a default input mode.

See Input Modes for a full explanation of input modes.

Parameters:
`characterSubset` - a string naming a Unicode character subset, or `null`

Since: MIDP 2.0

setMaxSize(int)

Declaration:
```
public int setMaxSize(int maxSize)
```

Description:
Sets the maximum size (number of characters) that can be contained in this `TextField`. If the current contents of the `TextField` are larger than `maxSize`, the contents are truncated to fit.

Parameters:
`maxSize` - the new maximum size

Returns: assigned maximum capacity - may be smaller than requested.

Throws:
`IllegalArgumentException` - if `maxSize` is zero or less.

`IllegalArgumentException` - if the contents after truncation would be illegal for the current input constraints

See Also: `getMaxSize()` *340*

setString(String)

Declaration:
```
public void setString(String text)
```

Description:
Sets the contents of the `TextField` as a string value, replacing the previous contents.

Parameters:
`text` - the new value of the `TextField`, or `null` if the TextField is to be made empty

Throws:
`IllegalArgumentException` - if `text` is illegal for the current input constraints

`IllegalArgumentException` - if the text would exceed the current maximum capacity

See Also: `getString()` *340*

size()

Declaration:
```
public int size()
```

size()

Description:
Gets the number of characters that are currently stored in this `TextField`.

Returns: number of characters in the `TextField`

javax.microedition.lcdui
Ticker

Declaration

```
public class Ticker

Object
  |
  +--javax.microedition.lcdui.Ticker
```

Description

Implements a “ticker-tape”, a piece of text that runs continuously across the display. The direction and speed of scrolling are determined by the implementation. While animating, the ticker string scrolls continuously. That is, when the string finishes scrolling off the display, the ticker starts over at the beginning of the string.

There is no API provided for starting and stopping the ticker. The application model is that the ticker is always scrolling continuously. However, the implementation is allowed to pause the scrolling for power consumption purposes, for example, if the user doesn’t interact with the device for a certain period of time. The implementation should resume scrolling the ticker when the user interacts with the device again.

The text of the ticker may contain line breaks. The complete text MUST be displayed in the ticker; line break characters should not be displayed but may be used as separators.

The same ticker may be shared by several `Displayable` objects (“screens”). This can be accomplished by calling `setTicker()`*221* on each of them. Typical usage is for an application to place the same ticker on all of its screens. When the application switches between two screens that have the same ticker, a desirable effect is for the ticker to be displayed at the same location on the display and to continue scrolling its contents at the same position. This gives the illusion of the ticker being attached to the display instead of to each screen.

An alternative usage model is for the application to use different tickers on different sets of screens or even a different one on each screen. The ticker is an attribute of the `Displayable` class so that applications may implement this model without having to update the ticker to be displayed as the user switches among screens.

Since: MIDP 1.0

Member Summary	
Constructors	
	`Ticker(String str)`*346*
Methods	
`java.lang.String`	`getString()`*346*
`void`	`setString(String str)`*346*

Inherited Member Summary
Methods inherited from class `Object`

Inherited Member Summary
`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Constructors

Ticker(String)

Declaration:
`public Ticker(String str)`

Description:
Constructs a new `Ticker` object, given its initial contents string.

Parameters:
`str` - string to be set for the `Ticker`

Throws:
`NullPointerException` - if `str` is `null`

Methods

getString()

Declaration:
`public String getString()`

Description:
Gets the string currently being scrolled by the ticker.

Returns: string of the ticker

See Also: `setString(String)` *346*

setString(String)

Declaration:
`public void setString(String str)`

Description:
Sets the string to be displayed by this ticker. If this ticker is active and is on the display, it immediately begins showing the new string.

Parameters:
`str` - string to be set for the `Ticker`

Throws:
`NullPointerException` - if `str` is `null`

See Also: `getString()` *346*

CHAPTER 9

Package javax.microedition.lcdui.game

Description

The Game API package provides a series of classes that enable the development of rich gaming content for wireless devices.

Wireless devices have minimal processing power, so much of the API is intended to improve performance by minimizing the amount of work done in Java; this approach also has the added benefit of reducing application size. The API's are structured to provide considerable freedom when implementing them, thereby permitting the extensive use of native code, hardware acceleration and device-specific image data formats as needed.

The API uses the standard low-level graphics classes from MIDP (Graphics, Image, etc.) so that the high-level Game API classes can be used in conjunction with graphics primitives. For example, it would be possible to render a complex background using the Game API and then render something on top of it using graphics primitives such as drawLine, etc.

Methods that modify the state of Layer, LayerManager, Sprite, and TiledLayer objects generally do not have any immediately visible side effects. Instead, this state is merely stored within the object and is used during subsequent calls to the paint() method. This approach is suitable for gaming applications where there is a game cycle within which objects' states are updated, and where the entire screen is redrawn at the end of every game cycle.

API Overview

The API is comprised of five classes:

GameCanvas

This class is a subclass of LCDUI's Canvas and provides the basic 'screen' functionality for a game. In addition to the methods inherited from Canvas, this class also provides game-centric features such the ability to query the current state of the game keys and synchronous graphics flushing; these features simplify game development and improve performance.

Layer

The Layer class represents a visual element in a game such as a Sprite or a TiledLayer. This abstract class forms the basis for the Layer framework and provides basic attributes such as location, size, and visibility.

LayerManager

For games that employ several Layers, the LayerManager simplifies game development by automating the rendering process. It allows the developer set a view window that represents the user's view of the game. The LayerManager automatically renders the game's Layers to implement the desired view.

Sprite

A Sprite is basic animated Layer that can display one of several graphical frames. The frames are all of equal size and are provided by a single Image object. In addition to animating the frames sequentially, a custom sequence can also be set to animation the frames in an arbitrary manner. The Sprite class also provides various transformations (flip and rotation) and collision detection methods that simplify the implementation of a game's logic.

TiledLayer

This class enables a developer to create large areas of graphical content without the resource usage that a large Image object would require. It is a comprised of a grid of cells, and each cell can display one of several tiles that are provided by a single Image object. Cells can also be filled with animated tiles whose corresponding pixel data can be changed very rapidly; this feature is very useful for animating large groups of cells such as areas of water.

@since MIDP 2.0

Class Summary

Classes

GameCanvas 349	The GameCanvas class provides the basis for a game user interface.
Layer 356	A Layer is an abstract class representing a visual element of a game.
LayerManager 360	The LayerManager manages a series of Layers.
Sprite 365	A Sprite is a basic visual element that can be rendered with one of several frames stored in an Image; different frames can be shown to animate the Sprite.
TiledLayer 382	A TiledLayer is a visual element composed of a grid of cells that can be filled with a set of tile images.

javax.microedition.lcdui.game
GameCanvas

Declaration

```
public abstract class GameCanvas extends javax.microedition.lcdui.Canvas139

Object
  |
  +--javax.microedition.lcdui.Displayable218
        |
        +--javax.microedition.lcdui.Canvas139
              |
              +--javax.microedition.lcdui.game.GameCanvas
```

Description

The GameCanvas class provides the basis for a game user interface. In addition to the features inherited from Canvas (commands, input events, etc.) it also provides game-specific capabilities such as an off-screen graphics buffer and the ability to query key status.

A dedicated buffer is created for each GameCanvas instance. Since a unique buffer is provided for each GameCanvas instance, it is preferable to re-use a single GameCanvas instance in the interests of minimizing heap usage. The developer can assume that the contents of this buffer are modified only by calls to the Graphics object(s) obtained from the GameCanvas instance; the contents are not modified by external sources such as other MIDlets or system-level notifications. The buffer is initially filled with white pixels.

The buffer's size is set to the maximum dimensions of the GameCanvas. However, the area that may be flushed is limited by the current dimensions of the GameCanvas (as influenced by the presence of a Ticker, Commands, etc.) when the flush is requested. The current dimensions of the GameCanvas may be obtained by calling `getWidth`149 and `getHeight`148 .

A game may provide its own thread to run the game loop. A typical loop will check for input, implement the game logic, and then render the updated user interface. The following code illustrates the structure of a typcial game loop:

```
    // Get the Graphics object for the off-screen buffer
    Graphics g = getGraphics();
    while (true) {
         // Check user input and update positions if necessary
         int keyState = getKeyStates();
         if ((keyState & LEFT_PRESSED) != 0) {
             sprite.move(-1, 0);
         }
         else if ((keyState & RIGHT_PRESSED) != 0) {
             sprite.move(1, 0);
         }
   // Clear the background to white
   g.setColor(0xFFFFFF);
   g.fillRect(0,0,getWidth(), getHeight());
         // Draw the Sprite
         sprite.paint(g);
         // Flush the off-screen buffer
         flushGraphics();
    }
```

Since: MIDP 2.0

Member Summary

Fields

static int	DOWN_PRESSED[351]
static int	FIRE_PRESSED[351]
static int	GAME_A_PRESSED[351]
static int	GAME_B_PRESSED[351]
static int	GAME_C_PRESSED[351]
static int	GAME_D_PRESSED[351]
static int	LEFT_PRESSED[352]
static int	RIGHT_PRESSED[352]
static int	UP_PRESSED[352]

Constructors

protected	GameCanvas(boolean suppressKeyEvents)[352]

Methods

void	flushGraphics()[353]
void	flushGraphics(int x, int y, int width, int height)[353]
protected	getGraphics()[353]
javax.microedition.lcdui.Graphics	
int	getKeyStates()[354]
void	paint(javax.microedition.lcdui.Graphics g)[355]

Inherited Member Summary

Fields inherited from class Canvas[139]

DOWN[143], FIRE[144], GAME_A[144], GAME_B[144], GAME_C[144], GAME_D[144], KEY_NUM0[144], KEY_NUM1[145], KEY_NUM2[145], KEY_NUM3[145], KEY_NUM4[145], KEY_NUM5[145], KEY_NUM6[145], KEY_NUM7[146], KEY_NUM8[146], KEY_NUM9[146], KEY_POUND[146], KEY_STAR[146], LEFT[146], RIGHT[147], UP[147]

Methods inherited from class Canvas[139]

getGameAction(int)[147], getHeight()[148], getKeyCode(int)[148], getKeyName(int)[148], getWidth()[149], hasPointerEvents()[149], hasPointerMotionEvents()[149], hasRepeatEvents()[149], hideNotify()[149], isDoubleBuffered()[150], keyPressed(int)[150], keyReleased(int)[150], keyRepeated(int)[150], pointerDragged(int, int)[152], pointerPressed(int, int)[152], pointerReleased(int, int)[152], repaint()[152], repaint()[152], serviceRepaints()[153], setFullScreenMode(boolean)[153], showNotify()[154], sizeChanged(int, int)[154]

Methods inherited from class Displayable[218]

addCommand(Command)[219], getTicker()[219], getTitle()[220], isShown()[220], removeCommand(Command)[220], setCommandListener(CommandListener)[221], setTicker(Ticker)[221], setTitle(String)[221]

Methods inherited from class Object

equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Fields

DOWN_PRESSED

Declaration:
`public static final int DOWN_PRESSED`

Description:
The bit representing the DOWN key. This constant has a value of `0x0040` (1 << Canvas.DOWN).

FIRE_PRESSED

Declaration:
`public static final int FIRE_PRESSED`

Description:
The bit representing the FIRE key. This constant has a value of `0x0100` (1 << Canvas.FIRE).

GAME_A_PRESSED

Declaration:
`public static final int GAME_A_PRESSED`

Description:
The bit representing the GAME_A key (may not be supported on all devices). This constant has a value of `0x0200` (1 << Canvas.GAME_A).

GAME_B_PRESSED

Declaration:
`public static final int GAME_B_PRESSED`

Description:
The bit representing the GAME_B key (may not be supported on all devices). This constant has a value of `0x0400` (1 << Canvas.GAME_B).

GAME_C_PRESSED

Declaration:
`public static final int GAME_C_PRESSED`

Description:
The bit representing the GAME_C key (may not be supported on all devices). This constant has a value of `0x0800` (1 << Canvas.GAME_C).

GAME_D_PRESSED

Declaration:
`public static final int GAME_D_PRESSED`

Description:
The bit representing the GAME_D key (may not be supported on all devices). This constant has a value of `0x1000` (1 << Canvas.GAME_D).

LEFT_PRESSED

LEFT_PRESSED

Declaration:
```
public static final int LEFT_PRESSED
```

Description:
The bit representing the LEFT key. This constant has a value of `0x0004` (1 << Canvas.LEFT).

RIGHT_PRESSED

Declaration:
```
public static final int RIGHT_PRESSED
```

Description:
The bit representing the RIGHT key. This constant has a value of `0x0020` (1 << Canvas.RIGHT).

UP_PRESSED

Declaration:
```
public static final int UP_PRESSED
```

Description:
The bit representing the UP key. This constant has a value of `0x0002` (1 << Canvas.UP).

Constructors

GameCanvas(boolean)

Declaration:
```
protected GameCanvas(boolean suppressKeyEvents)
```

Description:
Creates a new instance of a GameCanvas. A new buffer is also created for the GameCanvas and is initially filled with white pixels.

If the developer only needs to query key status using the getKeyStates method, the regular key event mechanism can be suppressed for game keys while this GameCanvas is shown. If not needed by the application, the suppression of key events may improve performance by eliminating unnecessary system calls to keyPressed, keyRepeated and keyReleased methods.

If requested, key event suppression for a given GameCanvas is started when it is shown (i.e. when showNotify is called) and stopped when it is hidden (i.e. when hideNotify is called). Since the showing and hiding of screens is serialized with the event queue, this arrangement ensures that the suppression effects only those key events intended for the corresponding GameCanvas. Thus, if key events are being generated while another screen is still shown, those key events will continue to be queued and dispatched until that screen is hidden and the GameCanvas has replaced it.

Note that key events can be suppressed only for the defined game keys (UP, DOWN, FIRE, etc.); key events are always generated for all other keys.

Parameters:
`suppressKeyEvents` - `true` to suppress the regular key event mechanism for game keys, otherwise `false`.

Methods

flushGraphics()

Declaration:
`public void flushGraphics()`

Description:
Flushes the off-screen buffer to the display. The size of the flushed area is equal to the size of the GameCanvas. The contents of the off-screen buffer are not changed as a result of the flush operation. This method does not return until the flush has been completed, so the app may immediately begin to render the next frame to the same buffer once this method returns.

This method does nothing and returns immediately if the GameCanvas is not currently shown or the flush request cannot be honored because the system is busy.

See Also: `flushGraphics(int, int, int, int)` 353

flushGraphics(int, int, int, int)

Declaration:
`public void flushGraphics(int x, int y, int width, int height)`

Description:
Flushes the specified region of the off-screen buffer to the display. The contents of the off-screen buffer are not changed as a result of the flush operation. This method does not return until the flush has been completed, so the app may immediately begin to render the next frame to the same buffer once this method returns.

If the specified region extends beyond the current bounds of the GameCanvas, only the intersecting region is flushed. No pixels are flushed if the specified width or height is less than 1.

This method does nothing and returns immediately if the GameCanvas is not currently shown or the flush request cannot be honored because the system is busy.

Parameters:

`x` - the left edge of the region to be flushed

`y` - the top edge of the region to be flushed

`width` - the width of the region to be flushed

`height` - the height of the region to be flushed

See Also: `flushGraphics()` 353

getGraphics()

Declaration:
`protected javax.microedition.lcdui.Graphics` 247 `getGraphics()`

Description:
Obtains the Graphics object for rendering a GameCanvas. The returned Graphics object renders to the off-screen buffer belonging to this GameCanvas.

Rendering operations do not appear on the display until flushGraphics() is called; flushing the buffer does not change its contents (the pixels are not cleared as a result of the flushing operation).

A new Graphics object is created and returned each time this method is called; therefore, the needed Graphics object(s) should be obtained before the game starts then re-used while the game is running. For each GameCanvas instance, all of the provided graphics objects will render to the same off-screen buffer.

The newly created Graphics object has the following properties:

- the destination is this GameCanvas' buffer;
- the clip region encompasses the entire buffer;
- the current color is black;
- the font is the same as the font returned by `Font.getDefaultFont()`$_{227}$;
- the stroke style is `SOLID`$_{254}$; and
- the origin of the coordinate system is located at the upper-left corner of the buffer.

Returns: the Graphics object that renders to this GameCanvas' off-screen buffer

See Also: `flushGraphics()`$_{353}$, `flushGraphics(int, int, int, int)`$_{353}$

getKeyStates()

Declaration:
```
public int getKeyStates()
```

Description:
Gets the states of the physical game keys. Each bit in the returned integer represents a specific key on the device. A key's bit will be 1 if the key is currently down or has been pressed at least once since the last time this method was called. The bit will be 0 if the key is currently up and has not been pressed at all since the last time this method was called. This latching behavior ensures that a rapid key press and release will always be caught by the game loop, regardless of how slowly the loop runs.

For example:

```
// Get the key state and store it
int keyState = getKeyStates();
if ((keyState & LEFT_KEY) != 0) {
    positionX--;
}
else if ((keyState & RIGHT_KEY) != 0) {
    positionX++;
}
```

Calling this method has the side effect of clearing any latched state. Another call to getKeyStates immediately after a prior call will therefore report the system's best idea of the current state of the keys, the latched bits having been cleared by the first call.

Some devices may not be able to query the keypad hardware directly and therefore, this method may be implemented by monitoring key press and release events instead. Thus the state reported by getKeyStates might lag the actual state of the physical keys since the timeliness of the key information is be subject to the capabilities of each device. Also, some devices may be incapable of detecting simultaneous presses of multiple keys.

This method returns 0 unless the GameCanvas is currently visible as reported by `javax.microedition.lcdui.Displayable.isShown()`$_{220}$. Upon becoming visible, a GameCanvas will initially indicate that all keys are unpressed (0); if a key is held down while the GameCanvas is being shown, the key must be first released and then pressed in order for the key press to be reported by the GameCanvas.

Returns: An integer containing the key state information (one bit per key), or 0 if the GameCanvas is not currently shown.

See Also: `UP_PRESSED`[352], `DOWN_PRESSED`[351], `LEFT_PRESSED`[352], `RIGHT_PRESSED`[352], `FIRE_PRESSED`[351], `GAME_A_PRESSED`[351], `GAME_B_PRESSED`[351], `GAME_C_PRESSED`[351], `GAME_D_PRESSED`[351]

paint(Graphics)

Declaration:
```
public void paint(javax.microedition.lcdui.Graphics[247] g)
```

Description:
Paints this GameCanvas. By default, this method renders the the off-screen buffer at (0,0). Rendering of the buffer is subject to the clip region and origin translation of the Graphics object.

Overrides: `paint`[151] in class `Canvas`[139]

Parameters:
`g` - the Graphics object with which to render the screen.

Throws:
`NullPointerException` - if `g` is `null`

javax.microedition.lcdui.game
Layer

Declaration

```
public abstract class Layer

Object
  |
  +--javax.microedition.lcdui.game.Layer
```

Direct Known Subclasses: `Sprite`[365], `TiledLayer`[382]

Description

A Layer is an abstract class representing a visual element of a game. Each Layer has position (in terms of the upper-left corner of its visual bounds), width, height, and can be made visible or invisible. Layer subclasses must implement a `paint(Graphics)`[358] method so that they can be rendered.

The Layer's (x,y) position is always interpreted relative to the coordinate system of the Graphics object that is passed to the Layer's paint() method. This coordinate system is referred to as the *painter's* coordinate system. The initial location of a Layer is (0,0).

Since: MIDP 2.0

Member Summary

Methods

int	`getHeight()`[357]
int	`getWidth()`[357]
int	`getX()`[357]
int	`getY()`[357]
boolean	`isVisible()`[357]
void	`move(int dx, int dy)`[358]
abstract void	`paint(javax.microedition.lcdui.Graphics g)`[358]
void	`setPosition(int x, int y)`[358]
void	`setVisible(boolean visible)`[358]

Inherited Member Summary

Methods inherited from class `Object`

`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Methods

getHeight()

Declaration:
`public final int getHeight()`

Description:
Gets the current height of this layer, in pixels.

Returns: the height in pixels

See Also: `getWidth()` 357

getWidth()

Declaration:
`public final int getWidth()`

Description:
Gets the current width of this layer, in pixels.

Returns: the width in pixels

See Also: `getHeight()` 357

getX()

Declaration:
`public final int getX()`

Description:
Gets the horizontal position of this Layer's upper-left corner in the painter's coordinate system.

Returns: the Layer's horizontal position.

See Also: `getY()` 357, `setPosition(int, int)` 358, `move(int, int)` 358

getY()

Declaration:
`public final int getY()`

Description:
Gets the vertical position of this Layer's upper-left corner in the painter's coordinate system.

Returns: the Layer's vertical position.

See Also: `getX()` 357, `setPosition(int, int)` 358, `move(int, int)` 358

isVisible()

Declaration:
`public final boolean isVisible()`

Description:
Gets the visibility of this Layer.

Returns: `true` if the `Layer` is visible, `false` if it is invisible.

See Also: `setVisible(boolean)` 358

move(int, int)

Declaration:
```
public void move(int dx, int dy)
```

Description:
Moves this Layer by the specified horizontal and vertical distances.
The Layer's coordinates are subject to wrapping if the passed parameters will cause them to exceed beyond Integer.MAX_VALUE or Integer.MIN_VALUE.

Parameters:
`dx` - the distance to move along horizontal axis (positive to the right, negative to the left)

`dy` - the distance to move along vertical axis (positive down, negative up)

See Also: `setPosition(int, int)`[358], `getX()`[357], `getY()`[357]

paint(Graphics)

Declaration:
```
public abstract void paint(javax.microedition.lcdui.Graphics[247] g)
```

Description:
Paints this Layer if it is visible. The upper-left corner of the Layer is rendered at it's current (x,y) position relative to the origin of the provided Graphics object. Applications may make use of Graphics clipping and translation to control where the Layer is rendered and to limit the region that is rendered.

Implementations of this method are responsible for checking if this Layer is visible; this method does nothing if the Layer is not visible.

The attributes of the Graphics object (clip region, translation, drawing color, etc.) are not modified as a result of calling this method.

Parameters:
`g` - the graphics object for rendering the `Layer`

Throws:
`NullPointerException` - if `g` is `null`

setPosition(int, int)

Declaration:
```
public void setPosition(int x, int y)
```

Description:
Sets this Layer's position such that its upper-left corner is located at (x,y) in the painter's coordinate system. A Layer is located at (0,0) by default.

Parameters:
`x` - the horizontal position

`y` - the vertical position

See Also: `move(int, int)`[358], `getX()`[357], `getY()`[357]

setVisible(boolean)

Declaration:
```
public void setVisible(boolean visible)
```

Description:
Sets the visibility of this Layer. A visible Layer is rendered when its `paint(Graphics)`[358] method is called; an invisible Layer is not rendered.

Parameters:
`visible` - `true` to make the `Layer` visible, `false` to make it invisible

See Also: `isVisible()`[357]

javax.microedition.lcdui.game
LayerManager

Declaration

```
public class LayerManager

Object
  |
  +--javax.microedition.lcdui.game.LayerManager
```

Description

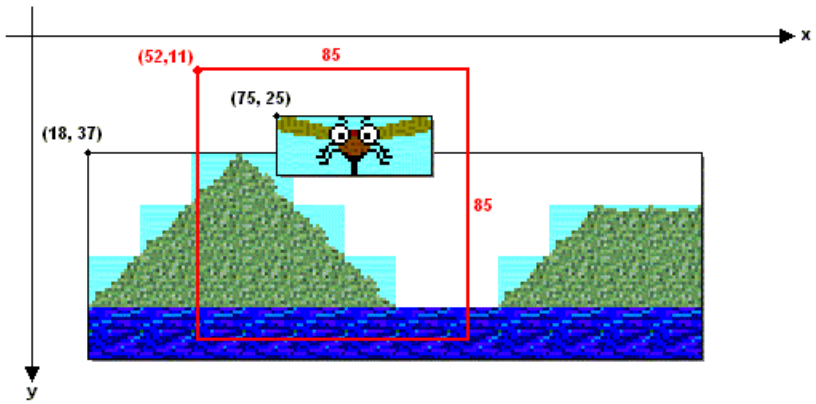
The LayerManager manages a series of Layers. The LayerManager simplifies the process of rendering the Layers that have been added to it by automatically rendering the correct regions of each Layer in the appropriate order.

The LayerManager maintains an ordered list to which Layers can be appended, inserted and removed. A Layer's index correlates to its z-order; the layer at index 0 is closest to the user while a the Layer with the highest index is furthest away from the user. The indices are always contiguous; that is, if a Layer is removed, the indices of subsequent Layers will be adjusted to maintain continuity.

The LayerManager class provides several features that control how the game's Layers are rendered on the screen.

The *view window* controls the size of the visible region and its position relative to the LayerManager's coordinate system. Changing the position of the view window enables effects such as scrolling or panning the user's view. For example, to scroll to the right, simply move the view window's location to the right. The size of the view window controls how large the user's view will be, and is usually fixed at a size that is appropriate for the device's screen.

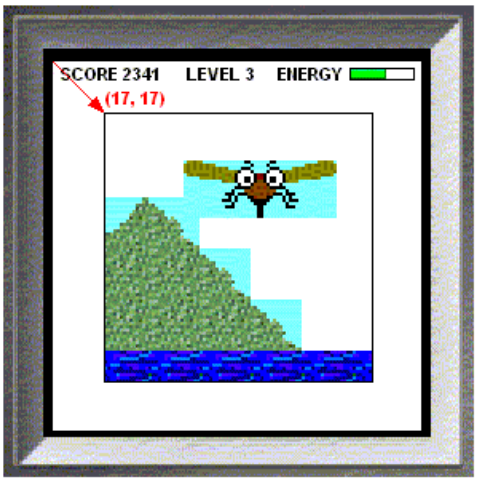
In this example, the view window is set to 85 x 85 pixels and is located at (52, 11) in the LayerManager's coordinate system. The Layers appear at their respective positions relative to the LayerManager's origin.



The `paint(Graphics, int, int)`[363] method includes an (x,y) location that controls where the view window is rendered relative to the screen. Changing these parameters does not change the contents of the view

window, it simply changes the location where the view window is drawn. Note that this location is relative to the origin of the Graphics object, and thus it is subject to the translation attributes of the Graphics object.

For example, if a game uses the top of the screen to display the current score, the view window may be rendered at (17, 17) to provide enough space for the score.



Since: MIDP 2.0

Member Summary

Constructors

	`LayerManager()`[362]

Methods

void	`append(Layer l)`[362]
Layer	`getLayerAt(int index)`[362]
int	`getSize()`[362]
void	`insert(Layer l, int index)`[363]
void	`paint(javax.microedition.lcdui.Graphics g, int x, int y)`[363]
void	`remove(Layer l)`[364]
void	`setViewWindow(int x, int y, int width, int height)`[364]

Inherited Member Summary

Methods inherited from class `Object`

`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Constructors

LayerManager()

Declaration:
`public LayerManager()`

Description:
Creates a new LayerManager.

Methods

append(Layer)

Declaration:
`public void append(javax.microedition.lcdui.game.Layer`[356] `l)`

Description:
Appends a Layer to this LayerManager. The Layer is appended to the list of existing Layers such that it has the highest index (i.e. it is furthest away from the user). The Layer is first removed from this LayerManager if it has already been added.

Parameters:
`l` - the `Layer` to be added

Throws:
`NullPointerException` - if the `Layer` is `null`

See Also: `insert(Layer, int)`[363], `remove(Layer)`[364]

getLayerAt(int)

Declaration:
`public javax.microedition.lcdui.game.Layer`[356] `getLayerAt(int index)`

Description:
Gets the Layer with the specified index.

Parameters:
`index` - the index of the desired Layer

Returns: the Layer that has the specified index

Throws:
`IndexOutOfBoundsException` - if the specified `index` is less than zero, or if it is equal to or greater than the number of Layers added to the this `LayerManager`

getSize()

Declaration:
`public int getSize()`

Description:
Gets the number of Layers in this LayerManager.

Returns: the number of Layers

insert(Layer, int)

Declaration:
```
public void insert(javax.microedition.lcdui.game.Layer 356 l, int index)
```

Description:
Inserts a new Layer in this LayerManager at the specified index. The Layer is first removed from this LayerManager if it has already been added.

Parameters:
`l` - the `Layer` to be inserted

`index` - the index at which the new `Layer` is to be inserted

Throws:
`NullPointerException` - if the `Layer` is `null`

`IndexOutOfBoundsException` - if the index is less than `0` or greater than the number of Layers already added to the this `LayerManager`

See Also: `append(Layer)` 362, `remove(Layer)` 364

paint(Graphics, int, int)

Declaration:
```
public void paint(javax.microedition.lcdui.Graphics 247 g, int x, int y)
```

Description:
Renders the LayerManager's current view window at the specified location.

The LayerManager renders each of its layers in order of descending index, thereby implementing the correct z-order. Layers that are completely outside of the view window are not rendered.

The coordinates passed to this method determine where the LayerManager's view window will be rendered relative to the origin of the Graphics object. For example, a game may use the top of the screen to display the current score, so to render the game's layers below that area, the view window might be rendered at (0, 20). The location is relative to the Graphics object's origin, so translating the Graphics object will change where the view window is rendered on the screen.

The clip region of the Graphics object is intersected with a region having the same dimensions as the view window and located at (x,y). The LayerManager then translates the graphics object such that the point (x,y) corresponds to the location of the viewWindow in the coordinate system of the LayerManager. The Layers are then rendered in the appropriate order. The translation and clip region of the Graphics object are restored to their prior values before this method returns.

Rendering is subject to the clip region and translation of the Graphics object. Thus, only part of the specified view window may be rendered if the clip region is not large enough.

For performance reasons, this method may ignore Layers that are invisible or that would be rendered entirely outside of the Graphics object's clip region. The attributes of the Graphics object are not restored to a known state between calls to the Layers' paint methods. The clip region may extend beyond the bounds of a Layer; it is the responsibility of the Layer to ensure that rendering operations are performed within its bounds.

Parameters:
`g` - the graphics instance with which to draw the LayerManager

`x` - the horizontal location at which to render the view window, relative to the Graphics' translated origin

`y` - the vertical location at which to render the view window, relative to the Graphics' translated origin

Throws:
`NullPointerException` - if `g` is `null`

See Also: `setViewWindow(int, int, int, int)`[364]

remove(Layer)

Declaration:
```
public void remove(javax.microedition.lcdui.game.Layer[356] l)
```

Description:
Removes the specified Layer from this LayerManager. This method does nothing if the specified Layer is not added to the this LayerManager.

Parameters:
`l` - the `Layer` to be removed

Throws:
`NullPointerException` - if the specified `Layer` is `null`

See Also: `append(Layer)`[362], `insert(Layer, int)`[363]

setViewWindow(int, int, int, int)

Declaration:
```
public void setViewWindow(int x, int y, int width, int height)
```

Description:
Sets the view window on the LayerManager.

The view window specifies the region that the LayerManager draws when its `paint(Graphics, int, int)`[363] method is called. It allows the developer to control the size of the visible region, as well as the location of the view window relative to the LayerManager's coordinate system.

The view window stays in effect until it is modified by another call to this method. By default, the view window is located at (0,0) in the LayerManager's coordinate system and its width and height are both set to Integer.MAX_VALUE.

Parameters:
`x` - the horizontal location of the view window relative to the LayerManager's origin

`y` - the vertical location of the view window relative to the LayerManager's origin

`width` - the width of the view window

`height` - the height of the view window

Throws:
`IllegalArgumentException` - if the `width` or `height` is less than `0`

javax.microedition.lcdui.game
Sprite

Declaration

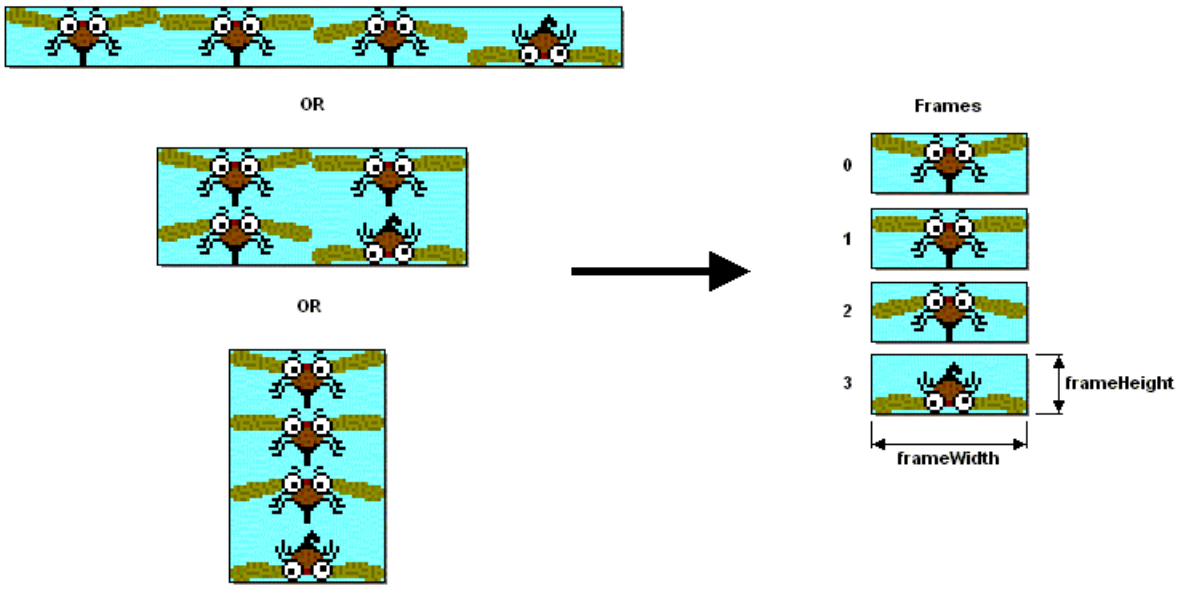
```
public class Sprite extends Layer356

Object
  |
  +--javax.microedition.lcdui.game.Layer356
        |
        +--javax.microedition.lcdui.game.Sprite
```

Description

A Sprite is a basic visual element that can be rendered with one of several frames stored in an Image; different frames can be shown to animate the Sprite. Several transforms such as flipping and rotation can also be applied to a Sprite to further vary its appearance. As with all Layer subclasses, a Sprite's location can be changed and it can also be made visible or invisible.

Sprite Frames

The raw frames used to render a Sprite are provided in a single Image object, which may be mutable or immutable. If more than one frame is used, the Image is broken up into a series of equally-sized frames of a specified width and height. As shown in the figure below, the same set of frames may be stored in several different arrangements depending on what is the most convenient for the game developer.

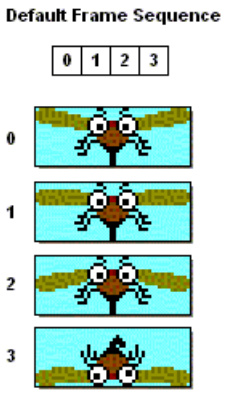


Each frame is assigned a unique index number. The frame located in the upper-left corner of the Image is assigned an index of 0. The remaining frames are then numbered consecutively in row-major order (indices are

assigned across the first row, then the second row, and so on). The method `getRawFrameCount()`[377] returns the total number of raw frames.

Frame Sequence

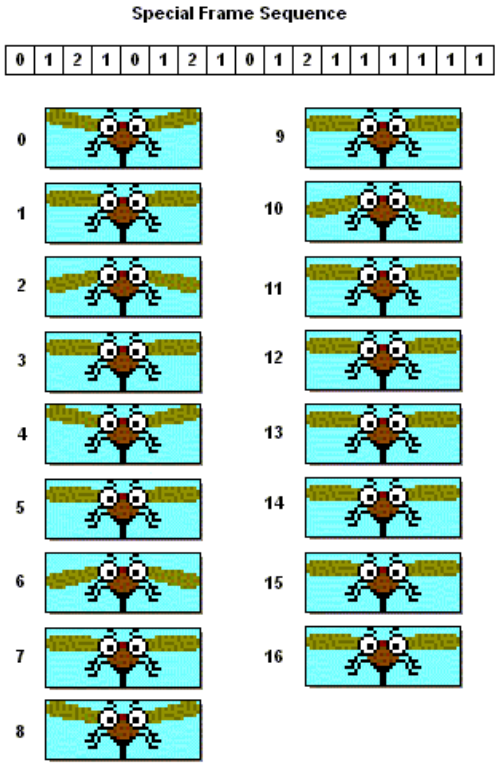
A Sprite's frame sequence defines an ordered list of frames to be displayed. The default frame sequence mirrors the list of available frames, so there is a direct mapping between the sequence index and the corresponding frame index. This also means that the length of the default frame sequence is equal to the number of raw frames. For example, if a Sprite has 4 frames, its default frame sequence is {0, 1, 2, 3}.



The developer must manually switch the current frame in the frame sequence. This may be accomplished by calling `setFrame(int)`[378], `prevFrame()`[378], or `nextFrame()`[378]. Note that these methods always operate on the sequence index, they do not operate on frame indices; however, if the default frame sequence is used, then the sequence indices and the frame indices are interchangeable.

If desired, an arbitrary frame sequence may be defined for a Sprite. The frame sequence must contain at least one element, and each element must reference a valid frame index. By defining a new frame sequence, the developer can conveniently display the Sprite's frames in any order desired; frames may be repeated, omitted, shown in reverse order, etc.

For example, the diagram below shows how a special frame sequence might be used to animate a mosquito. The frame sequence is designed so that the mosquito flaps its wings three times and then pauses for a moment before the cycle is repeated.



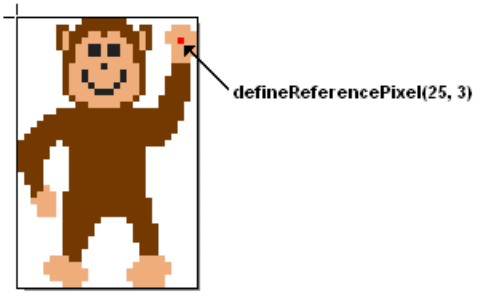
By calling `nextFrame()`[378] each time the display is updated, the resulting animation would like this:

Reference Pixel

Being a subclass of Layer, Sprite inherits various methods for setting and retrieving its location such as `setPosition(x,y)`[358], `getX()`[357], and `getY()`[357]. These methods all define position in terms of the upper-left corner of the Sprite's visual bounds; however, in some cases, it is more convenient to define the Sprite's position in terms of an arbitrary pixel within its frame, especially if transforms are applied to the Sprite.

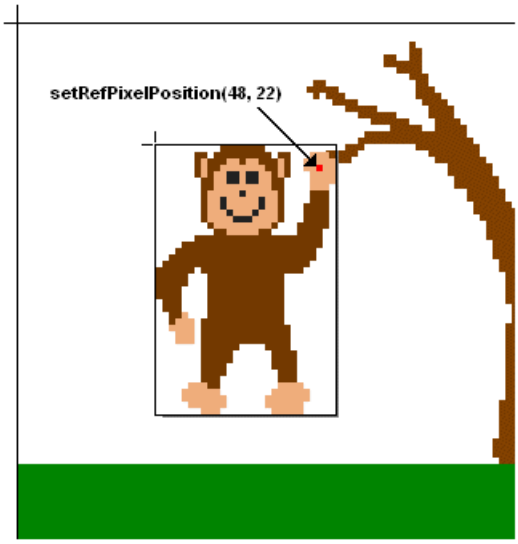
Therefore, Sprite includes the concept of a *reference pixel*. The reference pixel is defined by specifying its location in the Sprite's untransformed frame using `defineReferencePixel(x,y)`[376]. By default, the reference pixel is defined to be the pixel at (0,0) in the frame. If desired, the reference pixel may be defined outside of the frame's bounds.

In this example, the reference pixel is defined to be the pixel that the monkey appears to be hanging from:



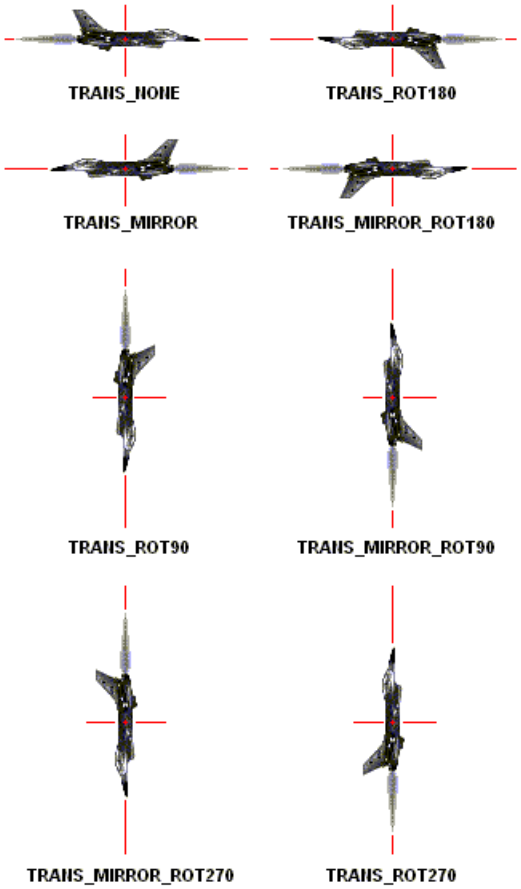
`getRefPixelX()`$_{377}$ and `getRefPixelY()`$_{377}$ can be used to query the location of the reference pixel in the painter's coordinate system. The developer can also use `setRefPixelPosition(x,y)`$_{380}$ to position the Sprite so that reference pixel appears at a specific location in the painter's coordinate system. These methods automatically account for any transforms applied to the Sprite.

In this example, the reference pixel's position is set to a point at the end of a tree branch; the Sprite's location changes so that the reference pixel appears at this point and the monkey appears to be hanging from the branch:



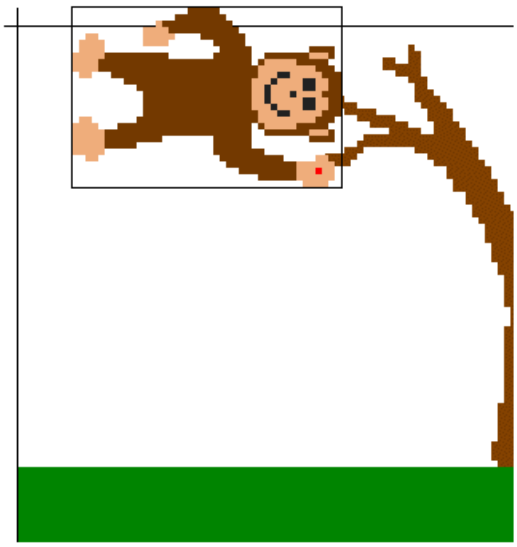
Sprite Transforms

Various transforms can be applied to a Sprite. The available transforms include rotations in multiples of 90 degrees, and mirrored (about the vertical axis) versions of each of the rotations. A Sprite's transform is set by calling `setTransform(transform)`$_{380}$.



When a transform is applied, the Sprite is automatically repositioned such that the reference pixel appears stationary in the painter's coordinate system. Thus, the reference pixel effectively becomes the center of the transform operation. Since the reference pixel does not move, the values returned by `getRefPixelX()`$_{377}$ and `getRefPixelY()`$_{377}$ remain the same; however, the values returned by `getX()`$_{357}$ and `getY()`$_{357}$ may change to reflect the movement of the Sprite's upper-left corner.

Referring to the monkey example once again, the position of the reference pixel remains at (48, 22) when a 90 degree rotation is applied, thereby making it appear as if the monkey is swinging from the branch:



Sprite Drawing
Sprites can be drawn at any time using the `paint(Graphics)`[378] method. The Sprite will be drawn on the Graphics object according to the current state information maintained by the Sprite (i.e. position, frame, visibility). Erasing the Sprite is always the responsibility of code outside the Sprite class.

Sprites can be implemented using whatever techniques a manufacturers wishes to use (e.g hardware acceleration may be used for all Sprites, for certain sizes of Sprites, or not at all).

For some platforms, certain Sprite sizes may be more efficient than others; manufacturers may choose to provide developers with information about device-specific characteristics such as these.

Since: MIDP 2.0

Member Summary

Fields

static int	`TRANS_MIRROR`[371]
static int	`TRANS_MIRROR_ROT180`[371]
static int	`TRANS_MIRROR_ROT270`[372]
static int	`TRANS_MIRROR_ROT90`[372]
static int	`TRANS_NONE`[372]
static int	`TRANS_ROT180`[372]
static int	`TRANS_ROT270`[372]
static int	`TRANS_ROT90`[372]

Constructors

`Sprite(javax.microedition.lcdui.Image image)`[373]
`Sprite(javax.microedition.lcdui.Image image, int frameWidth, int frameHeight)`[373]
`Sprite(Sprite s)`[374]

Methods

boolean	`collidesWith(javax.microedition.lcdui.Image image, int x, int y, boolean pixelLevel)`[374]

Member Summary	
boolean	collidesWith(Sprite s, boolean pixelLevel)[374]
boolean	collidesWith(TiledLayer t, boolean pixelLevel)[375]
void	defineCollisionRectangle(int x, int y, int width, int height)[375]
void	defineReferencePixel(int x, int y)[376]
int	getFrame()[376]
int	getFrameSequenceLength()[377]
int	getRawFrameCount()[377]
int	getRefPixelX()[377]
int	getRefPixelY()[377]
void	nextFrame()[378]
void	paint(javax.microedition.lcdui.Graphics g)[378]
void	prevFrame()[378]
void	setFrame(int sequenceIndex)[378]
void	setFrameSequence(int[] sequence)[379]
void	setImage(javax.microedition.lcdui.Image img, int frameWidth, int frameHeight)[379]
void	setRefPixelPosition(int x, int y)[380]
void	setTransform(int transform)[380]

Inherited Member Summary

Methods inherited from class `Layer`[356]

getHeight()[357], getWidth()[357], getX()[357], getY()[357], isVisible()[357], move(int, int)[358], setPosition(int, int)[358], setVisible(boolean)[358]

Methods inherited from class `Object`

equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Fields

TRANS_MIRROR

Declaration:
```
public static final int TRANS_MIRROR
```

Description:
Causes the Sprite to appear reflected about its vertical center. This constant has a value of 2.

TRANS_MIRROR_ROT180

Declaration:
```
public static final int TRANS_MIRROR_ROT180
```

TRANS_MIRROR_ROT270

Description:
Causes the Sprite to appear reflected about its vertical center and then rotated clockwise by 180 degrees. This constant has a value of `1`.

TRANS_MIRROR_ROT270

Declaration:
```
public static final int TRANS_MIRROR_ROT270
```

Description:
Causes the Sprite to appear reflected about its vertical center and then rotated clockwise by 270 degrees. This constant has a value of `4`.

TRANS_MIRROR_ROT90

Declaration:
```
public static final int TRANS_MIRROR_ROT90
```

Description:
Causes the Sprite to appear reflected about its vertical center and then rotated clockwise by 90 degrees. This constant has a value of `7`.

TRANS_NONE

Declaration:
```
public static final int TRANS_NONE
```

Description:
No transform is applied to the Sprite. This constant has a value of `0`.

TRANS_ROT180

Declaration:
```
public static final int TRANS_ROT180
```

Description:
Causes the Sprite to appear rotated clockwise by 180 degrees. This constant has a value of `3`.

TRANS_ROT270

Declaration:
```
public static final int TRANS_ROT270
```

Description:
Causes the Sprite to appear rotated clockwise by 270 degrees. This constant has a value of `6`.

TRANS_ROT90

Declaration:
```
public static final int TRANS_ROT90
```

Description:
Causes the Sprite to appear rotated clockwise by 90 degrees. This constant has a value of `5`.

Constructors

Sprite(Image)

Declaration:
```
public Sprite(javax.microedition.lcdui.Image[270] image)
```

Description:
Creates a new non-animated Sprite using the provided Image. This constructor is functionally equivalent to calling `new Sprite(image, image.getWidth(), image.getHeight())`

By default, the Sprite is visible and its upper-left corner is positioned at (0,0) in the painter's coordinate system.

Parameters:
`image` - the `Image` to use as the single frame for the Sprite

Throws:
`NullPointerException` - if `img` is `null`

Sprite(Image, int, int)

Declaration:
```
public Sprite(javax.microedition.lcdui.Image[270] image, int frameWidth, int frameHeight)
```

Description:
Creates a new animated Sprite using frames contained in the provided Image. The frames must be equally sized, with the dimensions specified by `frameWidth` and `frameHeight`. They may be laid out in the image horizontally, vertically, or as a grid. The width of the source image must be an integer multiple of the frame width, and the height of the source image must be an integer multiple of the frame height. The values returned by `Layer.getWidth()`[357] and `Layer.getHeight()`[357] will reflect the frame width and frame height subject to the Sprite's current transform.

Sprites have a default frame sequence corresponding to the raw frame numbers, starting with frame 0. The frame sequence may be modified with `setFrameSequence(int[])`[379].

By default, the Sprite is visible and its upper-left corner is positioned at (0,0) in the painter's coordinate system.

Parameters:
`image` - the `Image` to use for `Sprite`

`frameWidth` - the `width`, in pixels, of the individual raw frames

`frameHeight` - the `height`, in pixels, of the individual raw frames

Throws:
`NullPointerException` - if `img` is `null`

`IllegalArgumentException` - if `frameHeight` or `frameWidth` is less than `1`

`IllegalArgumentException` - if the `image` width is not an integer multiple of the `frameWidth`

`IllegalArgumentException` - if the `image` height is not an integer multiple of the `frameHeight`

Sprite(Sprite)

Declaration:
```
public Sprite(javax.microedition.lcdui.game.Sprite[365] s)
```

Description:
Creates a new Sprite from another Sprite.

All instance attributes (raw frames, position, frame sequence, current frame, reference point, collision rectangle, transform, and visibility) of the source Sprite are duplicated in the new Sprite.

Parameters:
`s` - the `Sprite` to create a copy of

Throws:
`NullPointerException` - if `s` is `null`

Methods

collidesWith(Image, int, int, boolean)

Declaration:
```
public final boolean collidesWith(javax.microedition.lcdui.Image[270] image, int x, int y,
            boolean pixelLevel)
```

Description:
Checks for a collision between this Sprite and the specified Image with its upper left corner at the specified location. If pixel-level detection is used, a collision is detected only if opaque pixels collide. That is, an opaque pixel in the Sprite would have to collide with an opaque pixel in Image for a collision to be detected. Only those pixels within the Sprite's collision rectangle are checked.

If pixel-level detection is not used, this method simply checks if the Sprite's collision rectangle intersects with the Image's bounds.

Any transform applied to the Sprite is automatically accounted for.

The Sprite must be visible in order for a collision to be detected.

Parameters:
`image` - the `Image` to test for collision

`x` - the horizontal location of the `Image`'s upper left corner

`y` - the vertical location of the `Image`'s upper left corner

`pixelLevel` - `true` to test for collision on a pixel-by-pixel basis, `false` to test using simple bounds checking

Returns: `true` if this `Sprite` has collided with the `Image`, otherwise `false`

Throws:
`NullPointerException` - if `image` is `null`

collidesWith(Sprite, boolean)

Declaration:
```
public final boolean collidesWith(javax.microedition.lcdui.game.Sprite[365] s,
            boolean pixelLevel)
```

Description:
Checks for a collision between this Sprite and the specified Sprite.

If pixel-level detection is used, a collision is detected only if opaque pixels collide. That is, an opaque pixel in the first Sprite would have to collide with an opaque pixel in the second Sprite for a collision to be detected. Only those pixels within the Sprites' respective collision rectangles are checked.

If pixel-level detection is not used, this method simply checks if the Sprites' collision rectangles intersect.

Any transforms applied to the Sprites are automatically accounted for.

Both Sprites must be visible in order for a collision to be detected.

Parameters:
`s` - the `Sprite` to test for collision with

`pixelLevel` - `true` to test for collision on a pixel-by-pixel basis, `false` to test using simple bounds checking.

Returns: `true` if the two Sprites have collided, otherwise `false`

Throws:
`NullPointerException` - if Sprite `s` is `null`

collidesWith(TiledLayer, boolean)

Declaration:
```
public final boolean collidesWith(javax.microedition.lcdui.game.TiledLayer382 t,
            boolean pixelLevel)
```

Description:
Checks for a collision between this Sprite and the specified TiledLayer. If pixel-level detection is used, a collision is detected only if opaque pixels collide. That is, an opaque pixel in the Sprite would have to collide with an opaque pixel in TiledLayer for a collision to be detected. Only those pixels within the Sprite's collision rectangle are checked.

If pixel-level detection is not used, this method simply checks if the Sprite's collision rectangle intersects with a non-empty cell in the TiledLayer.

Any transform applied to the Sprite is automatically accounted for.

The Sprite and the TiledLayer must both be visible in order for a collision to be detected.

Parameters:
`t` - the `TiledLayer` to test for collision with

`pixelLevel` - `true` to test for collision on a pixel-by-pixel basis, `false` to test using simple bounds checking against non-empty cells.

Returns: `true` if this `Sprite` has collided with the `TiledLayer`, otherwise `false`

Throws:
`NullPointerException` - if `t` is `null`

defineCollisionRectangle(int, int, int, int)

Declaration:
```
public void defineCollisionRectangle(int x, int y, int width, int height)
```

Description:
Defines the Sprite's bounding rectangle that is used for collision detection purposes. This rectangle is specified relative to the un-transformed Sprite's upper-left corner and defines the area that is checked for collision detection. For pixel-level detection, only those pixels within the collision rectangle are checked. By default, a Sprite's collision rectangle is located at 0,0 as has the same dimensions as the Sprite. The collision rectangle may be specified to be larger or smaller than the default rectangle; if made larger, the pixels outside the bounds of the Sprite are considered to be transparent for pixel-level collision detection.

Parameters:

`x` - the horizontal location of the collision rectangle relative to the untransformed Sprite's left edge

`y` - the vertical location of the collision rectangle relative to the untransformed Sprite's top edge

`width` - the width of the collision rectangle

`height` - the height of the collision rectangle

Throws:

`IllegalArgumentException` - if the specified `width` or `height` is less than `0`

defineReferencePixel(int, int)

Declaration:
```
public void defineReferencePixel(int x, int y)
```

Description:
Defines the reference pixel for this Sprite. The pixel is defined by its location relative to the upper-left corner of the Sprite's un-transformed frame, and it may lay outside of the frame's bounds.

When a transformation is applied, the reference pixel is defined relative to the Sprite's initial upper-left corner before transformation. This corner may no longer appear as the upper-left corner in the painter's coordinate system under current transformation.

By default, a Sprite's reference pixel is located at (0,0); that is, the pixel in the upper-left corner of the raw frame.

Changing the reference pixel does not change the Sprite's physical position in the painter's coordinate system; that is, the values returned by `getX()` 357 and `getY()` 357 will not change as a result of defining the reference pixel. However, subsequent calls to methods that involve the reference pixel will be impacted by its new definition.

Parameters:

`x` - the horizontal location of the reference pixel, relative to the left edge of the un-transformed frame

`y` - the vertical location of the reference pixel, relative to the top edge of the un-transformed frame

See Also: `setRefPixelPosition(int, int)` 380, `getRefPixelX()` 377, `getRefPixelY()` 377

getFrame()

Declaration:
```
public final int getFrame()
```

Description:
Gets the current index in the frame sequence.

The index returned refers to the current entry in the frame sequence, not the index of the actual frame that is displayed.

Returns: the current index in the frame sequence

See Also: `setFrameSequence(int[])`[379], `setFrame(int)`[378]

getFrameSequenceLength()

Declaration:
```
public int getFrameSequenceLength()
```

Description:
Gets the number of elements in the frame sequence. The value returned reflects the length of the Sprite's frame sequence; it does not reflect the number of raw frames. However, these two values will be the same if the default frame sequence is used.

Returns: the number of elements in this Sprite's frame sequence

See Also: `getRawFrameCount()`[377]

getRawFrameCount()

Declaration:
```
public int getRawFrameCount()
```

Description:
Gets the number of raw frames for this Sprite. The value returned reflects the number of frames; it does not reflect the length of the Sprite's frame sequence. However, these two values will be the same if the default frame sequence is used.

Returns: the number of raw frames for this Sprite

See Also: `getFrameSequenceLength()`[377]

getRefPixelX()

Declaration:
```
public int getRefPixelX()
```

Description:
Gets the horizontal position of this Sprite's reference pixel in the painter's coordinate system.

Returns: the horizontal location of the reference pixel

See Also: `defineReferencePixel(int, int)`[376], `setRefPixelPosition(int, int)`[380], `getRefPixelY()`[377]

getRefPixelY()

Declaration:
```
public int getRefPixelY()
```

Description:
Gets the vertical position of this Sprite's reference pixel in the painter's coordinate system.

Returns: the vertical location of the reference pixel

See Also: `defineReferencePixel(int, int)`[376], `setRefPixelPosition(int, int)`[380], `getRefPixelX()`[377]

nextFrame()

Declaration:
```
public void nextFrame()
```

Description:
Selects the next frame in the frame sequence.

The frame sequence is considered to be circular, i.e. if `nextFrame()`[378] is called when at the end of the sequence, this method will advance to the first entry in the sequence.

See Also: `setFrameSequence(int[])`[379], `prevFrame()`[378]

paint(Graphics)

Declaration:
```
public final void paint(javax.microedition.lcdui.Graphics[247] g)
```

Description:
Draws the Sprite.

Draws current frame of Sprite using the provided Graphics object. The Sprite's upper left corner is rendered at the Sprite's current position relative to the origin of the Graphics object. The current position of the Sprite's upper-left corner can be retrieved by calling `Layer.getX()`[357] and `Layer.getY()`[357].

Rendering is subject to the clip region of the Graphics object. The Sprite will be drawn only if it is visible.

If the Sprite's Image is mutable, the Sprite is rendered using the current contents of the Image.

Overrides: `paint`[358] in class `Layer`[356]

Parameters:
`g` - the graphics object to draw `Sprite` on

Throws:
`NullPointerException` - if `g` is `null`

prevFrame()

Declaration:
```
public void prevFrame()
```

Description:
Selects the previous frame in the frame sequence.

The frame sequence is considered to be circular, i.e. if `prevFrame()`[378] is called when at the start of the sequence, this method will advance to the last entry in the sequence.

See Also: `setFrameSequence(int[])`[379], `nextFrame()`[378]

setFrame(int)

Declaration:
```
public void setFrame(int sequenceIndex)
```

Description:
Selects the current frame in the frame sequence.

The current frame is rendered when `paint(Graphics)`[378] is called.

The index provided refers to the desired entry in the frame sequence, not the index of the actual frame itself.

Parameters:
`sequenceIndex` - the index of of the desired entry in the frame sequence

Throws:
`IndexOutOfBoundsException` - if `frameIndex` is less than`0`

`IndexOutOfBoundsException` - if `frameIndex` is equal to or greater than the length of the current frame sequence (or the number of raw frames for the default sequence)

See Also: `setFrameSequence(int[])`$_{379}$, `getFrame()`$_{376}$

setFrameSequence(int[])

Declaration:
```
public void setFrameSequence(int[] sequence)
```

Description:
Set the frame sequence for this Sprite.

All Sprites have a default sequence that displays the Sprites frames in order. This method allows for the creation of an arbitrary sequence using the available frames. The current index in the frame sequence is reset to zero as a result of calling this method.

The contents of the sequence array are copied when this method is called; thus, any changes made to the array after this method returns have no effect on the Sprite's frame sequence.

Passing in `null` causes the Sprite to revert to the default frame sequence.

Parameters:
`sequence` - an array of integers, where each integer represents a frame index

Throws:
`ArrayIndexOutOfBoundsException` - if seq is non-null and any member of the array has a value less than `0` or greater than or equal to the number of frames as reported by `getRawFrameCount()`$_{377}$

`IllegalArgumentException` - if the array has less than `1` element

See Also: `nextFrame()`$_{378}$, `prevFrame()`$_{378}$, `setFrame(int)`$_{378}$, `getFrame()`$_{376}$

setImage(Image, int, int)

Declaration:
```
public void setImage(javax.microedition.lcdui.Image270 img, int frameWidth,
            int frameHeight)
```

Description:
Changes the Image containing the Sprite's frames.

Replaces the current raw frames of the Sprite with a new set of raw frames. See the constructor `Sprite(Image, int, int)`$_{373}$ for information on how the frames are created from the image. The values returned by `Layer.getWidth()`$_{357}$ and `Layer.getHeight()`$_{357}$ will reflect the new frame width and frame height subject to the Sprite's current transform.

Changing the image for the Sprite could change the number of raw frames. If the new frame set has as many or more raw frames than the previous frame set, then:

- The current frame will be unchanged
- If a custom frame sequence has been defined (using `setFrameSequence(int[])`$_{379}$), it will remain unchanged. If no custom frame sequence is defined (i.e. the default frame sequence is in use),

the default frame sequence will be updated to be the default frame sequence for the new frame set. In other words, the new default frame sequence will include all of the frames from the new raw frame set, as if this new image had been used in the constructor.

If the new frame set has fewer frames than the previous frame set, then:

- The current frame will be reset to entry 0
- Any custom frame sequence will be discarded and the frame sequence will revert to the default frame sequence for the new frame set.

The reference point location is unchanged as a result of calling this method, both in terms of its defined location within the Sprite and its position in the painter's coordinate system. However, if the frame size is changed and the Sprite has been transformed, the position of the Sprite's upper-left corner may change such that the reference point remains stationary.

If the Sprite's frame size is changed by this method, the collision rectangle is reset to its default value (i.e. it is set to the new bounds of the untransformed Sprite).

Parameters:

`img` - the `Image` to use for `Sprite`

`frameWidth` - the width in pixels of the individual raw frames

`frameHeight` - the height in pixels of the individual raw frames

Throws:

`NullPointerException` - if `img` is `null`

`IllegalArgumentException` - if `frameHeight` or `frameWidth` is less than `1`

`IllegalArgumentException` - if the image width is not an integer multiple of the `frameWidth`

`IllegalArgumentException` - if the image height is not an integer multiple of the `frameHeight`

setRefPixelPosition(int, int)

Declaration:

```
public void setRefPixelPosition(int x, int y)
```

Description:

Sets this Sprite's position such that its reference pixel is located at (x,y) in the painter's coordinate system.

Parameters:

`x` - the horizontal location at which to place the reference pixel

`y` - the vertical location at which to place the reference pixel

See Also: `defineReferencePixel(int, int)`376, `getRefPixelX()`377, `getRefPixelY()`377

setTransform(int)

Declaration:

```
public void setTransform(int transform)
```

Description:
Sets the transform for this Sprite. Transforms can be applied to a Sprite to change its rendered appearance. Transforms are applied to the original Sprite image; they are not cumulative, nor can they be combined. By default, a Sprite's transform is `TRANS_NONE`[372] .

Since some transforms involve rotations of 90 or 270 degrees, their use may result in the overall width and height of the Sprite being swapped. As a result, the values returned by `Layer.getWidth()`[357] and `Layer.getHeight()`[357] may change.

The collision rectangle is also modified by the transform so that it remains static relative to the pixel data of the Sprite. Similarly, the defined reference pixel is unchanged by this method, but its visual location within the Sprite may change as a result.

This method repositions the Sprite so that the location of the reference pixel in the painter's coordinate system does not change as a result of changing the transform. Thus, the reference pixel effectively becomes the centerpoint for the transform. Consequently, the values returned by `getRefPixelX()`[377] and `getRefPixelY()`[377] will be the same both before and after the transform is applied, but the values returned by `getX()`[357] and `getY()`[357] may change.

Parameters:
`transform` - the desired transform for this `Sprite`

Throws:
`IllegalArgumentException` - if the requested `transform` is invalid

See Also: `TRANS_NONE`[372], `TRANS_ROT90`[372], `TRANS_ROT180`[372], `TRANS_ROT270`[372], `TRANS_MIRROR`[371], `TRANS_MIRROR_ROT90`[372], `TRANS_MIRROR_ROT180`[371], `TRANS_MIRROR_ROT270`[372]

javax.microedition.lcdui.game
TiledLayer

Declaration

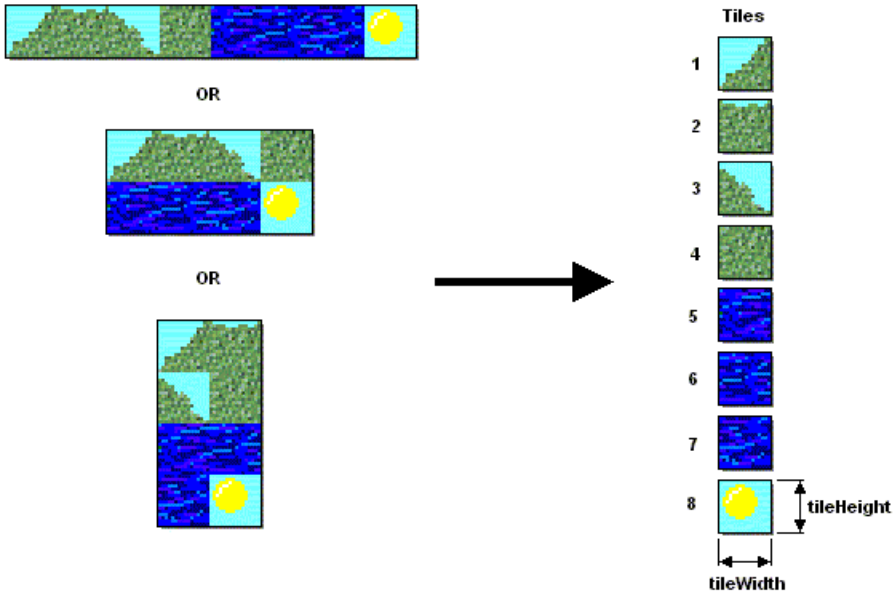
```
public class TiledLayer extends Layer356

Object
  |
  +--javax.microedition.lcdui.game.Layer356
        |
        +--javax.microedition.lcdui.game.TiledLayer
```

Description

A TiledLayer is a visual element composed of a grid of cells that can be filled with a set of tile images. This class allows large virtual layers to be created without the need for an extremely large Image. This technique is commonly used in 2D gaming platforms to create very large scrolling backgrounds,

Tiles

The tiles used to fill the TiledLayer's cells are provided in a single Image object which may be mutable or immutable. The Image is broken up into a series of equally-sized tiles; the tile size is specified along with the Image. As shown in the figure below, the same tile set can be stored in several different arrangements depending on what is the most convenient for the game developer.



Each tile is assigned a unique index number. The tile located in the upper-left corner of the Image is assigned an index of 1. The remaining tiles are then numbered consecutively in row-major order (indices are assigned across the first row, then the second row, and so on). These tiles are regarded as *static tiles* because there is a fixed link between the tile and the image data associated with it.

A static tile set is created when the TiledLayer is instantiated; it can also be updated at any time using the `setStaticTileSet(Image, int, int)`[389] method.

In addition to the static tile set, the developer can also define several *animated tiles*. An animated tile is a virtual tile that is dynamically associated with a static tile; the appearance of an animated tile will be that of the static tile that it is currently associated with.

Animated tiles allow the developer to change the appearance of a group of cells very easily. With the group of cells all filled with the animated tile, the appearance of the entire group can be changed by simply changing the static tile associated with the animated tile. This technique is very useful for animating large repeating areas without having to explicitly change the contents of numerous cells.

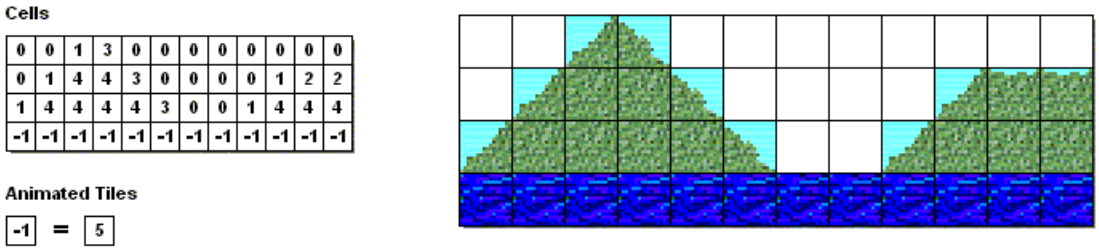
Animated tiles are created using the `createAnimatedTile(int)`[386] method, which returns the index to be used for the new animated tile. The animated tile indices are always negative and consecutive, beginning with -1. Once created, the static tile associated with an animated tile can be changed using the `setAnimatedTile(int, int)`[388] method.

Cells

The TiledLayer's grid is made up of equally sized cells; the number of rows and columns in the grid are specified in the constructor, and the physical size of the cells is defined by the size of the tiles.

The contents of each cell is specified by means of a tile index; a positive tile index refers to a static tile, and a negative tile index refers to an animated tile. A tile index of 0 indicates that the cell is empty; an empty cell is fully transparent and nothing is drawn in that area by the TiledLayer. By default, all cells contain tile index 0.

The contents of cells may be changed using `setCell(int, int, int)`[389] and `fillCells(int, int, int, int, int)`[386]. Several cells may contain the same tile; however, a single cell cannot contain more than one tile. The following example illustrates how a simple background can be created using a TiledLayer.



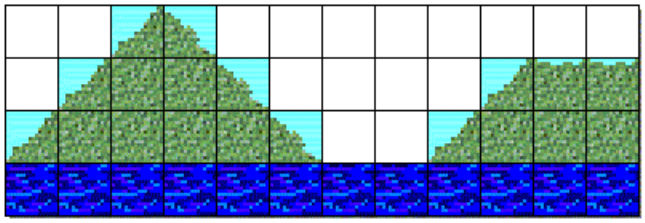
In this example, the area of water is filled with an animated tile having an index of -1, which is initially associated with static tile 5. The entire area of water may be animated by simply changing the associated static tile using `setAnimatedTile(-1, 7)`.

Cells

0	0	1	3	0	0	0	0	0	0	0	0
0	1	4	4	3	0	0	0	0	1	2	2
1	4	4	4	4	3	0	0	1	4	4	4
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Animated Tiles

-1 = 7



Rendering a TiledLayer

A TiledLayer can be rendered by manually calling its paint method; it can also be rendered automatically using a LayerManager object.

The paint method will attempt to render the entire TiledLayer subject to the clip region of the Graphics object; the upper left corner of the TiledLayer is rendered at its current (x,y) position relative to the Graphics object's origin. The rendered region may be controlled by setting the clip region of the Graphics object accordingly.

Since: MIDP 2.0

Member Summary

Constructors

```
TiledLayer(int columns, int rows,
javax.microedition.lcdui.Image image, int tileWidth, int
tileHeight)385
```

Methods

```
 int createAnimatedTile(int staticTileIndex)386
void fillCells(int col, int row, int numCols, int numRows, int
     tileIndex)386
 int getAnimatedTile(int animatedTileIndex)386
 int getCell(int col, int row)387
 int getCellHeight()387
 int getCellWidth()387
 int getColumns()387
 int getRows()388
void paint(javax.microedition.lcdui.Graphics g)388
void setAnimatedTile(int animatedTileIndex, int staticTileIndex)388
void setCell(int col, int row, int tileIndex)389
void setStaticTileSet(javax.microedition.lcdui.Image image, int
     tileWidth, int tileHeight)389
```

Inherited Member Summary

Methods inherited from class `Layer` *356*

`getHeight()` *357*, `getWidth()` *357*, `getX()` *357*, `getY()` *357*, `isVisible()` *357*, `move(int, int)` *358*, `setPosition(int, int)` *358*, `setVisible(boolean)` *358*

Methods inherited from class `Object`

`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait()`, `wait()`, `wait()`

Constructors

TiledLayer(int, int, Image, int, int)

Declaration:

```
public TiledLayer(int columns, int rows, javax.microedition.lcdui.Image 270 image,
            int tileWidth, int tileHeight)
```

Description:

Creates a new TiledLayer.

The TiledLayer's grid will be `rows` cells high and `columns` cells wide. All cells in the grid are initially empty (i.e. they contain tile index 0). The contents of the grid may be modified through the use of `setCell(int, int, int)` *389* and `fillCells(int, int, int, int, int)` *386*.

The static tile set for the TiledLayer is created from the specified Image with each tile having the dimensions of tileWidth x tileHeight. The width of the source image must be an integer multiple of the tile width, and the height of the source image must be an integer multiple of the tile height; otherwise, an IllegalArgumentException is thrown;

The entire static tile set can be changed using `setStaticTileSet(Image, int, int)` *389*. These methods should be used sparingly since they are both memory and time consuming. Where possible, animated tiles should be used instead to animate tile appearance.

Parameters:

`columns` - the width of the `TiledLayer`, expressed as a number of cells

`rows` - the height of the `TiledLayer`, expressed as a number of cells

`image` - the `Image` to use for creating the static tile set

`tileWidth` - the width in pixels of a single tile

`tileHeight` - the height in pixels of a single tile

Throws:

`NullPointerException` - if `image` is `null`

`IllegalArgumentException` - if the number of `rows` or `columns` is less than `1`

`IllegalArgumentException` - if `tileHeight` or `tileWidth` is less than `1`

`IllegalArgumentException` - if the `image` width is not an integer multiple of the `tileWidth`

`IllegalArgumentException` - if the `image` height is not an integer multiple of the `tileHeight`

Methods

createAnimatedTile(int)

Declaration:
```
public int createAnimatedTile(int staticTileIndex)
```

Description:
Creates a new animated tile and returns the index that refers to the new animated tile. It is initially associated with the specified tile index (either a static tile or 0).

The indices for animated tiles are always negative. The first animated tile shall have the index -1, the second, -2, etc.

Parameters:
`staticTileIndex` - the index of the associated tile (must be `0` or a valid static tile index)

Returns: the index of newly created animated tile

Throws:
`IndexOutOfBoundsException` - if the `staticTileIndex` is invalid

fillCells(int, int, int, int, int)

Declaration:
```
public void fillCells(int col, int row, int numCols, int numRows, int tileIndex)
```

Description:
Fills a region cells with the specific tile. The cells may be filled with a static tile index, an animated tile index, or they may be left empty (index `0`).

Parameters:
`col` - the column of top-left cell in the region

`row` - the row of top-left cell in the region

`numCols` - the number of columns in the region

`numRows` - the number of rows in the region

`tileIndex` - the Index of the tile to place in all cells in the specified region

Throws:
`IndexOutOfBoundsException` - if the rectangular region defined by the parameters extends beyond the bounds of the `TiledLayer` grid

`IllegalArgumentException` - if `numCols` is less than zero

`IllegalArgumentException` - if `numRows` is less than zero

`IndexOutOfBoundsException` - if there is no tile with index `tileIndex`

See Also: `setCell(int, int, int)`*389*, `getCell(int, int)`*387*

getAnimatedTile(int)

Declaration:
```
public int getAnimatedTile(int animatedTileIndex)
```

Description:
Gets the tile referenced by an animated tile.

Returns the tile index currently associated with the animated tile.

Parameters:
`animatedTileIndex` - the index of the animated tile

Returns: the index of the tile reference by the animated tile

Throws:
`IndexOutOfBoundsException` - if the animated tile index is invalid

See Also: `setAnimatedTile(int, int)`388

getCell(int, int)

Declaration:
```
public int getCell(int col, int row)
```

Description:
Gets the contents of a cell.

Gets the index of the static or animated tile currently displayed in a cell. The returned index will be 0 if the cell is empty.

Parameters:
`col` - the column of cell to check

`row` - the row of cell to check

Returns: the index of tile in cell

Throws:
`IndexOutOfBoundsException` - if `row` or `col` is outside the bounds of the `TiledLayer` grid

See Also: `setCell(int, int, int)`389, `fillCells(int, int, int, int, int)`386

getCellHeight()

Declaration:
```
public final int getCellHeight()
```

Description:
Gets the height of a single cell, in pixels.

Returns: the height in pixels of a single cell in the `TiledLayer` grid

getCellWidth()

Declaration:
```
public final int getCellWidth()
```

Description:
Gets the width of a single cell, in pixels.

Returns: the width in pixels of a single cell in the `TiledLayer` grid

getColumns()

Declaration:
```
public final int getColumns()
```

getRows()

Description:
Gets the number of columns in the TiledLayer grid. The overall width of the TiledLayer, in pixels, may be obtained by calling `Layer.getWidth()`[357].

Returns: the width in columns of the `TiledLayer` grid

getRows()

Declaration:
```
public final int getRows()
```

Description:
Gets the number of rows in the TiledLayer grid. The overall height of the TiledLayer, in pixels, may be obtained by calling `Layer.getHeight()`[357].

Returns: the height in rows of the `TiledLayer` grid

paint(Graphics)

Declaration:
```
public final void paint(javax.microedition.lcdui.Graphics[247] g)
```

Description:
Draws the TiledLayer. The entire TiledLayer is rendered subject to the clip region of the Graphics object. The TiledLayer's upper left corner is rendered at the TiledLayer's current position relative to the origin of the Graphics object. The current position of the TiledLayer's upper-left corner can be retrieved by calling `Layer.getX()`[357] and `Layer.getY()`[357]. The appropriate use of a clip region and/or translation allows an arbitrary region of the TiledLayer to be rendered.

If the TiledLayer's Image is mutable, the TiledLayer is rendered using the current contents of the Image.

Overrides: `paint`[358] in class `Layer`[356]

Parameters:
`g` - the graphics object to draw the `TiledLayer`

Throws:
`NullPointerException` - if `g` is `null`

setAnimatedTile(int, int)

Declaration:
```
public void setAnimatedTile(int animatedTileIndex, int staticTileIndex)
```

Description:
Associates an animated tile with the specified static tile.

Parameters:
`animatedTileIndex` - the index of the animated tile

`staticTileIndex` - the index of the associated tile (must be `0` or a valid static tile index)

Throws:
`IndexOutOfBoundsException` - if the `staticTileIndex` is invalid

`IndexOutOfBoundsException` - if the animated tile index is invalid

See Also: `getAnimatedTile(int)`[386]

setCell(int, int, int)

Declaration:
```
public void setCell(int col, int row, int tileIndex)
```

Description:
Sets the contents of a cell.

The contents may be set to a static tile index, an animated tile index, or it may be left empty (index 0)

Parameters:
`col` - the column of cell to set

`row` - the row of cell to set

`tileIndex` - the index of tile to place in cell

Throws:
`IndexOutOfBoundsException` - if there is no tile with index `tileIndex`

`IndexOutOfBoundsException` - if `row` or `col` is outside the bounds of the `TiledLayer` grid

See Also: `getCell(int, int)`[387], `fillCells(int, int, int, int, int)`[386]

setStaticTileSet(Image, int, int)

Declaration:
```
public void setStaticTileSet(javax.microedition.lcdui.Image[270] image, int tileWidth,
            int tileHeight)
```

Description:
Change the static tile set.

Replaces the current static tile set with a new static tile set. See the constructor `TiledLayer(int, int, Image, int, int)`[385] for information on how the tiles are created from the image.

If the new static tile set has as many or more tiles than the previous static tile set, the the animated tiles and cell contents will be preserve. If not, the contents of the grid will be cleared (all cells will contain index 0) and all animated tiles will be deleted.

Parameters:
`image` - the `Image` to use for creating the static tile set

`tileWidth` - the width in pixels of a single tile

`tileHeight` - the height in pixels of a single tile

Throws:
`NullPointerException` - if `image` is `null`

`IllegalArgumentException` - if `tileHeight` or `tileWidth` is less than `1`

`IllegalArgumentException` - if the `image` width is not an integer multiple of the `tileWidth`

`IllegalArgumentException` - if the `image` height is not an integer multiple of the `tileHeight`

setStaticTileSet(Image, int, int)

CHAPTER 10

Package javax.microedition.media

Description

The MIDP 2.0 Media API is a directly compatible building block of the Mobile Media API (JSR-135) specification. The use of this building block is intended for J2ME™ profiles aiming to include sound support in the specification, while maintaining upwards compatibility with the full Multimedia API. Such specification example is MIDP 2.0 (JSR-118). The development of these two interoperable API's enables seamless sound and multimedia content creation across the J2ME™ range of devices using the same API principles.

Introduction

J2ME™ devices range from cell phones with simple tone generation to PDAs and Web tablets with advanced audio and video rendering capabilities. To accommodate diverse configurations and multimedia processing capabilities, an API with a high level of abstraction is needed. The goal of the MMAPI Expert Group work has been to address this wide range of application areas, and the result of the work is a proposal of two API sets:

- Mobile Media API (JSR 135)
- MIDP 2.0 Media API

The first API is intended for J2ME™ devices with advanced sound and multimedia capabilities, including powerful mobile phones, PDAs, and set-top boxes, for example. The latter API is a directly compatible subset of the Multimedia API, and is intended for resource-constrained devices such as mass-market mobile devices (running MIDP 2.0). Furthermore, this subset API can be adopted to other J2ME™ profiles requiring sound support. In the following, a more detailed description of the background and requirements of the building block API is given.

Background of the Media API

Some J2ME™ devices are very resource constrained. It may not be feasible for a device to support a full range of multimedia types, such as video on some cell phones. As a result, not all devices are required to support the full generality of a multimedia API, such as extensibility to support custom protocols and content types.

The proposed builidng block subset API has been designed to meet the above constraints. This proposed building block fulfills the requirements set by the MIDP 2.0 Expert Group. These requirements include:

- Low footprint audio playback
- Protocol and content format agnostic
- Supports tone generation
- Supports general media flow controls: start, stop, etc.
- Supports media-specific type controls: volume etc.
- Supports capability query

This subset differs from the full Mobile Media API in the following ways:

- It is audio-only. It excludes all Controls specific to video or graphics.
- It does not support custom protocols via custom DataSources. The javax.microedition.media.protocol package (DataSource) is excluded.

It is important to note that the building block subset used in MIDP 2.0 is a proper subset of the full Mobile Media API and is fully forward compatible. In order to get the full Mobile Media API functionality into MIDP 2.0 one needs to only implement the additional classes and methods from that API.

Basic Concepts

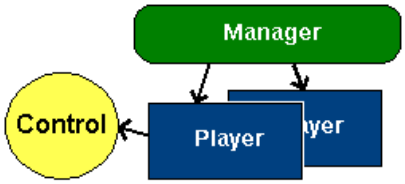
The proposed audio building block system constists of three main parts.

`Manager` `Player` `Control`

The `Manager` is the top level controller of audio resources. Applications use `Manager` to request `Players` and to query properties, supported content types and supported protocols. The manager also includes a method to play simple tones.

The `Player` plays the multimedia content. The application obtains a `Player` by giving the locator string to `Manager`.

`A Control` is an interface that is used to implement all different controls a `Player` might have. An application can query a `Player` of controls it supports and then ask for a specific `Control` e.g. `VolumeControl` to control volume.



API Details

The `createPlayer` method is the top-level entry point to the API:

```
Player Manager.createPlayer(String url)
```

The `url` fully specifies the protocol and the content of the data:

```
<protocol>:<content location>
```

The `Manager` parses the URL, recognizes the content type and creates a `Player` to handle the presentation of the data. The resulting `Player` is returned for use by the application. Connections created by `createPlayer` follow the `Generic Connection` framework rules and policies.

The `Player` provides general methods to control the data flow and presentation, for example:

```
Player.realize()
Player.prefetch()
Player.start()
```

Fine-grained control is an important feature of the API; therefore, each `Player` also provides type-specific controls with the `getControls` and `getControl` methods:

```
Control[] Player.getControls()
Control Player.getControl(int controlType)
```

Since different types of media will yield different types of controls from its corresponding `Player`, the `getControls and getControl` methods can expose features that are unique to a particular media type.

Tone Generation

Tone generation is important for games and other audio applications. On very small devices, it is particularly important since that is likely to be the only form of multimedia capabilities supported. In its simplest form, tone generation reduces to a single buzzer or some simple monophonic tone generation. The Manager class provides a top level method to handle this simple form of single tone generation:

```
Manager.playTone(int note, int duration, int volume)
```

The implementation of this method can be mapped directly to the device's hardware tone generator to provide the most responsive sound generation.

In addition, the API also provides a way to create a specific type of Player for synthesizing tone sequences:

```
Player p = Manager.createPlayer(Manager.TONE_DEVICE_LOCATOR)
```

The Player created provides a special type of Control, ToneControl which can be used for programming a tone sequence. This enables more sophisticated applications written for slightly more powerful devices.

Usage Scenarios

In this section we demonstrate how the API could be used in four common scenarios.

Scenario 1: Single-Tone Generation

```
try {
    Manager.playTone(ToneControl.C4, 5000 /* ms */, 100 /* max vol */);
} catch (MediaException e) { }
```

Scenario 2: Simple Media Playback with Looping

Notice that in MIDP 2.0 the wav format is mandatory only in a case the device supports sampled audio.

```
try {
    Player p = Manager.createPlayer("http://webserver/music.wav");
    p.setLoopCount(5);
    p.start();
} catch (IOException ioe) {
} catch (MediaException me) { }
```

Scenario 3: Playing Back from Media Stored in JAR

Notice that in MIDP 2.0 the wav format is mandatory only in a case the device supports sampled audio.

```
try {
    InputStream is = getClass().getResourceAsStream("music.wav");
    Player p = Manager.createPlayer(is, "audio/X-wav");
    p.start();
} catch (IOException ioe) {
} catch (MediaException me) { }
```

Scenario 4: Tone Sequence Generation

```
/**
 * "Mary Had A Little Lamb" has "ABAC" structure.
 * Use block to repeat "A" section.
 */
byte tempo = 30; // set tempo to 120 bpm
byte d = 8;      // eighth-note
byte C4 = ToneControl.C4;;
byte D4 = (byte)(C4 + 2); // a whole step
byte E4 = (byte)(C4 + 4); // a major third
byte G4 = (byte)(C4 + 7); // a fifth
byte rest = ToneControl.SILENCE; // rest
byte[] mySequence = {
    ToneControl.VERSION, 1,   // version 1
    ToneControl.TEMPO, tempo, // set tempo
    ToneControl.BLOCK_START, 0,   // start define "A" section
    E4,d, D4,d, C4,d, E4,d,       // content of "A" section
    E4,d, E4,d, E4,d, rest,d,
    ToneControl.BLOCK_END, 0,     // end define "A" section
    ToneControl.PLAY_BLOCK, 0,    // play "A" section
    D4,d, D4,d, D4,d, rest,d,     // play "B" section
    E4,d, G4,d, G4,d, rest,d,
    ToneControl.PLAY_BLOCK, 0,    // repeat "A" section
    D4,d, D4,d, E4,d, D4,d, C4,d  // play "C" section
};
try{
    Player p = Manager.createPlayer(Manager.TONE_DEVICE_LOCATOR);
    p.realize();
    ToneControl c = (ToneControl)p.getControl("ToneControl");
    c.setSequence(mySequence);
    p.start();
} catch (IOException ioe) {
} catch (MediaException me) { }
```

Since: MIDP 2.0

Class Summary

Interfaces

`Control`[396]	A `Control` object is used to control some media processing functions.
`Controllable`[397]	`Controllable` provides an interface for obtaining the `Control`s from an object like a `Player`.
`Player`[406]	`Player` controls the rendering of time based media data.
`PlayerListener`[416]	`PlayerListener` is the interface for receiving asynchronous events generated by `Player`s.

Classes

`Manager`[399]	`Manager` is the access point for obtaining system dependent resources such as `Player`s for multimedia processing.

Class Summary

Exceptions

MediaException[404]	A `MediaException` indicates an unexpected error condition in a method.

javax.microedition.media
Control

Declaration
```
public interface Control
```

All Known Subinterfaces: `javax.microedition.media.control.ToneControl`[422], `javax.microedition.media.control.VolumeControl`[428]

Description
A `Control` object is used to control some media processing functions. The set of operations are usually functionally related. Thus a `Control` object provides a logical grouping of media processing functions.

`Control`s are obtained from `Controllable`. The `Player` interface extends `Controllable`. Therefore a `Player` implementation can use the `Control` interface to extend its media processing functions. For example, a `Player` can expose a `VolumeControl` to allow the volume level to be set.

Multiple `Control`s can be implemented by the same object. For example, an object can implement both `VolumeControl` and `ToneControl`. In this case, the object can be used for controlling both the volume and tone generation.

The `javax.microedition.media.control` package specifies a set of pre-defined `Control`s.

See Also: `Controllable`[397], `Player`[406]

javax.microedition.media
Controllable

Declaration
```
public interface Controllable
```

All Known Subinterfaces: Player[406]

Description
`Controllable` provides an interface for obtaining the `Controls` from an object like a `Player`. It provides methods to query all the supported `Controls` and to obtain a particular `Control` based on its class name.

Member Summary	
Methods	
`Control`	`getControl(String controlType)`[397]
`Control[]`	`getControls()`[398]

Methods

getControl(String)

Declaration:
```
public javax.microedition.media.Control[396] getControl(String controlType)
```

Description:
Obtain the object that implements the specified `Control` interface.

If the specified `Control` interface is not supported then `null` is returned.

If the `Controllable` supports multiple objects that implement the same specified `Control` interface, only one of them will be returned. To obtain all the `Control`'s of that type, use the `getControls` method and check the list for the requested type.

Parameters:
`controlType` - the class name of the `Control`. The class name should be given either as the fully-qualified name of the class; or if the package of the class is not given, the package `javax.microedition.media.control` is assumed.

Returns: the object that implements the control, or `null`.

Throws:
`IllegalArgumentException` - Thrown if `controlType` is `null`.

`IllegalStateException`[37] - Thrown if `getControl` is called in a wrong state. See `Player` for more details.

getControls()

Declaration:
`public javax.microedition.media.Control[]`[396] **`getControls`**`()`

Description:
Obtain the collection of `Control`s from the object that implements this interface.

Since a single object can implement multiple `Control` interfaces, it's necessary to check each object against different `Control` types. For example:

```
Controllable controllable;
 :
Control cs[];
cs = controllable.getControls();
for (int i = 0; i < cs.length; i++) {
if (cs[i] instanceof ControlTypeA)
doSomethingA();
if (cs[i] instanceof ControlTypeB)
doSomethingB();
// etc.
}
```

The list of `Control` objects returned will not contain any duplicates. And the list will not change over time.

If no `Control` is supported, a zero length array is returned.

Returns: the collection of `Control` objects.

Throws:

`IllegalStateException`[37] - Thrown if `getControls` is called in a wrong state. See `Player` for more details.

javax.microedition.media
Manager

Declaration

```
public final class Manager

Object
  |
  +--javax.microedition.media.Manager
```

Description

`Manager` is the access point for obtaining system dependent resources such as `Players` for multimedia processing.

A Player is an object used to control and render media that is specific to the content type of the data.

`Manager` provides access to an implementation specific mechanism for constructing `Players`.

For convenience, `Manager` also provides a simplified method to generate simple tones.

Simple Tone Generation

The `playTone` function is defined to generate tones. Given the note and duration, the function will produce the specified tone.

Creating Players

`Manager` provides two methods to create a `Player` for playing back media:

- Create from a media locator.
- Create from an `InputStream`.

The `Player` returned can be used to control the presentation of the media.

Content Types

Content types identify the type of media data. They are defined to be the registered MIME types (http://www.iana.org/assignments/media-types/ (http://www.iana.org/assignments/media-types/)); plus some user-defined types that generally follow the MIME syntax (RFC 2045, RFC 2046).

For example, here are a few common content types:

1. Wave audio files: `audio/x-wav`
2. AU audio files: `audio/basic`
3. MP3 audio files: `audio/mpeg`
4. MIDI files: `audio/midi`
5. Tone sequences: `audio/x-tone-seq`

Media Locator

Media locators are specified in URI syntax (http://www.ietf.org/rfc/rfc2396.txt) which is defined in the form:

TONE_DEVICE_LOCATOR

<scheme>:<scheme-specific-part>

The “scheme” part of the locator string identifies the name of the protocol being used to deliver the data.

See Also: `Player`[406]

Member Summary

Fields

static java.lang.String	TONE_DEVICE_LOCATOR[400]

Methods

static Player	createPlayer(java.io.InputStream stream, String type)[401]
static Player	createPlayer(String locator)[401]
static java.lang.String[]	getSupportedContentTypes(String protocol)[402]
static java.lang.String[]	getSupportedProtocols(String content_type)[402]
static void	playTone(int note, int duration, int volume)[402]

Inherited Member Summary

Methods inherited from class `Object`

`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Fields

TONE_DEVICE_LOCATOR

Declaration:

```
public static final String TONE_DEVICE_LOCATOR
```

Description:

The locator to create a tone `Player` to play back tone sequences. e.g.

```
try {
    Player p = Manager.createPlayer(Manager.TONE_DEVICE_LOCATOR);
    p.realize();
    ToneControl tc = (ToneControl)p.getControl(“ToneControl”);
    tc.setSequence(mySequence);
    p.start();
} catch (IOException ioe) {
} catch (MediaException me) {}
```

If a tone sequence is not set on the tone `Player` via its `ToneControl`, the `Player` does not carry any sequence. `getDuration` returns 0 for this `Player`.

The content type of the `Player` created from this locator is `audio/x-tone-seq`.

A `Player` for this locator may not be supported for all implementations.

Value “device://tone” is assigned to `TONE_DEVICE_LOCATOR`.

Methods

createPlayer(InputStream, String)

Declaration:

```
public static javax.microedition.media.Player[406] createPlayer(java.io.InputStream stream,
            String type)
            throws IOException, MediaException
```

Description:
Create a `Player` to play back media from an `InputStream`.

The `type` argument specifies the content-type of the input media. If `null` is given, `Manager` will attempt to determine the type. However, since determining the media type is non-trivial for some media types, it may not be feasible in some cases. The `Manager` may throw a `MediaException` to indicate that.

Parameters:

`stream` - The `InputStream` that delivers the input media.

`type` - The `ContentType` of the media.

Returns: A new `Player`.

Throws:

`IllegalArgumentException` - Thrown if `stream` is `null`.

`MediaException`[404] - Thrown if a `Player` cannot be created for the given stream and type.

`java.io.IOException` - Thrown if there was a problem reading data from the `InputStream`.

`SecurityException` - Thrown if the caller does not have security permission to create the `Player`.

createPlayer(String)

Declaration:

```
public static javax.microedition.media.Player[406] createPlayer(String locator)
            throws IOException, MediaException
```

Description:
Create a `Player` from an input locator.

Parameters:

`locator` - A locator string in URI syntax that describes the media content.

Returns: A new `Player`.

Throws:

`IllegalArgumentException` - Thrown if `locator` is `null`.

`MediaException`[404] - Thrown if a `Player` cannot be created for the given locator.

`java.io.IOException` - Thrown if there was a problem connecting with the source pointed to by the `locator`.

`SecurityException` - Thrown if the caller does not have security permission to create the `Player`.

getSupportedContentTypes(String)

Declaration:
```
public static String[] getSupportedContentTypes(String protocol)
```

Description:
Return the list of supported content types for the given protocol.

See content types for the syntax of the content types returned. See protocol name for the syntax of the protocol used.

For example, if the given `protocol` is `"http"`, then the supported content types that can be played back with the `http` protocol will be returned.

If `null` is passed in as the `protocol`, all the supported content types for this implementation will be returned. The returned array must be non-empty.

If the given `protocol` is an invalid or unsupported protocol, then an empty array will be returned.

Parameters:
`protocol` - The input protocol for the supported content types.

Returns: The list of supported content types for the given protocol.

getSupportedProtocols(String)

Declaration:
```
public static String[] getSupportedProtocols(String content_type)
```

Description:
Return the list of supported protocols given the content type. The protocols are returned as strings which identify what locators can be used for creating `Player`'s.

See protocol name for the syntax of the protocols returned. See content types for the syntax of the content type used.

For example, if the given `content_type` is `"audio/x-wav"`, then the supported protocols that can be used to play back `audio/x-wav` will be returned.

If `null` is passed in as the `content_type`, all the supported protocols for this implementation will be returned. The returned array must be non-empty.

If the given `content_type` is an invalid or unsupported content type, then an empty array will be returned.

Parameters:
`content_type` - The content type for the supported protocols.

Returns: The list of supported protocols for the given content type.

playTone(int, int, int)

Declaration:
```
public static void playTone(int note, int duration, int volume)
            throws MediaException
```

Description:
Play back a tone as specified by a note and its duration. A note is given in the range of 0 to 127 inclusive. The frequency of the note can be calculated from the following formula:

```
SEMITONE_CONST = 17.31234049066755 = 1/(ln(2^(1/12)))
note = ln(freq/8.176)*SEMITONE_CONST
The musical note A = MIDI note 69 (0x45) = 440 Hz.
```

This call is a non-blocking call. Notice that this method may utilize CPU resources significantly on devices that don't have hardware support for tone generation.

Parameters:

`note` - Defines the tone of the note as specified by the above formula.

`duration` - The duration of the tone in milli-seconds. Duration must be positive.

`volume` - Audio volume range from 0 to 100. 100 represents the maximum volume at the current hardware level. Setting the volume to a value less than 0 will set the volume to 0. Setting the volume to greater than 100 will set the volume to 100.

Throws:

`IllegalArgumentException` - Thrown if the given note or duration is out of range.

`MediaException`$_{404}$ - Thrown if the tone cannot be played due to a device-related problem.

javax.microedition.media
MediaException

Declaration
```
public class MediaException extends Exception
```

```
Object
  |
  +--Throwable
        |
        +--Exception
              |
              +--javax.microedition.media.MediaException
```

Description
A `MediaException` indicates an unexpected error condition in a method.

Member Summary	
Constructors	
	`MediaException()`[404] `MediaException(String reason)`[404]

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()`
Methods inherited from class `Throwable`
`getMessage(), printStackTrace(), toString()`

Constructors

MediaException()

Declaration:
```
public MediaException()
```

Description:
Constructs a `MediaException` with `null` as its error detail message.

MediaException(String)

Declaration:
```
public MediaException(String reason)
```

Description:
Constructs a `MediaException` with the specified detail message. The error message string `s` can later be retrieved by the `Throwable.getMessage()` method of class `java.lang.Throwable`.

Parameters:
`reason` - the detail message.

javax.microedition.media
Player

Declaration
`public interface` **`Player extends`** `Controllable`[397]

All Superinterfaces: `Controllable`[397]

Description
`Player` controls the rendering of time based media data. It provides the methods to manage the `Player`'s life cycle, controls the playback progress and obtains the presentation components.

Simple Playback

A `Player` can be created from one of the `Manager`'s `createPlayer` methods. After the `Player` is created, calling `start` will start the playback as soon as possible. The method will return when the playback is started. The playback will continue in the background and will stop automatically when the end of media is reached.

Simple playback example illustrates this.

Player Life Cycle

A `Player` has five states: *UNREALIZED*, *REALIZED*, *PREFETCHED*, *STARTED*, *CLOSED*.

The purpose of these life-cycle states is to provide programmatic control over potentially time-consuming operations. For example, when a `Player` is first constructed, it's in the *UNREALIZED* state. Transitioned from *UNREALIZED* to *REALIZED*, the `Player` performs the communication necessary to locate all of the resources it needs to function (such as communicating with a server or a file system). The `realize` method allows an application to initiate this potentially time-consuming process at an appropriate time.

Typically, a `Player` moves from the *UNREALIZED* state to the *REALIZED* state, then to the *PREFETCHED* state, and finally on to the *STARTED* state.

A `Player` stops when it reaches the end of media; or when the `stop` method is invoked. When that happens, the `Player` moves from the *STARTED* state back to the *PREFETCHED* state. It is then ready to repeat the cycle.

To use a `Player`, you must set up parameters to manage its movement through these life-cycle states and then move it through the states using the `Player`'s state transition methods.

Player States

This section describes the semantics of each of the `Player` states.

UNREALIZED State

A `Player` starts in the *UNREALIZED* state. An unrealized `Player` does not have enough information to acquire all the resources it needs to function.

The following methods must not be used when the `Player` is in the *UNREALIZED* state.

- `getContentType`
- `setMediaTime`
- `getControls`
- `getControl`

An `IllegalStateException` will be thrown.

The `realize` method transitions the `Player` from the *UNREALIZED* state to the *REALIZED* state.

REALIZED State

A `Player` is in the *REALIZED* state when it has obtained the information required to acquire the media resources. Realizing a `Player` can be a resource and time consuming process. The `Player` may have to communicate with a server, read a file, or interact with a set of objects.

Although a realized `Player` does not have to acquire any resources, it is likely to have acquired all of the resources it needs except those that imply exclusive use of a scarce system resource, such as an audio device.

Normally, a `Player` moves from the *UNREALIZED* state to the *REALIZED* state. After `realize` has been invoked on a `Player`, the only way it can return to the *UNREALIZED* state is if `deallocate` is invoked before `realize` is completed. Once a `Player` reaches the *REALIZED* state, it never returns to the *UNREALIZED* state. It remains in one of four states: *REALIZED*, *PREFETCHED*, *STARTED* or *CLOSED*.

PREFETCHED State

Once realized, a `Player` may still need to perform a number of time-consuming tasks before it is ready to be started. For example, it may need to acquire scarce or exclusive resources, fill buffers with media data, or perform other start-up processing. Calling `prefetch` on the `Player` carries out these tasks.

Once a `Player` is in the *PREFETCHED* state, it may be started. Prefetching reduces the startup latency of a `Player` to the minimum possible value.

When a started `Player` stops, it returns to the *PREFETCHED* state.

STARTED State

Once prefetched, a `Player` can enter the *STARTED* state by calling the `start` method. A *STARTED* `Player` means the `Player` is running and processing data. A `Player` returns to the *PREFETCHED* state when it stops, because the `stop` method was invoked, or it has reached the end of the media.

When the `Player` moves from the *PREFETCHED* to the *STARTED* state, it posts a `STARTED` event. When it moves from the *STARTED* state to the *PREFETCHED* state, it posts a `STOPPED`, `END_OF_MEDIA` event depending on the reason it stopped.

The following method must not be used when the `Player` is in the *STARTED* state:

- `setLoopCount`

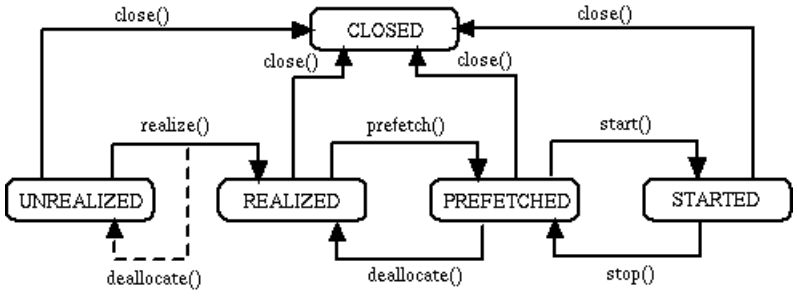
An `IllegalStateException` will be thrown.

MediaException(String)

CLOSED state

Calling `close` on the `Player` puts it in the *CLOSED* state. In the *CLOSED* state, the `Player` has released most of its resources and must not be used again.

The `Player`'s five states and the state transition methods are summarized in the following diagram:



Player Events

`Player` events asynchronously deliver information about the `Player`'s state changes and other relevant information from the `Player`'s `Controls`.

To receive events, an object must implement the `PlayerListener` interface and use the `addPlayerListener` method to register its interest in a `Player`'s events. All `Player` events are posted to each registered listener.

The events are guaranteed to be delivered in the order that the actions representing the events occur. For example, if a `Player` stops shortly after it starts because it is playing back a very short media file, the `STARTED` event must always preceed the `END_OF_MEDIA` event.

An `ERROR` event may be sent any time an irrecoverable error has occured. When that happens, the `Player` is in the *CLOSED* state.

The `Player` event mechanism is extensible and some `Players` define events other than the ones described here. For a list of pre-defined player events, check the `PlayerListener` interface.

Managing the Resources Used by a Player

The `prefetch` method is used to acquire scarce or exclusive resources such as the audio device. Conversely, the `deallocate` method is used to release the scarce or exclusive resources. By using these two methods, an application can programmatically manage the `Player`'s resources.

For example, in an implementation with an exclusive audio device, to alternate the audio playback of multiple `Players`, an application can selectively deallocate and prefetch individual `Players`.

Player's Controls

`Player` implements `Controllable` which provides extra controls via some type-specific `Control` interfaces. `getControl` and `getControls` cannot be called when the `Player` is in the *UNREALIZED* or *CLOSED* state. An `IllegalStateException` will be thrown.

Simple Playback Example

```
try {
    Player p = Manager.createPlayer("http://abc.wav");
    p.start();
} catch (MediaException pe) {
} catch (IOException ioe) {
}
```

Member Summary

Fields

static int	CLOSED[409]
static int	PREFETCHED[410]
static int	REALIZED[410]
static int	STARTED[410]
static long	TIME_UNKNOWN[410]
static int	UNREALIZED[410]

Methods

void	addPlayerListener(PlayerListener playerListener)[411]
void	close()[411]
void	deallocate()[411]
java.lang.String	getContentType()[411]
long	getDuration()[412]
long	getMediaTime()[412]
int	getState()[412]
void	prefetch()[412]
void	realize()[413]
void	removePlayerListener(PlayerListener playerListener)[413]
void	setLoopCount(int count)[414]
long	setMediaTime(long now)[414]
void	start()[415]
void	stop()[415]

Inherited Member Summary

Methods inherited from interface Controllable[397]

getControl(String)[397], getControls()[398]

Fields

CLOSED

Declaration:
public static final int **CLOSED**

Description:
The state of the `Player` indicating that the `Player` is closed.

Value 0 is assigned to `CLOSED`.

PREFETCHED

Declaration:
```
public static final int PREFETCHED
```

Description:
The state of the `Player` indicating that it has acquired all the resources to begin playing.

Value 300 is assigned to `PREFETCHED`.

REALIZED

Declaration:
```
public static final int REALIZED
```

Description:
The state of the `Player` indicating that it has acquired the required information but not the resources to function.

Value 200 is assigned to `REALIZED`.

STARTED

Declaration:
```
public static final int STARTED
```

Description:
The state of the `Player` indicating that the `Player` has already started.

Value 400 is assigned to `STARTED`.

TIME_UNKNOWN

Declaration:
```
public static final long TIME_UNKNOWN
```

Description:
The returned value indicating that the requested time is unknown.

Value -1 is assigned to `TIME_UNKNOWN`.

UNREALIZED

Declaration:
```
public static final int UNREALIZED
```

Description:
The state of the `Player` indicating that it has not acquired the required information and resources to function.

Value 100 is assigned to `UNREALIZED`.

Methods

addPlayerListener(PlayerListener)

Declaration:
`public void addPlayerListener(javax.microedition.media.PlayerListener`[416] `playerListener)`

Description:
Add a player listener for this player.

Parameters:
`playerListener` - the listener to add. If `null` is used, the request will be ignored.

Throws:
`IllegalStateException`[37] - Thrown if the `Player` is in the *CLOSED* state.

See Also: `removePlayerListener(PlayerListener)`[413]

close()

Declaration:
`public void close()`

Description:
Close the `Player` and release its resources.

When the method returns, the `Player` is in the *CLOSED* state and can no longer be used. A `CLOSED` event will be delivered to the registered `PlayerListener`s.

If `close` is called on a closed `Player` the request is ignored.

deallocate()

Declaration:
`public void deallocate()`

Description:
Release the scarce or exclusive resources like the audio device acquired by the `Player`.

When `deallocate` returns, the `Player` is in the *UNREALIZED* or *REALIZED* state.

If the `Player` is blocked at the `realize` call while realizing, calling `deallocate` unblocks the `realize` call and returns the `Player` to the *UNREALIZED* state. Otherwise, calling `deallocate` returns the `Player` to the *REALIZED* state.

If `deallocate` is called when the `Player` is in the *UNREALIZED* or *REALIZED* state, the request is ignored.

If the `Player` is `STARTED` when `deallocate` is called, `deallocate` will implicitly call `stop` on the `Player`.

Throws:
`IllegalStateException`[37] - Thrown if the `Player` is in the *CLOSED* state.

getContentType()

Declaration:
`public String getContentType()`

Description:
Get the content type of the media that's being played back by this `Player`.

See content type for the syntax of the content type returned.

Returns: The content type being played back by this `Player`.

Throws:
`IllegalStateException`[37] - Thrown if the `Player` is in the *UNREALIZED* or *CLOSED* state.

getDuration()

Declaration:
```
public long getDuration()
```

Description:
Get the duration of the media. The value returned is the media's duration when played at the default rate. If the duration cannot be determined (for example, the `Player` is presenting live media) `getDuration` returns `TIME_UNKNOWN`.

Returns: The duration in microseconds or `TIME_UNKNOWN`.

Throws:
`IllegalStateException`[37] - Thrown if the `Player` is in the *CLOSED* state.

getMediaTime()

Declaration:
```
public long getMediaTime()
```

Description:
Gets this `Player`'s current *media time*. If the *media time* cannot be determined, `getMediaTime` returns `TIME_UNKNOWN`.

Returns: The current *media time* in microseconds or `TIME_UNKNOWN`.

Throws:
`IllegalStateException`[37] - Thrown if the `Player` is in the *CLOSED* state.

See Also: `setMediaTime(long)`[414]

getState()

Declaration:
```
public int getState()
```

Description:
Gets the current state of this `Player`. The possible states are: *UNREALIZED*, *REALIZED*, *PREFETCHED*, *STARTED*, *CLOSED*.

Returns: The `Player`'s current state.

prefetch()

Declaration:
```
public void prefetch()
            throws MediaException
```

Description:
Acquires the scarce and exclusive resources and processes as much data as necessary to reduce the start latency.

When `prefetch` completes successfully, the `Player` is in the *PREFETCHED* state.

If `prefetch` is called when the `Player` is in the *UNREALIZED* state, it will implicitly call `realize`.

If `prefetch` is called when the `Player` is already in the *PREFETCHED* state, the `Player` may still process data necessary to reduce the start latency. This is to guarantee that start latency can be maintained at a minimum.

If `prefetch` is called when the `Player` is in the *STARTED* state, the request will be ignored.

If the `Player` cannot obtain all of the resources it needs, it throws a `MediaException`. When that happens, the `Player` will not be able to start. However, `prefetch` may be called again when the needed resource is later released perhaps by another `Player` or application.

Throws:

`IllegalStateException`[37] - Thrown if the `Player` is in the *CLOSED* state.

`MediaException`[404] - Thrown if the `Player` cannot be prefetched.

`SecurityException` - Thrown if the caller does not have security permission to prefetch the `Player`.

realize()

Declaration:

```
public void realize()
            throws MediaException
```

Description:
Constructs portions of the `Player` without acquiring the scarce and exclusive resources. This may include examining media data and may take some time to complete.

When `realize` completes successfully, the `Player` is in the *REALIZED* state.

If `realize` is called when the `Player` is in the *REALIZED*, *PREFETCHTED* or *STARTED* state, the request will be ignored.

Throws:

`IllegalStateException`[37] - Thrown if the `Player` is in the *CLOSED* state.

`MediaException`[404] - Thrown if the `Player` cannot be realized.

`SecurityException` - Thrown if the caller does not have security permission to realize the `Player`.

removePlayerListener(PlayerListener)

Declaration:

```
public void removePlayerListener(javax.microedition.media.PlayerListener[416]
            playerListener)
```

Description:
Remove a player listener for this player.

Parameters:

`playerListener` - the listener to remove. If `null` is used or the given `playerListener` is not a listener for this `Player`, the request will be ignored.

Throws:

`IllegalStateException`[37] - Thrown if the `Player` is in the *CLOSED* state.

See Also: `addPlayerListener(PlayerListener)`[411]

setLoopCount(int)

Declaration:

```
public void setLoopCount(int count)
```

Description:

Set the number of times the `Player` will loop and play the content.

By default, the loop count is one. That is, once started, the `Player` will start playing from the current media time to the end of media once.

If the loop count is set to N where N is bigger than one, starting the `Player` will start playing the content from the current media time to the end of media. It will then loop back to the beginning of the content (media time zero) and play till the end of the media. The number of times it will loop to the beginning and play to the end of media will be N-1.

Setting the loop count to 0 is invalid. An `IllegalArgumentException` will be thrown.

Setting the loop count to -1 will loop and play the content indefinitely.

If the `Player` is stopped before the preset loop count is reached either because `stop` is called, calling `start` again will resume the looping playback from where it was stopped until it fully reaches the preset loop count.

An *END_OF_MEDIA* event will be posted every time the `Player` reaches the end of media. If the `Player` loops back to the beginning and starts playing again because it has not completed the loop count, a *STARTED* event will be posted.

Parameters:

`count` - indicates the number of times the content will be played. 1 is the default. 0 is invalid. -1 indicates looping indefintely.

Throws:

`IllegalArgumentException` - Thrown if the given count is invalid.

`IllegalStateException`[37] - Thrown if the `Player` is in the *STARTED* or *CLOSED* state.

setMediaTime(long)

Declaration:

```
public long setMediaTime(long now)
            throws MediaException
```

Description:

Sets the `Player`'s *media time*.

For some media types, setting the media time may not be very accurate. The returned value will indicate the actual media time set.

Setting the media time to negative values will effectively set the media time to zero. Setting the media time to beyond the duration of the media will set the time to the end of media.

There are some media types that cannot support the setting of media time. Calling `setMediaTime` will throw a `MediaException` in those cases.

Parameters:
`now` - The new media time in microseconds.

Returns: The actual media time set in microseconds.

Throws:
`IllegalStateException`[37] - Thrown if the `Player` is in the *UNREALIZED* or *CLOSED* state.

`MediaException`[404] - Thrown if the media time cannot be set.

See Also: `getMediaTime()`[412]

start()

Declaration:
```
public void start()
            throws MediaException
```

Description:
Starts the `Player` as soon as possible. If the `Player` was previously stopped by calling `stop`, it will resume playback from where it was previously stopped. If the `Player` has reached the end of media, calling `start` will automatically start the playback from the start of the media.

When `start` returns successfully, the `Player` must have been started and a `STARTED` event will be delivered to the registered `PlayerListener`s. However, the `Player` is not guaranteed to be in the *STARTED* state. The `Player` may have already stopped (in the *PREFETCHED* state) because the media has 0 or a very short duration.

If `start` is called when the `Player` is in the *UNREALIZED* or *REALIZED* state, it will implicitly call `prefetch`.

If `start` is called when the `Player` is in the *STARTED* state, the request will be ignored.

Throws:
`IllegalStateException`[37] - Thrown if the `Player` is in the *CLOSED* state.

`MediaException`[404] - Thrown if the `Player` cannot be started.

`SecurityException` - Thrown if the caller does not have security permission to start the `Player`.

stop()

Declaration:
```
public void stop()
            throws MediaException
```

Description:
Stops the `Player`. It will pause the playback at the current media time.

When `stop` returns, the `Player` is in the *PREFETCHED* state. A `STOPPED` event will be delivered to the registered `PlayerListener`s.

If `stop` is called on a stopped `Player`, the request is ignored.

Throws:
`IllegalStateException`[37] - Thrown if the `Player` is in the *CLOSED* state.

`MediaException`[404] - Thrown if the `Player` cannot be stopped.

javax.microedition.media
PlayerListener

Declaration
`public interface` **`PlayerListener`**

Description
`PlayerListener` is the interface for receiving asynchronous events generated by `Players`. Applications may implement this interface and register their implementations with the `addPlayerListener` method in `Player`.

A number of standard `Player` events are defined here in this interface. Event types are defined as strings to support extensibility as different implementations may introduce proprietary events by adding new event types. To avoid name conflicts, proprietary events should be named with the "reverse-domainname" convention. For example, a company named "mycompany" should name its proprietary event names with strings like `"com.mycompany.myEvent"` etc.

Applications that rely on proprietary events may not function properly across different implementations. In order to make the applications that use those events to behave well in environments that don't implement them, `String.equals()` should be used to check the event.

Code fragment for catching standard events in playerUpdate()

```
if (eventType == PlayerListener.STARTED) {...}
```

Code fragment for catching proprietary events in playerUpdate()

```
if (eventType.equals("com.company.myEvent")) {...}
```

See Also: `Player` 406

Member Summary	
Fields	
static	CLOSED 417
java.lang.String	
static	DEVICE_AVAILABLE 417
java.lang.String	
static	DEVICE_UNAVAILABLE 417
java.lang.String	
static	DURATION_UPDATED 418
java.lang.String	
static	END_OF_MEDIA 418
java.lang.String	
static	ERROR 418
java.lang.String	
static	STARTED 418
java.lang.String	
static	STOPPED 418
java.lang.String	

Member Summary	
static java.lang.String	VOLUME_CHANGED[419]
Methods	
void	playerUpdate(Player player, String event, Object eventData)[419]

Fields

CLOSED

Declaration:
`public static final String CLOSED`

Description:
Posted when a `Player` is closed. When this event is received, the `eventData` parameter is null.

Value `closed` is assigned to `CLOSED`.

DEVICE_AVAILABLE

Declaration:
`public static final String DEVICE_AVAILABLE`

Description:
Posted when the system or another higher priority application has released an exclusive device which is now available to the `Player`.

The `Player` will be in the *REALIZED* state when this event is received. The application may acquire the device with the `prefetch` or `start` method.

A `DEVICE_UNAVAILABLE` event must preceed this event.

The `eventData` parameter is a `String` specifying the name of the device.

Value `deviceAvailable` is assigned to `DEVICE_AVAILABLE`.

DEVICE_UNAVAILABLE

Declaration:
`public static final String DEVICE_UNAVAILABLE`

Description:
Posted when the system or another higher priority application has temporarily taken control of an exclusive device which was previously available to the `Player`.

The `Player` will be in the *REALIZED* state when this event is received.

This event must be followed by either a `DEVICE_AVAILABLE` event when the device becomes available again, or an `ERROR` event if the device becomes permanently unavailable.

The `eventData` parameter is a `String` specifying the name of the device.

Value `deviceUnavailable` is assigned to `DEVICE_UNAVAILABLE`.

DURATION_UPDATED

Declaration:
`public static final String DURATION_UPDATED`

Description:
Posted when the duration of a `Player` is updated. This happens for some media types where the duration cannot be derived ahead of time. It can only be derived after the media is played for a period of time —- for example, when it reaches a key frame with duration info; or when it reaches the end of media.

When this event is received, the `eventData` parameter will be a `Long` object designating the duration of the media.

Value `durationUpdated` is assigned to `DURATION_UPDATED`.

END_OF_MEDIA

Declaration:
`public static final String END_OF_MEDIA`

Description:
Posted when a `Player` has reached the end of the media. When this event is received, the `eventData` parameter will be a `Long` object designating the media time when the `Player` reached end of media and stopped.

Value `endOfMedia` is assigned to `END_OF_MEDIA`.

ERROR

Declaration:
`public static final String ERROR`

Description:
Posted when an error had occurred. When this event is received, the `eventData` parameter will be a `String` object specifying the error message.

Value `error` is assigned to `ERROR`.

STARTED

Declaration:
`public static final String STARTED`

Description:
Posted when a `Player` is started. When this event is received, the `eventData` parameter will be a `Long` object designating the media time when the `Player` is started.

Value `started` is assigned to `STARTED`.

STOPPED

Declaration:
`public static final String STOPPED`

Description:
Posted when a `Player` stops in response to the `stop` method call. When this event is received, the `eventData` parameter will be a `Long` object designating the media time when the `Player` stopped.

Value `stopped` is assigned to `STOPPED`.

VOLUME_CHANGED

Declaration:
```
public static final String VOLUME_CHANGED
```

Description:
Posted when the volume of an audio device is changed. When this event is received, the `eventData` parameter will be a `VolumeControl` object. The new volume can be queried from the `VolumeControl`.

Value `volumeChanged` is assigned to `VOLUME_CHANGED`.

Methods

playerUpdate(Player, String, Object)

Declaration:
```
public void playerUpdate(javax.microedition.media.Player[406] player, String event,
            Object eventData)
```

Description:
This method is called to deliver an event to a registered listener when a `Player` event is observed.

Parameters:

`player` - The player which generated the event.

`event` - The event generated as defined by the enumerated types.

`eventData` - The associated event data.

playerUpdate(Player, String, Object)

CHAPTER 11

Package javax.microedition.media.control

Description

This package defines the specific `Control` types that can be used with a `Player`.

Since: MIDP 2.0

Class Summary	
Interfaces	
`ToneControl` 422	`ToneControl` is the interface to enable playback of a user-defined monotonic tone sequence.
`VolumeControl` 428	`VolumeControl` is an interface for manipulating the audio volume of a `Player`.

javax.microedition.media.control
ToneControl

Declaration

```
public interface ToneControl extends javax.microedition.media.Control396
```

All Superinterfaces: `javax.microedition.media.Control`[396]

Description

`ToneControl` is the interface to enable playback of a user-defined monotonic tone sequence.

A tone sequence is specified as a list of tone-duration pairs and user-defined sequence blocks. The list is packaged as an array of bytes. The `setSequence` method is used to input the sequence to the `ToneControl`.

The syntax of a tone sequence is described in Augmented BNF (http://www.ietf.org/rfc/rfc2234) notations:

```
sequence              = version *1tempo_definition *1resolution_definition
           *block_definition 1*sequence_event
version               = VERSION version_number
VERSION               = byte-value
version_number        = 1 ; version # 1
tempo_definition      = TEMPO tempo_modifier
TEMPO                 = byte-value
tempo_modifier        = byte-value
             ; multiply by 4 to get the tempo (in bpm) used
             ; in the sequence.

resolution_definition = RESOLUTION resolution_unit
RESOLUTION            = byte-value
resolution_unit       = byte-value
block_definition      = BLOCK_START block_number
                           1*sequence_event
                        BLOCK_END block_number
BLOCK_START           = byte-value
BLOCK_END             = byte-value
block_number          = byte-value
             ; block_number specified in BLOCK_END has to be the
             ; same as the one in BLOCK_START
sequence_event        = tone_event / block_event /
                          volume_event / repeat_event

tone_event            = note duration
note                  = byte-value ; note to be played
duration              = byte-value ; duration of the note
block_event           = PLAY_BLOCK block_number
PLAY_BLOCK            = byte-value
block_number          = byte-value
             ; block_number must be previously defined
             ; by a full block_definition
volume_event          = SET_VOLUME volume
SET_VOLUME            = byte-value
volume                = byte-value ; new volume
repeat_event          = REPEAT multiplier tone_event
REPEAT                = byte-value
multiplier            = byte-value
             ; number of times to repeat a tone
byte-value            = -128 - 127
             ; the value of each constant and additional
             ; constraints on each parameter are specified below.
```

`VERSION`, `TEMPO`, `RESOLUTION`, `BLOCK_START`, `BLOCK_END`, `PLAY_BLOCK` `SET_VOLUME` `REPEAT` are pre-defined constants.

Following table shows the valid range of the parameters:

Parameter	Valid Range	Effective Range	Default
`tempo_modifier`	5<= `tempo_modifier` <= 127	20bpm to 508bpm	120bpm
`resolution_unit`	1<= `resolution_unit` <= 127	1/1 note to 1/127 note	1/64 note
`block_number`	0<= `block_number` <= 127	-	-
`note`	0<= `note` <= 127 or SILENCE	C-1 to G9 or rest	-
`duration`	1<= `duration` <= 127	-	-
`volume`	0<= `volume` <= 100	0% to 100% volume	100%
`multiplier`	2<= `multiplier` <= 127	-	-

The frequency of the note can be calculated from the following formula:

```
SEMITONE_CONST = 17.31234049066755 = 1/(ln(2^(1/12)))
note = ln(freq/8.176)*SEMITONE_CONST
```

The musical note A = note 69 (0x45) = 440 Hz.
Middle C (C4) and SILENCE are defined as constants.

The duration of each tone is measured in units of 1/resolution notes and tempo is specified in beats/minute, where 1 beat = 1/4 note. Because the range of positive values of `byte` is only 1 - 127, the tempo is formed by multiplying the tempo modifier by 4. Very slow tempos are excluded so range of tempo modifiers is 5 - 127 providing an effective range of 20 - 508 bpm.

To compute the effective duration in milliseconds for a tone, the following formula can be used:

```
duration * 60 * 1000 * 4 / (resolution * tempo)
```

The following table lists some common durations in musical notes:

Note Length	Duration, Resolution=64	Duration, Resolution=96
1/1	64	96
1/4	16	24
1/4 dotted	24	36
1/8	8	12
1/8 triplets	-	8
4/1	REPEAT 4 <note> 64	REPEAT 4 <note> 96

Example

```
// “Mary Had A Little Lamb” has “ABAC” structure.
// Use block to repeat “A” section.
byte tempo = 30; // set tempo to 120 bpm
byte d = 8;      // eighth-note
byte C4 = ToneControl.C4;;
byte D4 = (byte)(C4 + 2); // a whole step
byte E4 = (byte)(C4 + 4); // a major third
byte G4 = (byte)(C4 + 7); // a fifth
byte rest = ToneControl.SILENCE; // rest
byte[] mySequence = {
   ToneControl.VERSION, 1,   // version 1
   ToneControl.TEMPO, tempo, // set tempo
   ToneControl.BLOCK_START, 0,   // start define “A” section
   E4,d, D4,d, C4,d, E4,d,       // content of “A” section
   E4,d, E4,d, E4,d, rest,d,
   ToneControl.BLOCK_END, 0,     // end define “A” section
   ToneControl.PLAY_BLOCK, 0,    // play “A” section
   D4,d, D4,d, D4,d, rest,d,     // play “B” section
   E4,d, G4,d, G4,d, rest,d,
   ToneControl.PLAY_BLOCK, 0,    // repeat “A” section
   D4,d, D4,d, E4,d, D4,d, C4,d  // play “C” section
};
```

```
try{
   Player p = Manager.createPlayer(Manager.TONE_DEVICE_LOCATOR);
   p.realize();
   ToneControl c = (ToneControl)p.getControl("ToneControl");
   c.setSequence(mySequence);
   p.start();
} catch (IOException ioe) {
} catch (MediaException me) { }
```

Member Summary	
Fields	
static byte	BLOCK_END 425
static byte	BLOCK_START 425
static byte	C4 425
static byte	PLAY_BLOCK 426
static byte	REPEAT 426
static byte	RESOLUTION 426
static byte	SET_VOLUME 426
static byte	SILENCE 426
static byte	TEMPO 426
static byte	VERSION 427
Methods	
void	setSequence(byte[] sequence) 427

Fields

BLOCK_END

Declaration:
```
public static final byte BLOCK_END
```

Description:
Defines an ending point for a block.

Value -6 is assigned to BLOCK_END.

BLOCK_START

Declaration:
```
public static final byte BLOCK_START
```

Description:
Defines a starting point for a block.

Value -5 is assigned to BLOCK_START.

C4

Declaration:
```
public static final byte C4
```

PLAY_BLOCK

Description:
Middle C.

Value 60 is assigned to `C4`.

PLAY_BLOCK

Declaration:
`public static final byte PLAY_BLOCK`

Description:
Play a defined block.

Value -7 is assigned to `PLAY_BLOCK`.

REPEAT

Declaration:
`public static final byte REPEAT`

Description:
The REPEAT event tag.

Value -9 is assigned to `REPEAT`.

RESOLUTION

Declaration:
`public static final byte RESOLUTION`

Description:
The RESOLUTION event tag.

Value -4 is assigned to `RESOLUTION`.

SET_VOLUME

Declaration:
`public static final byte SET_VOLUME`

Description:
The SET_VOLUME event tag.

Value -8 is assigned to `SET_VOLUME`.

SILENCE

Declaration:
`public static final byte SILENCE`

Description:
Silence.

Value -1 is assigned to `SILENCE`.

TEMPO

Declaration:
`public static final byte TEMPO`

Description:
The TEMPO event tag.

Value -3 is assigned to `TEMPO`.

VERSION

Declaration:
```
public static final byte VERSION
```

Description:
The VERSION attribute tag.

Value -2 is assigned to `VERSION`.

Methods

setSequence(byte[])

Declaration:
```
public void setSequence(byte[] sequence)
```

Description:
Sets the tone sequence.

Parameters:
`sequence` - The sequence to set.

Throws:
`IllegalArgumentException` - Thrown if the sequence is `null` or invalid.

`IllegalStateException`[37] - Thrown if the `Player` that this control belongs to is in the *PREFETCHED* or *STARTED* state.

javax.microedition.media.control
VolumeControl

Declaration
`public interface` **`VolumeControl extends`** **`javax.microedition.media.Control`**[396]

All Superinterfaces: `javax.microedition.media.Control`[396]

Description
`VolumeControl` is an interface for manipulating the audio volume of a `Player`.

Volume Settings
This interface allows the output volume to be specified using an integer value that varies between 0 and 100.

Specifying Volume in the Level Scale
The level scale specifies volume in a linear scale. It ranges from 0 to 100, where 0 represents silence and 100 represents the highest volume. The mapping for producing a linear multiplicative value is implementation dependent.

Mute
Setting mute on or off doesn't change the volume level returned by getLevel. If mute is `true`, no audio signal is produced by this `Player`; if mute is `false` an audio signal is produced and the volume is restored.

Volume Change Events
When the state of the `VolumeControl` changes, a `VOLUME_CHANGED` event is delivered through the `PlayerListener`.

See Also: `javax.microedition.media.Control`[396], `javax.microedition.media.Player`[406], `javax.microedition.media.PlayerListener`[416]

Member Summary

Methods

`int`	`getLevel()`[429]
`boolean`	`isMuted()`[429]
`int`	`setLevel(int level)`[429]
`void`	`setMute(boolean mute)`[429]

Methods

getLevel()

Declaration:
```
public int getLevel()
```

Description:
Get the current volume level set.

`getLevel` may return `-1` if and only if the `Player` is in the *REALIZED* state (the audio device has not been initialized) and `setLevel` has not yet been called.

Returns: The current volume level or `-1`.

See Also: `setLevel(int)` 429

isMuted()

Declaration:
```
public boolean isMuted()
```

Description:
Get the mute state of the signal associated with this `VolumeControl`.

Returns: The mute state.

See Also: `setMute(boolean)` 429

setLevel(int)

Declaration:
```
public int setLevel(int level)
```

Description:
Set the volume using a linear point scale with values between 0 and 100.
0 is silence; 100 is the loudest useful level that this `VolumeControl` supports. If the given level is less than 0 or greater than 100, the level will be set to 0 or 100 respectively.
When `setLevel` results in a change in the volume level, a `VOLUME_CHANGED` event will be delivered through the `PlayerListener`.

Parameters:
`level` - The new volume specified in the level scale.

Returns: The level that was actually set.

See Also: `getLevel()` 429

setMute(boolean)

Declaration:
```
public void setMute(boolean mute)
```

Description:
Mute or unmute the `Player` associated with this `VolumeControl`.

Calling `setMute(true)` on the `Player` that is already muted is ignored, as is calling `setMute(false)` on the `Player` that is not currently muted. Setting mute on or off doesn't change the volume level returned by getLevel.

setMute(boolean)

When `setMute` results in a change in the muted state, a `VOLUME_CHANGED` event will be delivered through the `PlayerListener`.

Parameters:

`mute` - Specify `true` to mute the signal, `false` to unmute the signal.

See Also: `isMuted()`*429*

CHAPTER 12

Package javax.microedition.midlet

Description

The MIDlet package defines Mobile Information Device Profile applications and the interactions between the application and the environment in which the application runs. An application of the Mobile Information Device Profile is a `MIDlet`.

Applications

The MIDP defines an application model to allow the limited resources of the device to be shared by multiple MIDP applications, or MIDlets. It defines what a MIDlet is, how it is packaged, what runtime environment is available to the MIDlet, and how it should be behave so that the device can manage its resources. The application model defines how multiple MIDlets forming a suite can be packaged together and share resources within the context of a single Java Virtual Machine. Sharing is feasible with the limited resources and security framework of the device since they are required to share class files and to be subject to a single set of policies and controls.

MIDP MIDlet Suite

A MIDP application MUST use only functionality specified by the MIDP specification as it is developed, tested, deployed, and run.

The elements of a MIDlet suite are:

- Runtime execution environment
- MIDlet suite packaging
- Application descriptor
- Application lifecycle

Each device is presumed to implement the functions required by its users to install, select, run, and remove MIDlets. The term application management software is used to refer collectively to these device specific functions. The application management software provides an environment in which the MIDlet is installed, started, stopped, and uninstalled. It is responsible for handling errors during the installation, execution, and removal of MIDlet suites and interacting with the user as needed. It provides to the MIDlet(s) the Java runtime environment required by the MIDP Specification.

One or more MIDlets MAY be packaged in a single JAR file. Each MIDlet consists of a class that extends the `MIDlet` class and other classes as may be needed by the MIDlet. The manifest in the JAR file contains attributes that are used during installation and execution of MIDlets. The MIDlet is the entity that is launched by the application management software. When a MIDlet suite is invoked, a Java Virtual Machine is needed on which the classes can be executed. A new instance of the MIDlet is created by the application management software and used to direct the MIDlet to start, pause, and destroy itself.

Sharing of data and other information between MIDlets is controlled by the individual APIs and their implementations. For example, the Record Management System API specifies the methods that are used when the record stores associated with a MIDlet suite are shared among MIDlets.

MIDlet Suite Security

The MIDP 1.0 specification constrained each MIDlet suite to operate in a sandbox wherein all of the APIs available to the MIDlets would prevent access to sensitive functions of the device. That sandbox concept is used in this specification and all untrusted MIDlet suites are subject to its limitations. Every implementation of this specification MUST support running untrusted MIDlet suites.

MIDP 2.0 introduces the concept of trusted applications that may be permitted to use APIs that are considered sensitive and are restricted. If and when a device determines that a MIDlet suite can be trusted the device allows access as indicated by the policy. Security for MIDP Applications section describes the concepts and capabilities of untrusted and trusted applications.

MIDP Execution Environment

The MIDP defines the execution environment provided to MIDlets. The execution environment is shared by all MIDlets within a MIDlet suite, and any MIDlet can interact with other MIDlets packaged together. The application management software initiates the applications and makes the following available to the MIDlet:

- Classes and native code that implement the CLDC, including a Java Virtual Machine
- Classes and native code that implement the MIDP runtime
- All classes from a single JAR file for execution
- Non-class files from a single JAR file as resources
- Contents of the descriptor file, when it is present
- Any other APIs available on the device such as implementations of additional JSRs, Licensee Open Classes, Optional Packages, etc.

The CLDC and Java Virtual Machine provide multi-threading, locking and synchronization, the execution of byte codes, dispatching of methods, etc. A single VM is the scope of all policy, naming, and resource management. If a device supports multiple VMs, each may have its own scope, naming, and resource management policies. MIDlet Suites MUST NOT contain classes that are in packages defined by the CLDC or MIDP.

The MIDP provides the classes that implement the MIDP APIs. The implementation MUST ensure that the application programmer cannot override, modify, or add any classes to these protected system packages.

A single JAR file contains all of the MIDlet's classes. The MIDlet may load and invoke methods from any class in the JAR file, in the MIDP, or in the CLDC. All of the classes within these three scopes are shared in the execution environment of the MIDlets from the JAR file. All states accessible via those classes are available to any Java class running on behalf of the MIDlet. There is a single space containing the objects of all MIDlets, MIDP, and CLDC in use by the MIDlet suite. The usual Java locking and synchronization primitives SHOULD be used when necessary to avoid concurrency problems. Each library will specify how it handles concurrency and how the MIDlet should use it to run safely in a multi-threaded environment.

The class files of the MIDlet are only available for execution and can neither be read as resources nor extracted for re-use. The implementation of the CLDC may store and interpret the contents of the JAR file in any manner suitable.

The files from the JAR file that are not Java class files are made available using `java.lang.Class.getResourceAsStream`. For example, the manifest would be available in this manner.

The contents of the MIDlet descriptor file, when it is present, are made available via the `javax.microedition.midlet.MIDlet.getAppProperty` method.

MIDlet Suite Packaging
One or more MIDlets are packaged in a single JAR file that includes:

- A manifest describing the contents
- Java classes for the MIDlet(s) and classes shared by the MIDlets
- Resource files used by the MIDlet(s)

The developer is responsible for creating and distributing the components of the JAR file as appropriate for the target user, device, network, locale, and jurisdiction. For example, for a particular locale, the resource files would be tailored to contain the strings and images needed for that locale.

The JAR manifest defines attributes that are used by the application management software to identify and install the MIDlet suite and as defaults for attributes not found in the application descriptor. The attributes are defined for use in both the manifest and the optional application descriptor.

The predefined attributes listed below allow the application management software to identify, retrieve, install, and invoke the MIDlet.

MIDlet Attributes

Attribute Name	Attribute Description
MIDlet-Name	The name of the MIDlet suite that identifies the MIDlets to the user.
MIDlet-Version	The version number of the MIDlet suite. Version numbers are formatted so they can be used by the application management software for install and upgrade uses, as well as communication with the user.
MIDlet-Vendor	The organization that provides the MIDlet suite.
MIDlet-Icon	The case-sensitive absolute name of a PNG file within the JAR used to represent the MIDlet suite. It SHOULD be used when the Application Management Software displays an icon to identify the suite.
MIDlet-Description	The description of the MIDlet suite.
MIDlet-Info-URL	A URL for information further describing the MIDlet suite. The syntax and meaning MUST conform to RFC2396 and RFCs that define each scheme.
MIDlet-<n>	The name, icon, and class of the nth MIDlet in the JAR file separated by a comma. The lowest value of <n> MUST be 1 and consecutive ordinals MUST be used. The first missing entry terminates the list. Any additional entries are ignored. Leading and trailing spaces in name, icon and class are ignored. Name is used to identify this MIDlet to the user. The name must be present and be non-null. Icon is the case-sensitive absolute path name of an image (PNG) within the JAR for the icon of the nth MIDlet. The icon may be omitted. Class is the name of the class extending the `javax.microedition.midlet.MIDlet` class for the nth MIDlet. The classname MUST be non-null and contain only characters for Java class names. The class MUST have a public no-args constructor. The class name IS case sensitive.

MIDlet-Jar-URL	The URL from which the JAR file can be loaded. The syntax and meaning MUST conform to RFC2396 and RFCs that define each scheme. Both absolute and relative URLs MUST be supported. The context for a relative URL is the URL from which this application descriptor was loaded.
MIDlet-Jar-Size	The number of bytes in the JAR file.
MIDlet-Data-Size	The minimum number of bytes of persistent data required by the MIDlet. The device may provide additional storage according to its own policy. The default is zero.
MicroEdition-Profile	The J2ME profiles required, using the same format and value as the System property `microedition.profiles` (for example "MIDP-2.0"). The device must implement *all* of the profiles listed. If any of the profiles are not implemented the installation MUST fail. Multiple profiles are separated with a blank (Unicode U+0020).
MicroEdition-Configuration	The J2ME Configuration required using the same format and value as the System property `microedition.configuration` (for example "CLDC-1.0").
MIDlet-Permissions	Zero or more permissions that are critical to the function of the MIDlet suite. See the MIDlet Suite Security section for details of usage.
MIDlet-Permissions-Opt	Zero or more permissions that are non-critical to the function of the MIDlet suite. See the MIDlet Suite Security section for details of usage.
MIDlet-Push-<n>	Register a MIDlet to handle inbound connections. Refer to javax.microedition.io.PushRegistry for details.
MIDlet-Install-Notify	Refer to the OTA Specification for details.
MIDlet-Delete-Notify	Refer to the OTA Specification for details.
MIDlet-Delete-Confirm	Refer to the OTA Specification for details.

Some attributes use multiple values, for those attributes the values are separated by a comma (Unicode U+002C) except where noted. Leading and trailing whitespace (Unicode U+0020) and tab (Unicode U+0009) are ignored on each value.

Version Numbering

Version numbers have the format Major.Minor[.Micro] (X.X[.X]), where the .Micro portion MAY be omitted. (If the .Micro portion is not omitted, then it defaults to zero). In addition, each portion of the version number is allowed a maximum of two decimal digits (i.e., 0-99). Version numbers are described in the the Java(TM) Product Versioning Specification http://java.sun.com/products/jdk/1.2/docs/guide/versioning/spec/VersioningSpecification.html (http://java.sun.com/products/jdk/1.2/docs/guide/versioning/spec/VersioningSpecification.html).

For example, 1.0.0 can be used to specify the first version of a MIDlet suite. For each portion of the version number, leading zeros are not significant. For example, 08 is equivalent to 8. Also, 1.0 is equivalent to 1.0.0. However, 1.1 is equivalent to 1.1.0, and not 1.0.1.

A missing MIDlet-Version tag is assumed to be 0.0.0, which means that any non-zero version number is considered as a newer version of the MIDlet suite.

JAR Manifest

The manifest provides information about the contents of the JAR file. JAR file formats and specifications are available at http://java.sun.com/products/jdk/1.2/docs/guide/jar/index.html. (http://java.sun.com/products/jdk/

1.2/docs/guide/jar/index.html) Refer to the JDK JAR and manifest documentation for the syntax and related details. MIDP implementations MUST implement handling of lines longer than 72 bytes as defined in the manifest specification. An attribute MUST not appear more than once within the manifest. If an attribute is duplicated the effect is unspecified. Manifest attributes are passed to the MIDlet when requested using the `MIDlet.getAppProperty` method, unless the attribute is duplicated in the application descriptor, for handling of duplicate attributes see the "Application Descriptor" section.

The manifest MUST contain the following attributes:

- MIDlet-Name
- MIDlet-Version
- MIDlet-Vendor

The manifest or the application descriptor MUST contain the following attributes:

- MIDlet-<n> for each MIDlet
- MicroEdition-Profile
- MicroEdition-Configuration

The manifest MAY contain the following:

- MIDlet-Description
- MIDlet-Icon
- MIDlet-Info-URL
- MIDlet-Data-Size
- MIDlet-Permissions
- MIDlet-Permissions-Opt
- MIDlet-Push-<n>
- MIDlet-Install-Notify
- MIDlet-Delete-Notify
- MIDlet-Delete-Confirm
- Any application-specific attributes that do not begin with `MIDlet-` or `MicroEdition-`

For example, a manifest for a hypothetical suite of card games would look like the following example:

```
MIDlet-Name: CardGames
MIDlet-Version: 1.1.9
MIDlet-Vendor: CardsRUS
MIDlet-1: Solitaire, /Solitare.png, org.cardsrus.games.Solitare
MIDlet-2: JacksWild, /
JacksWild.png, org.cardsrus.games.JacksWild
MicroEdition-Profile: MIDP-2.0
MicroEdition-Configuration: CLDC-1.0
Solitaire-Author: John Q. Public
```

MIDlet Classes

All Java classes needed by the MIDlet are be placed in the JAR file using the standard structure, based on mapping the fully qualified class names to directory and file names. Each period is converted to a forward slash

(/) and the `.class` extension is appended. For example, a class `com.sun.microedition.Test` would be placed in the JAR file with the name `com/sun/microedition/Test.class` .

Application Descriptor

Each JAR file MAY be accompanied by an application descriptor. The application descriptor is used in conjunction with the JAR manifest by the application management software to manage the MIDlet and is used by the MIDlet itself for configuration specific attributes. The descriptor allows the application management software on the device to verify that the MIDlet is suited to the device before loading the full JAR file of the MIDlet suite. It also allows configuration-specific attributes (parameters) to be supplied to the MIDlet(s) without modifying the JAR file.

To allow devices to dispatch an application descriptor to the MIDP application management software, a file extension and MIME type (http://www.iana.org/assignments/media-types/text/vnd.sun.j2me.app-descriptor) are registered with the IANA:

- The file extension of an application descriptor file is `jad`
- The MIME type of an application descriptor file is `text/vnd.sun.j2me.app-descriptor`.

A predefined set of attributes is specified to allow the application management software to identify, retrieve, and install the MIDlet(s). All attributes appearing in the descriptor file are made available to the MIDlet(s). The developer may use attributes not beginning with `MIDlet-` or `MicroEdition-` for application-specific purposes. Attribute names are case-sensitive and MUST match exactly. An attribute MUST NOT appear more than once within the manifest. If an attribute is duplicated the effect is unspecified. The MIDlet retrieves attributes by name by calling the `MIDlet.getAppProperty` method.

The application descriptor MUST contain the following attributes:

- MIDlet-Name
- MIDlet-Version
- MIDlet-Vendor
- MIDlet-Jar-URL
- MIDlet-Jar-Size

The application descriptor MAY contain:

- MIDlet-<n> for each MIDlet
- MicroEdition-Profile
- MicroEdition-Configuration
- MIDlet-Description
- MIDlet-Icon
- MIDlet-Info-URL
- MIDlet-Data-Size
- MIDlet-Permissions
- MIDlet-Permissions-Opt
- MIDlet-Push-<n>
- MIDlet-Install-Notify
- MIDlet-Delete-Notify

- MIDlet-Delete-Confirm
- Any application-specific attributes that do not begin with `MIDlet-` or `MicroEdition-`

The mandatory attributes MIDlet-Name, MIDlet-Version, and MIDlet-Vendor MUST be duplicated in the descriptor and manifest files since they uniquely identify the application. If they are not identical (not from the same application), then the JAR MUST NOT be installed.

Duplication of other manifest attributes in the application descriptor is not required and their values MAY differ even though both the manifest and descriptor files contain the same attribute for untrusted MIDlet suites. If the MIDlet suite is not trusted the value from the descriptor file will override the value from the manifest file. If the MIDlet suite is trusted then the values in the application descriptor MUST be identical to the corresponding attribute values in the Manifest.

MIDlets MUST NOT add any attributes to the manifest or the Application Descriptor that start with `MIDlet-` or `MicroEdition-` other than those defined in the relevant Configuration and Profiles (e.g. CLDC and MIDP) specifications. Unrecognized attributes MUST be ignored by the AMS.

Generally speaking, the format of the application descriptor is a sequence of lines consisting of an attribute name followed by a colon, the value of the attribute, and a carriage return. White space is ignored before and after the value. The order of the attributes is arbitrary.

The application descriptor MAY be encoded for transport or storage and MUST be converted to Unicode before parsing, using the rules below. For example, an ISO-8859-1 encoded file would need to be read through the equivalent of `java.io.InputStreamReader` with the appropriate encoding. The default character encoding for transporting a descriptor is UTF-8. Descriptors retrieved via HTTP, if that is supported, SHOULD use the standard HTTP content negotiation mechanisms, such as the Content-Encoding header and the Content-Type charset parameter to convert the stream to UCS-2.

BNF for Parsing Application Descriptors

```
appldesc:  *attrline
attrline:  attrname ":" [WSP] attrvalue [WSP] newlines
attrname:  1*<any Unicode char except
             CTLs or separators>
attrvalue: *valuechar | valuechar *(valuechar | WSP) valuechar
valuechar: <any valid Unicode character,
          excluding CTLS and WSP>
newlines = 1*newline  ; allow blank lines to be ignored
newline:   CR LF | LF

CR  = <Unicode carriage return (U+000D)>
LF  = <Unicode linefeed (U+000A)>

WSP:  1*( SP | HT )
SP  = <Unicode space (U+0020)>
HT  = <Unicode horizontal-tab (U+0009)>
CTL = <Unicode characters
         U+0000 - U+001F and U+007F>
separators: "(" | ")" | "<" |
           ">" | "@" |
           "," | ";" | ":" |
           "\" | <">|
           "/" | "[" | "]" |
           "?" | "=" |
           "{" | "}" | SP | HT
```

For example, an application descriptor for a hypothetical suite of card games would look look like the following example:

```
MIDlet-Name: CardGames
MIDlet-Version: 1.1.9
MIDlet-Vendor: CardsRUS
MIDlet-1: Solitaire, /Solitare.png, com.cardsrus.org.Solitare
MIDlet-2: JacksWild, /JacksWild.png, com.cardsrus.org.JacksWild
MicroEdition-Profile: MIDP-2.0
MicroEdition-Configuration: CLDC-1.0
MIDlet-Description: Really cool card games
MIDlet-Jar-URL: http://www.cardsrus.com/games/cardgames.jar
MIDlet-Jar-Size: 7378
MIDlet-Data-Size: 256
```

Application Lifecycle

Each MIDlet MUST extend the `MIDlet` class. The `MIDlet` class allows for the orderly starting, stopping, and cleanup of the MIDlet. The MIDlet can request the arguments from the application descriptor to communicate with the application management software. A MIDlet suite MUST NOT have a `public static void main()` method. If it exists, it MUST be ignored by the application management software. The application management software provides the initial class needed by the CLDC to start a MIDlet.

When a MIDlet suite is installed on a device, its classes, resource files, arguments, and persistent storage are kept on the device and ready for use. The MIDlet(s) are available to the user via the device's application management software.

When the MIDlet is run, an instance of the MIDlet's primary class is created using its public no-argument constructor, and the methods of the `MIDlet` are called to sequence the MIDlet through its various states. The MIDlet can either request changes in state or notify the application management software of state changes via the `MIDlet` methods. When the MIDlet is finished or terminated by the application management software, it is destroyed, and the resources it used can be reclaimed, including any objects it created and its classes. The MIDlet MUST NOT call `System.exit` , which will throw a `SecurityException` when called by a MIDlet.

The normal states of Java classes are not affected by these classes as they are loaded. Referring to any class will cause it to be loaded, and the normal static initialization will occur.

Class in `javax.microedition.midlet`	Description
`MIDlet`	Extended by a MIDlet to allow the application management software to start, stop, and destroy it.
`MIDletStateChangeException`	Thrown when the application cannot make the change requested.

MIDlet lifecycle

The `MIDlet` lifecycle defines the protocol between a `MIDlet` and its environment through the following:

- A simple well-defined state machine
- A concise definition of the MIDlet's states
- APIs to signal changes between the states

MIDlet Lifecycle Definitions

The following definitions are used in the `MIDlet` lifecycle:

- **application management software -** a part of the device's software operating environment that manages `MIDlets`. It maintains the MIDlet state and directs the `MIDlet` through state changes.
- **`MIDlet`** - a MIDP application on the device. The `MIDlet` can signal the application management software about whether is it wants to run or has completed. A `MIDlet` has no knowledge of other `MIDlets` through the `MIDlet` API.
- **`MIDlet States`** - the states a `MIDlet` can have are defined by the transitions allowable through the `MIDlet` interface. More specific application states are known only to the application.

MIDlet States

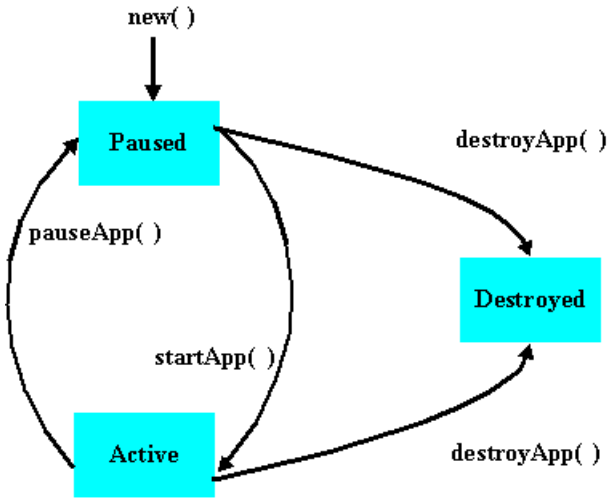
The `MIDlet` state machine is designed to ensure that the behavior of an application is consistent and as close as possible to what device manufactures and users expect, specifically:

- The perceived startup latency of an application SHOULD be very short.
- It SHOULD be possible to put an application into a state where it is not active.
- It SHOULD be possible to destroy an application at any time.

The valid states for `MIDlets` are:

State Name	Description
Paused	The `MIDlet` is initialized and is quiescent. It SHOULD not be holding or using any shared resources. This state is entered: After the `MIDlet` has been created using `new`. The public no-argument constructor for the `MIDlet` is called and returns without throwing an exception. The application typically does little or no initialization in this step. If an exception occurs, the application immediately enters the Destroyed state and is discarded. From the Active state after the `MIDlet.pauseApp()` method is called from the AMS and returns successfully. From the Active state when the `MIDlet.notifyPaused()` method returns successfully to the `MIDlet`. From the Active state if `startApp` throws an `MIDletStateChangeException`.
Active	The `MIDlet` is functioning normally. This state is entered: Just prior to the AMS calling the `MIDlet.startApp()` method.
Destroyed	The `MIDlet` has released all of its resources and terminated. This state is entered: When the AMS called the `MIDlet.destroyApp()` method and returns successfully, except in the case when the `unconditional` argument is false and a `MIDletStateChangeException` is thrown. The `destroyApp()` method shall release all resources held and perform any necessary cleanup so it may be garbage collected. When the `MIDlet.notifyDestroyed()` method returns successfully to the application. The `MIDlet` must have performed the equivalent of the `MIDlet.destroyApp()` method before calling `MIDlet.notifyDestroyed()`. **Note:** This state is only entered once.

The states and transitions for a `MIDlet` are:



MIDlet Lifecycle Model

A typical sequence of `MIDlet` execution is:

Application Management Software	`MIDlet`
The application management software creates a new instance of a `MIDlet`.	The default (no argument) constructor for the `MIDlet` is called; it is in the Paused state.
The application management software has decided that it is an appropriate time for the `MIDlet` to run, so it calls the `MIDlet.startApp` method for it to enter the Active state.	The `MIDlet` acquires any resources it needs and begins to perform its service.
The application management software wants the MIDlet to significantly reduce the amount of resources it is consuming, so that they may temporarily be used by other functions on the device such as a phone call or running another MIDlet. The AMS will signal this request to the MIDlet by calling the `MIDlet.pauseApp` method. The MIDlet should then reduce its resource consumption as much as possible.	The `MIDlet` stops performing its service and might choose to release some resources it currently holds.
The application management software has determined that the `MIDlet` is no longer needed, or perhaps needs to make room for a higher priority application in memory, so it signals the `MIDlet` that it is a candidate to be destroyed by calling the `MIDlet.destroyApp` method.	If it has been designed to do so, the `MIDlet` saves state or user preferences and performs clean up.

MIDlet Interface

- `pauseApp` - the `MIDlet` SHOULD release any temporary resources and become passive
- `startApp` - the `MIDlet` SHOULD acquire any resources it needs and resume

- `destroyApp` - the `MIDlet` SHOULD save any state and release all resources
- `notifyDestroyed` - the `MIDlet` notifies the application management software that it has cleaned up and is done
- `notifyPaused` - the `MIDlet` notifies the application management software that it has paused
- `resumeRequest` - the `MIDlet` asks application management software to be started again
- `getAppProperty` - gets a named property from the `MIDlet`

Application Implementation Notes

The application SHOULD take measures to avoid race conditions in the execution of the `MIDlet` methods. Each method may need to synchronize itself with the other methods avoid concurrency problems during state changes.

Example MIDlet Application

The example uses the MIDlet lifecycle to do a simple measurement of the speed of the Java Virtual Machine.

```
import javax.microedition.midlet.*;

/**
 * An example MIDlet runs a simple timing test
 * When it is started by the application management software it will
 * create a separate thread to do the test.
 * When it finishes it will notify the application management software
 * it is done.
 * Refer to the startApp, pauseApp, and destroyApp
 * methods so see how it handles each requested transition.
 */
public class MethodTimes extends MIDlet implements Runnable {
    // The state for the timing thread.
    Thread thread;

    /**
     * Start creates the thread to do the timing.
     * It should return immediately to keep the dispatcher
     * from hanging.
     */
    public void startApp() {
        thread = new Thread(this);
        thread.start();
    }

    /**
     * Pause signals the thread to stop by clearing the thread field.
     * If stopped before done with the iterations it will
     * be restarted from scratch later.
     */
    public void pauseApp() {
        thread = null;
    }

    /**
     * Destroy must cleanup everything.  The thread is signaled
     * to stop and no result is produced.
     */
    public void destroyApp(boolean unconditional) {
        thread = null;
    }

   /**
     * Run the timing test, measure how long it takes to
     * call a empty method 1000 times.
     * Terminate early if the current thread is no longer
     * the thread from the
     */
    public void run() {
        Thread curr = Thread.currentThread();  // Remember which thread is current
        long start = System.currentTimeMillis();
        for (int i = 0; i < 1000000 && thread == curr; i++) {
            empty();
        }
        long end = System.currentTimeMillis();

        // Check if timing was aborted, if so just exit
        // The rest of the application has already become quiescent.
        if (thread != curr) {
            return;
        }
        long millis = end - start;
        // Reporting the elapsed time is outside the scope of this example.

        // All done cleanup and quit
        destroyApp(true);
        notifyDestroyed();
    }
```

```
    /**
     * An Empty method.
     */
    void empty() {
    }
}
```

Since: MIDP 1.0

Class Summary	
Classes	
MIDlet[444]	A `MIDlet` is a MID Profile application.
Exceptions	
MIDletStateChangeException[450]	Signals that a requested `MIDlet` state change failed.

javax.microedition.midlet
MIDlet

Declaration

```
public abstract class MIDlet

Object
  |
  +--javax.microedition.midlet.MIDlet
```

Description

A `MIDlet` is a MID Profile application. The application must extend this class to allow the application management software to control the MIDlet and to be able to retrieve properties from the application descriptor and notify and request state changes. The methods of this class allow the application management software to create, start, pause, and destroy a MIDlet. A `MIDlet` is a set of classes designed to be run and controlled by the application management software via this interface. The states allow the application management software to manage the activities of multiple `MIDlets` within a runtime environment. It can select which `MIDlet`s are active at a given time by starting and pausing them individually. The application management software maintains the state of the `MIDlet` and invokes methods on the `MIDlet` to notify the MIDlet of change states. The `MIDlet` implements these methods to update its internal activities and resource usage as directed by the application management software. The `MIDlet` can initiate some state changes itself and notifies the application management software of those state changes by invoking the appropriate methods.

Note: The methods on this interface signal state changes. The state change is not considered complete until the state change method has returned. It is intended that these methods return quickly.

Member Summary

Constructors

protected	MIDlet() 445

Methods

int	checkPermission(String permission) 445
protected abstract void	destroyApp(boolean unconditional) 445
java.lang.String	getAppProperty(String key) 446
void	notifyDestroyed() 446
void	notifyPaused() 446
protected abstract void	pauseApp() 447
boolean	platformRequest(String URL) 447
void	resumeRequest() 448
protected abstract void	startApp() 448

Inherited Member Summary
Methods inherited from class `Object` `equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Constructors

MIDlet()

Declaration:
```
protected MIDlet()
```

Description:
Protected constructor for subclasses. The application management software is responsible for creating MIDlets and creation of MIDlets is restricted. MIDlets should not attempt to create other MIDlets.

Throws:
`SecurityException` - unless the application management software is creating the MIDlet.

Methods

checkPermission(String)

Declaration:
```
public final int checkPermission(String permission)
```

Description:
Get the status of the specified permission. If no API on the device defines the specific permission requested then it must be reported as denied. If the status of the permission is not known because it might require a user interaction then it should be reported as unknown.

Parameters:
`permission` - to check if denied, allowed, or unknown.

Returns: 0 if the permission is denied; 1 if the permission is allowed; -1 if the status is unknown

Since: MIDP 2.0

destroyApp(boolean)

Declaration:
```
protected abstract void destroyApp(boolean unconditional)
            throws MIDletStateChangeException
```

Description:
Signals the `MIDlet` to terminate and enter the *Destroyed* state. In the destroyed state the `MIDlet` must release all resources and save any persistent state. This method may be called from the *Paused* or *Active* states.

`MIDlet`s should perform any operations required before being terminated, such as releasing resources or saving preferences or state.

Note: The `MIDlet` can request that it not enter the *Destroyed* state by throwing an `MIDletStateChangeException`. This is only a valid response if the `unconditional` flag is set to `false`. If it is `true` the `MIDlet` is assumed to be in the *Destroyed* state regardless of how this method terminates. If it is not an unconditional request, the `MIDlet` can signify that it wishes to stay in its current state by throwing the `MIDletStateChangeException`. This request may be honored and the `destroy()` method called again at a later time.

If a Runtime exception occurs during `destroyApp` then they are ignored and the MIDlet is put into the *Destroyed* state.

Parameters:

`unconditional` - If true when this method is called, the `MIDlet` must cleanup and release all resources. If false the `MIDlet` may throw `MIDletStateChangeException` to indicate it does not want to be destroyed at this time.

Throws:

`MIDletStateChangeException` 450 - is thrown if the `MIDlet` wishes to continue to execute (Not enter the *Destroyed* state). This exception is ignored if `unconditional` is equal to `true`.

getAppProperty(String)

Declaration:

```
public final String getAppProperty(String key)
```

Description:

Provides a `MIDlet` with a mechanism to retrieve named properties from the application management software. The properties are retrieved from the combination of the application descriptor file and the manifest. For trusted applications the values in the manifest MUST NOT be overridden by those in the application descriptor. If they differ, the MIDlet will not be installed on the device. For untrusted applications, if an attribute in the descriptor has the same name as an attribute in the manifest the value from the descriptor is used and the value from the manifest is ignored.

Parameters:

`key` - the name of the property

Returns: A string with the value of the property. `null` is returned if no value is available for the key.

Throws:

`NullPointerException` - is thrown if key is `null`.

notifyDestroyed()

Declaration:

```
public final void notifyDestroyed()
```

Description:

Used by an `MIDlet` to notify the application management software that it has entered into the *Destroyed* state. The application management software will not call the MIDlet's `destroyApp` method, and all resources held by the `MIDlet` will be considered eligible for reclamation. The `MIDlet` must have performed the same operations (clean up, releasing of resources etc.) it would have if the `MIDlet.destroyApp()` had been called.

notifyPaused()

Declaration:

```
public final void notifyPaused()
```

Description:
Notifies the application management software that the MIDlet does not want to be active and has entered the *Paused* state. Invoking this method will have no effect if the `MIDlet` is destroyed, or if it has not yet been started.

It may be invoked by the `MIDlet` when it is in the *Active* state.

If a `MIDlet` calls `notifyPaused()`, in the future its `startApp()` method may be called make it active again, or its `destroyApp()` method may be called to request it to destroy itself.

If the application pauses itself it will need to call `resumeRequest` to request to reenter the `active` state.

pauseApp()

Declaration:
```
protected abstract void pauseApp()
```

Description:
Signals the `MIDlet` to enter the *Paused* state. In the *Paused* state the `MIDlet` must release shared resources and become quiescent. This method will only be called called when the `MIDlet` is in the *Active* state.

If a Runtime exception occurs during `pauseApp` the MIDlet will be destroyed immediately. Its `destroyApp` will be called allowing the MIDlet to cleanup.

platformRequest(String)

Declaration:
```
public final boolean platformRequest(String URL)
            throws ConnectionNotFoundException
```

Description:
Requests that the device handle (for example, display or install) the indicated URL.

If the platform has the appropriate capabilities and resources available, it SHOULD bring the appropriate application to the foreground and let the user interact with the content, while keeping the MIDlet suite running in the background. If the platform does not have appropriate capabilities or resources available, it MAY wait to handle the URL request until after the MIDlet suite exits. In this case, when the requesting MIDlet suite exits, the platform MUST then bring the appropriate application (if one exists) to the foreground to let the user interact with the content.

This is a non-blocking method. In addition, this method does NOT queue multiple requests. On platforms where the MIDlet suite must exit before the request is handled, the platform MUST handle only the last request made. On platforms where the MIDlet suite and the request can be handled concurrently, each request that the MIDlet suite makes MUST be passed to the platform software for handling in a timely fashion.

If the URL specified refers to a MIDlet suite (either an Application Descriptor or a JAR file), the application handling the request MUST interpret it as a request to install the named package. In this case, the platform's normal MIDlet suite installation process SHOULD be used, and the user MUST be allowed to control the process (including cancelling the download and/or installation). If the MIDlet suite being installed is an *update* of the currently running MIDlet suite, the platform MUST first stop the currently running MIDlet suite before performing the update. On some platforms, the currently running MIDlet suite MAY need to be stopped before any installations can occur.

If the URL specified is of the form `tel:<number>`, as specified in RFC2806 (http://www.ietf.org/rfc/rfc2806.txt), then the platform MUST interpret this as a request to initiate a voice call. The request MUST be passed to the “phone” application to handle if one is present in the platform. The “phone” application, if present, MUST be able to set up local and global phone calls and also perform DTMF post dialing. Not all elements of RFC2806 need be implemented, especially the area-specifier or any other requirement on the terminal to know its context. The isdn-subaddress, service-provider and future-extension may also be ignored. Pauses during dialing are not relevant in some telephony services.

Devices MAY choose to support additional URL schemes beyond the requirements listed above.

Many of the ways this method will be used could have a financial impact to the user (e.g. transferring data through a wireless network, or initiating a voice call). Therefore the platform MUST ask the user to explicitly acknowlege each request before the action is taken. Implementation freedoms are possible so that a pleasant user experience is retained. For example, some platforms may put up a dialog for each request asking the user for permission, while other platforms may launch the appropriate application and populate the URL or phone number fields, but not take the action until the user explicitly clicks the load or dial buttons.

Parameters:

`URL` - The URL for the platform to load. An empty string (not null) cancels any pending requests.

Returns: true if the MIDlet suite MUST first exit before the content can be fetched.

Throws:

`javax.microedition.io.ConnectionNotFoundException` - if the platform cannot handle the URL requested.

Since: MIDP 2.0

resumeRequest()

Declaration:

```
public final void resumeRequest()
```

Description:

Provides a `MIDlet` with a mechanism to indicate that it is interested in entering the *Active* state. Calls to this method can be used by the application management software to determine which applications to move to the *Active* state.

When the application management software decides to activate this application it will call the `startApp` method.

The application is generally in the *Paused* state when this is called. Even in the paused state the application may handle asynchronous events such as timers or callbacks.

startApp()

Declaration:

```
protected abstract void startApp()
            throws MIDletStateChangeException
```

Description:

Signals the `MIDlet` that it has entered the *Active* state. In the *Active* state the `MIDlet` may hold resources. The method will only be called when the `MIDlet` is in the *Paused* state.

Two kinds of failures can prevent the service from starting, transient and non-transient. For transient failures the `MIDletStateChangeException` exception should be thrown. For non-transient failures the `notifyDestroyed` method should be called.

If a Runtime exception occurs during `startApp` the MIDlet will be destroyed immediately. Its `destroyApp` will be called allowing the MIDlet to cleanup.

Throws:

`MIDletStateChangeException`[450] - is thrown if the `MIDlet` cannot start now but might be able to start at a later time.

javax.microedition.midlet
MIDletStateChangeException

Declaration
```
public class MIDletStateChangeException extends Exception
```

```
Object
  |
  +--Throwable
        |
        +--Exception
              |
              +--javax.microedition.midlet.MIDletStateChangeException
```

Description
Signals that a requested `MIDlet` state change failed. This exception is thrown by the `MIDlet` in response to state change calls into the application via the `MIDlet` interface

Since: MIDP 1.0

See Also: `MIDlet`$_{444}$

Member Summary	
Constructors	
	`MIDletStateChangeException()`$_{450}$
	`MIDletStateChangeException(String s)`$_{451}$

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()`
Methods inherited from class `Throwable`
`getMessage(), printStackTrace(), toString()`

Constructors

MIDletStateChangeException()

Declaration:
```
public MIDletStateChangeException()
```

Description:
Constructs an exception with no specified detail message.

MIDletStateChangeException(String)

Declaration:
```
public MIDletStateChangeException(String s)
```

Description:
Constructs an exception with the specified detail message.

Parameters:
`s` - the detail message

MIDletStateChangeException(String)

CHAPTER 13

Package javax.microedition.pki

Description

Certificates are used to authenticate information for secure Connections. The `Certificate` interface provides to the application information about the origin and type of the certificate. The `CertificateException` provides information about failures that may occur while verifying or using certificates.

The MIDP X.509 Certificate Profile below defines the format and usage of certificates. X.509 Certificates MUST be supported. Other certificate formats MAY be supported. The implementation MAY store only the essential information from certificates. Internally, the fields of the certificate MAY be stored in any format that is suitable for the implementation.

References

MIDP 2.0 devices are expected to operate using standard Internet and wireless protocols and techniques for transport and security. The current mechanisms for securing Internet content is based on existing Internet standards for public key cryptography:

- [RFC2437] - PKCS #1 RSA Encryption Version 2.0 (http://www.ietf.org/rfc/rfc2437)
- [RFC2459] - Internet X.509 Public Key Infrastructure (http://www.ietf.org/rfc/rfc2459)
- [WAPCERT] - WAP-211-WAPCert-20010522-a - WAP Certificate Profile Specification (http://www.wapforum.org/what/technical.htm)

MIDP X.509 Certificate Profile

WAP-211-WAPCert-20010522-a [WAPCert] which is based on RFC2459 Internet X.509 Public Key Infrastructure Certificate and CRL Profile [RFC2459].

Devices MUST conform to all mandatory requirements in [WAPCert] and SHOULD conform to all optional requirements in [WAPCert] except those requirements in excluded sections listed below. Mandatory and optional requirements are listed in Appendix C of [WAPCert]. Additional requirements, ON TOP of those listed in [WAPCert] are given below.

- Excluding [WAPCert] Section 6.2, User Certificates for Authentication
- Excluding [WAPCert] Section 6.3, User Certificates for Digital Signatures

RFC2459 contains sections which are not relevant to implementations of this specification. The WAP Certificate Profile does not mention these functions. The sections to be excluded are:

- Exclude the requirements from Paragraphs 4 of Section 4.2 - Standard Certificate Extensions. A conforming implementation of this specification does not need to recognize extensions that must or may be critical including certificate policies, name constraints, and policy constraints.
- Exclude RFC2459 Section 6.2 Extending Path Validation. Support for Policy Certificate Authority or policy attributes is not required.

Certificate Extensions
A version 1 X.509 certificate MUST be considered equivalent to a version 3 certificate with no extensions. At a minimum, a device conforming to this profile MUST recognize key usage (see RFC2459 sec. 4.2.1.3), basic constraints (see RFC2459 sec. 4.2.1.10).

Although a conforming device may not recognize the authority and subject key identifier (see RFC2459 sec. 4.2.1.1 and 4.2.1.2) extensions it MUST support certificate authorities that sign certificates using the same distinguished name but using multiple public keys.

Implementations MUST be able to process certificates with unknown distinguished name attributes.

Implementations MUST be able to process certificates with unknown, non-critical certificate extensions.

The `serialNumber` attribute defined by [WAPCert] must be recognized in distinguished names for Issuer and Subject.

Certificate Size
Devices must be able to process certificates that are not self-signed root CA certificates of size up to at least 1500 bytes.

Algorithm Support
A device MUST support the RSA signature algorithm with the SHA-1 hash function `sha1WithRSAEncryption` as defined by PKCS #1 [RFC2437]. Devices that support these algorithms MUST be capable of verifying signatures made with RSA keys of length up to and including 2048 bits.

Devices SHOULD support signature algorithms `md2WithRSAEncryption` and `md5WithRSAEncryption` as defined in [RFC2437]. Devices that support these algorithms MUST be capable of verifying signatures made with RSA keys of length up to and including 2048 bits.

Certificate Processing for HTTPS
Devices MUST recognize the extended key usage extension defined of RFC2818 if it is present and is marked critical and when present MUST verify that the extension contains the `id-kp-serverAuth` object identifier (see RFC2459 sec. 4.2.1.13).

SSL and TLS allow the web server to include the redundant root certificate in the server certificate message. In practice this certificate may not have the basic constraint extension (it is most likely a version 1 certificate), a device MUST ignore the redundant certificate in this case. Web servers SHOULD NOT include a self-signed root CA in a certificate chain.

Since: MIDP 2.0

Class Summary

Interfaces

`Certificate` 455	Interface common to certificates.

Exceptions

`CertificateException` 458	The `CertificateException` encapsulates an error that occurred while a `Certificate` is being used.

javax.microedition.pki
Certificate

Declaration
`public interface` **`Certificate`**

Description
Interface common to certificates. The features abstracted of `Certificates` include subject, issuer, type, version, serial number, signing algorithm, dates of valid use, and serial number.

Printable Representation for Binary Values

A non-string values in a certificate are represented as strings with each byte as two hex digits (capital letters for A-F) separated by ":" (Unicode U+003A).

For example: `0C:56:FA:80`

Printable Representation for X.509 Distinguished Names

For a X.509 certificate the value returned is the printable verision of the distingished name (DN) from the certificate.

An X.509 distinguished name of is set of attributes, each attribute is a sequence of an object ID and a value. For string comparison purposes, the following rules define a strict printable representation.

1. There is no added white space around separators.
2. The attributes are in the same order as in the certificate; attributes are not reordered.
3. If an object ID is in the table below, the label from the table will be substituted for the object ID, else the ID is formatted as a string using the binary printable representation above.
4. Each object ID or label and value within an attribute will be separated by a "=" (Unicode U+003D), even if the value is empty.
5. If value is not a string, then it is formatted as a string using the binary printable representation above.
6. Attributes will be separated by a ";" (Unicode U+003B)

Labels for X.500 Distinguished Name Attributes

Object ID	Binary	Label
id-at-commonName	`55:04:03`	CN
id-at-surname	`55:04:04`	SN
id-at-countryName	`55:04:06`	C
id-at-localityName	`55:04:07`	L
id-at-stateOrProvinceName	`55:04:08`	ST
id-at-streetAddress	`55:04:09`	STREET
id-at-organizationName	`55:04:0A`	O

id-at-organizationUnitName	55:04:0B	OU
emailAddress	2A:86:48:86:F7:0D:01:09:01	EmailAddress

Example of a printable distinguished name:

```
C=US;O=Any Company, Inc.;CN=www.anycompany.com
```

Since: MIDP 2.0

Member Summary

Methods

java.lang.String	getIssuer()[456]
long	getNotAfter()[456]
long	getNotBefore()[456]
java.lang.String	getSerialNumber()[457]
java.lang.String	getSigAlgName()[457]
java.lang.String	getSubject()[457]
java.lang.String	getType()[457]
java.lang.String	getVersion()[457]

Methods

getIssuer()

Declaration:
```
public String getIssuer()
```

Description:
Gets the name of this certificate's issuer.

Returns: The issuer of the `Certificate`; the value MUST NOT be `null`.

getNotAfter()

Declaration:
```
public long getNotAfter()
```

Description:
Gets the time after which this `Certificate` may not be used from the validity period.

Returns: The time in milliseconds after which the `Certificate` is not valid (expiration date); it MUST be positive; `Long.MAX_VALUE` is returned if the certificate does not have its validity restricted based on the time.

getNotBefore()

Declaration:
```
public long getNotBefore()
```

Description:
Gets the time before which this `Certificate` may not be used from the validity period.

Returns: The time in milliseconds before which the `Certificate` is not valid; it MUST be positive, `0` is returned if the certificate does not have its validity restricted based on the time.

getSerialNumber()

Declaration:
```
public String getSerialNumber()
```

Description:
Gets the printable form of the serial number of this `Certificate`. If the serial number within the `certificate` is binary it should be formatted as a string using the binary printable representation in class description. For example, 0C:56:FA:80.

Returns: A string containing the serial number in user-friendly form; `null` is returned if there is no serial number.

getSigAlgName()

Declaration:
```
public String getSigAlgName()
```

Description:
Gets the name of the algorithm used to sign the `Certificate`. The algorithm names returned should be the labels defined in RFC2459 Section 7.2.

Returns: The name of signature algorithm; the value MUST NOT be `null`.

getSubject()

Declaration:
```
public String getSubject()
```

Description:
Gets the name of this certificate's subject.

Returns: The subject of this `Certificate`; the value MUST NOT be `null`.

getType()

Declaration:
```
public String getType()
```

Description:
Get the type of the `Certificate`. For X.509 Certificates the value returned is "X.509".

Returns: The type of the `Certificate`; the value MUST NOT be `null`.

getVersion()

Declaration:
```
public String getVersion()
```

Description:
Gets the version number of this `Certificate`. The format of the version number depends on the specific type and specification. For a X.509 certificate per RFC 2459 it would be "2".

Returns: The version number of the `Certificate`; the value MUST NOT be `null`.

javax.microedition.pki
CertificateException

Declaration

```
public class CertificateException extends java.io.IOException

Object
  |
  +--Throwable
        |
        +--Exception
              |
              +--java.io.IOException
                    |
                    +--javax.microedition.pki.CertificateException
```

Description

The `CertificateException` encapsulates an error that occurred while a `Certificate` is being used. If multiple errors are found within a `Certificate` the more significant error should be reported in the exception.

Since: MIDP 2.0

Member Summary

Fields

static byte	BAD_EXTENSIONS 459
static byte	BROKEN_CHAIN 459
static byte	CERTIFICATE_CHAIN_TOO_LONG 459
static byte	EXPIRED 459
static byte	INAPPROPRIATE_KEY_USAGE 459
static byte	MISSING_SIGNATURE 459
static byte	NOT_YET_VALID 460
static byte	ROOT_CA_EXPIRED 460
static byte	SITENAME_MISMATCH 460
static byte	UNAUTHORIZED_INTERMEDIATE_CA 460
static byte	UNRECOGNIZED_ISSUER 460
static byte	UNSUPPORTED_PUBLIC_KEY_TYPE 460
static byte	UNSUPPORTED_SIGALG 460
static byte	VERIFICATION_FAILED 461

Constructors

	CertificateException(Certificate certificate, byte status) 461
	CertificateException(String message, Certificate certificate, byte status) 461

Methods

Certificate	getCertificate() 461
byte	getReason() 462

Inherited Member Summary

Methods inherited from class `Object`

`equals(Object), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()`

Methods inherited from class `Throwable`

`getMessage(), printStackTrace(), toString()`

Fields

BAD_EXTENSIONS

Declaration:
`public static final byte BAD_EXTENSIONS`

Description:
Indicates a certificate has unrecognized critical extensions. The value is 1.

BROKEN_CHAIN

Declaration:
`public static final byte BROKEN_CHAIN`

Description:
Indicates a certificate in a chain was not issued by the next authority in the chain. The value is 11.

CERTIFICATE_CHAIN_TOO_LONG

Declaration:
`public static final byte CERTIFICATE_CHAIN_TOO_LONG`

Description:
Indicates the server certificate chain exceeds the length allowed by an issuer’s policy. The value is 2.

EXPIRED

Declaration:
`public static final byte EXPIRED`

Description:
Indicates a certificate is expired. The value is 3.

INAPPROPRIATE_KEY_USAGE

Declaration:
`public static final byte INAPPROPRIATE_KEY_USAGE`

Description:
Indicates a certificate public key has been used in way deemed inappropriate by the issuer. The value is 10.

MISSING_SIGNATURE

Declaration:
`public static final byte MISSING_SIGNATURE`

NOT_YET_VALID

Description:
Indicates a certificate object does not contain a signature. The value is 5.

NOT_YET_VALID

Declaration:
`public static final byte NOT_YET_VALID`

Description:
Indicates a certificate is not yet valid. The value is 6.

ROOT_CA_EXPIRED

Declaration:
`public static final byte ROOT_CA_EXPIRED`

Description:
Indicates the root CA's public key is expired. The value is 12.

SITENAME_MISMATCH

Declaration:
`public static final byte SITENAME_MISMATCH`

Description:
Indicates a certificate does not contain the correct site name. The value is 7.

UNAUTHORIZED_INTERMEDIATE_CA

Declaration:
`public static final byte UNAUTHORIZED_INTERMEDIATE_CA`

Description:
Indicates an intermediate certificate in the chain does not have the authority to be a intermediate CA. The value is 4.

UNRECOGNIZED_ISSUER

Declaration:
`public static final byte UNRECOGNIZED_ISSUER`

Description:
Indicates a certificate was issued by an unrecognized entity. The value is 8.

UNSUPPORTED_PUBLIC_KEY_TYPE

Declaration:
`public static final byte UNSUPPORTED_PUBLIC_KEY_TYPE`

Description:
Indicates that type of the public key in a certificate is not supported by the device. The value is 13.

UNSUPPORTED_SIGALG

Declaration:
`public static final byte UNSUPPORTED_SIGALG`

Description:
Indicates a certificate was signed using an unsupported algorithm. The value is 9.

VERIFICATION_FAILED

Declaration:
```
public static final byte VERIFICATION_FAILED
```

Description:
Indicates a certificate failed verification. The value is 14.

Constructors

CertificateException(Certificate, byte)

Declaration:
```
public CertificateException(javax.microedition.pki.Certificate[455] certificate,
            byte status)
```

Description:
Create a new exception with a `Certificate` and specific error reason. The descriptive message for the new exception will be automatically provided, based on the reason.

Parameters:
`certificate` - the certificate that caused the exception

`status` - the reason for the exception; the status MUST be between BAD_EXTENSIONS and VERIFICATION_FAILED inclusive.

CertificateException(String, Certificate, byte)

Declaration:
```
public CertificateException(String message,
            javax.microedition.pki.Certificate[455] certificate, byte status)
```

Description:
Create a new exception with a message, `Certificate`, and specific error reason.

Parameters:
`message` - a descriptive message

`certificate` - the certificate that caused the exception

`status` - the reason for the exception; the status MUST be between BAD_EXTENSIONS and VERIFICATION_FAILED inclusive.

Methods

getCertificate()

Declaration:
```
public javax.microedition.pki.Certificate[455] getCertificate()
```

Description:
Get the `Certificate` that caused the exception.

Returns: the `Certificate` that included the failure.

getReason()

Declaration:
`public byte getReason()`

Description:
Get the reason code.

Returns: the reason code

CHAPTER 14

Package javax.microedition.rms

Description

The Mobile Information Device Profile provides a mechanism for MIDlets to persistently store data and later retrieve it. This persistent storage mechanism is modeled after a simple record oriented database and is called the Record Management System.

Persistent Storage

The MIDP provides a mechanism for MIDlets to persistently store data and retrieve it later. This persistent storage mechanism, called the Record Management System (RMS), is modeled after a simple record-oriented database.

Record Store

A record store consists of a collection of records that will remain persistent across multiple invocations of a MIDlet. The platform is responsible for making its best effort to maintain the integrity of the MIDlet's record stores throughout the normal use of the platform, including reboots, battery changes, etc.

Record stores are created in platform-dependent locations, which are not exposed to MIDlets. The naming space for record stores is controlled at the MIDlet suite granularity. MIDlets within a MIDlet suite are allowed to create multiple record stores, as long as they are each given different names. When a MIDlet suite is removed from a platform, all record stores associated with its MIDlets MUST also be removed. MIDlets within a MIDlet suite can access one another's record stores directly. New APIs in MIDP 2.0 allow for the explicit sharing of record stores if the MIDlet creating the RecordStore chooses to give such permission.

Sharing is accomplished through the ability to name a RecordStore in another MIDlet suite and by defining the accessibilty rules related to the Authentication of the two MIDlet suites.

RecordStores are uniquely named using the unique name of the MIDlet suite plus the name of the RecordStore. MIDlet suites are identified by the MIDlet-Vendor and MIDlet-Name attributes from the application descriptor.

Access controls are defined when RecordStores to be shared are created. Access controls are enforced when RecordStores are opened. The access modes allow private use or shareable with any other MIDlet suite.

Record store names are case sensitive and may consist of any combination of up to 32 Unicode characters. Record store names MUST be unique within the scope of a given MIDlet suite. In other words, MIDlets within a MIDlet suite are not allowed to create more than one record store with the same name; however, a MIDlet in one MIDlet suite is allowed to have a record store with the same name as a MIDlet in another MIDlet suite. In that case, the record stores are still distinct and separate.

No locking operations are provided in this API. Record store implementations ensure that all individual record store operations are atomic, synchronous, and serialized so that no corruption occurs with multiple accesses. However, if a MIDlet uses multiple threads to access a record store, it is the MIDlet's responsibility to coordinate this access, or unintended consequences may result. For example, if two threads in a MIDlet both call `RecordStore.setRecord()` concurrently on the same record, the record store will serialize these calls properly, and no database corruption will occur as a result. However, one of the writes will be subsequently overwritten by the other, which may cause problems within the MIDlet. Similarly, if a platform performs

transparent synchronization of a record store or other access from below, it is the platform's responsibility to enforce exclusive access to the record store between the MIDlets and synchronization engine.

This record store API uses long integers for time/date stamps, in the format used by `System.currentTimeMillis()`. The record store is time stamped with the last time it was modified. The record store also maintains a version, which is an integer that is incremented for each operation that modifies the contents of the record store. These are useful for synchronization engines as well as applications.

Records

Records are arrays of bytes. Developers can use `DataInputStream` and `DataOutputStream` as well as `ByteArrayInputStream` and `ByteArrayOutputStream` to pack and unpack different data types into and out of the byte arrays.

Records are uniquely identified within a given record store by their `recordId`, which is an integer value. This `recordId` is used as the primary key for the records. The first record created in a record store will have `recordId` equal to 1, and each subsequent `recordId` will monotonically increase by one. For example, if two records are added to a record store, and the first has a `recordId` of 'n', the next will have a `recordId` of (n+1). MIDlets can create other indices by using the `RecordEnumeration` class.

Example:

The example uses the Record Management System to store and retrieve high scores for a game. In the example, high scores are stored in separate records, and sorted when necessary using a RecordEnumeration.

```
import javax.microedition.rms.*;
import java.io.DataOutputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.ByteArrayInputStream;
import java.io.DataInputStream;
import java.io.EOFException;
/**
 * A class used for storing and showing game scores.
 */
public class RMSGameScores
    implements RecordFilter, RecordComparator
{
    /*
     * The RecordStore used for storing the game scores.
     */
    private RecordStore recordStore = null;
    /*
     * The player name to use when filtering.
     */
    public static String playerNameFilter = null;
    /*
     * Part of the RecordFilter interface.
     */
    public boolean matches(byte[] candidate)
throws IllegalArgumentException
    {
// If no filter set, nothing can match it.
if (this.playerNameFilter == null) {
    return false;
}
ByteArrayInputStream bais = new ByteArrayInputStream(candidate);
DataInputStream inputStream = new DataInputStream(bais);
String name = null;
try {
    int score = inputStream.readInt();
    name = inputStream.readUTF();
}
catch (EOFException eofe) {
    System.out.println(eofe);
    eofe.printStackTrace();
}
catch (IOException eofe) {
    System.out.println(eofe);
    eofe.printStackTrace();
}
return (this.playerNameFilter.equals(name));
    }
    /*
     * Part of the RecordComparator interface.
     */
    public int compare(byte[] rec1, byte[] rec2)
    {
// Construct DataInputStreams for extracting the scores from
// the records.
ByteArrayInputStream bais1 = new ByteArrayInputStream(rec1);
DataInputStream inputStream1 = new DataInputStream(bais1);
ByteArrayInputStream bais2 = new ByteArrayInputStream(rec2);
DataInputStream inputStream2 = new DataInputStream(bais2);
int score1 = 0;
int score2 = 0;
try {
    // Extract the scores.
    score1 = inputStream1.readInt();
    score2 = inputStream2.readInt();
}
catch (EOFException eofe) {
    System.out.println(eofe);
```

```
    eofe.printStackTrace();
}
catch (IOException eofe) {
    System.out.println(eofe);
    eofe.printStackTrace();
}
// Sort by score
if (score1 < score2) {
    return RecordComparator.PRECEDES;
}
else if (score1 > score2) {
    return RecordComparator.FOLLOWS;
}
else {
    return RecordComparator.EQUIVALENT;
}
    }
    /**
     * The constructor opens the underlying record store,
     * creating it if necessary.
     */
    public RMSGameScores()
    {
//
// Create a new record store for this example
//
try {
    recordStore = RecordStore.openRecordStore(“scores”, true);
}
catch (RecordStoreException rse) {
    System.out.println(rse);
    rse.printStackTrace();
}
    }
    /**
     * Add a new score to the storage.
     *
     * @param score the score to store.
     * @param playerName the name of the play achieving this score.
     */
    public void addScore(int score, String playerName)
    {
//
// Each score is stored in a separate record, formatted with
// the score, followed by the player name.
//
        int recId;  // returned by addRecord but not used
ByteArrayOutputStream baos = new ByteArrayOutputStream();
DataOutputStream outputStream = new DataOutputStream(baos);
try {
    // Push the score into a byte array.
    outputStream.writeInt(score);
    // Then push the player name.
    outputStream.writeUTF(playerName);
}
catch (IOException ioe) {
    System.out.println(ioe);
    ioe.printStackTrace();
}
// Extract the byte array
byte[] b = baos.toByteArray();
// Add it to the record store
try {
    recId = recordStore.addRecord(b, 0, b.length);
}
catch (RecordStoreException rse) {
    System.out.println(rse);
    rse.printStackTrace();
```

```
}
    }
    /**
     * A helper method for the printScores methods.
     */
    private void printScoresHelper(RecordEnumeration re)
    {
try {
    while(re.hasNextElement()) {
int id = re.nextRecordId();
ByteArrayInputStream bais = new ByteArrayInputStream(recordStore.getRecord(id));
DataInputStream inputStream = new DataInputStream(bais);
try {
    int score = inputStream.readInt();
    String playerName = inputStream.readUTF();
    System.out.println(playerName + " = " + score);
}
catch (EOFException eofe) {
    System.out.println(eofe);
    eofe.printStackTrace();
}
    }
}
catch (RecordStoreException rse) {
    System.out.println(rse);
    rse.printStackTrace();
}
catch (IOException ioe) {
    System.out.println(ioe);
    ioe.printStackTrace();
}
    }
    /**
     * This method prints all of the scores sorted by game score.
     */
    public void printScores()
    {
try {
    // Enumerate the records using the comparator implemented
    // above to sort by game score.
    RecordEnumeration re = recordStore.enumerateRecords(null, this,
true);
    printScoresHelper(re);
}
catch (RecordStoreException rse) {
    System.out.println(rse);
    rse.printStackTrace();
}
    }
    /**
     * This method prints all of the scores for a given player,
     * sorted by game score.
     */
    public void printScores(String playerName)
    {
try {
    // Enumerate the records using the comparator and filter
    // implemented above to sort by game score.
    RecordEnumeration re = recordStore.enumerateRecords(this, this,
true);
    printScoresHelper(re);
}
catch (RecordStoreException rse) {
    System.out.println(rse);
    rse.printStackTrace();
}
    }
    public static void main(String[] args)
```

```
    {
RMSGameScores rmsgs = new RMSGameScores();
rmsgs.addScore(100, “Alice”);
rmsgs.addScore(120, “Bill”);
rmsgs.addScore(80, “Candice”);
rmsgs.addScore(40, “Dean”);
rmsgs.addScore(200, “Ethel”);
rmsgs.addScore(110, “Farnsworth”);
rmsgs.addScore(220, “Farnsworth”);
System.out.println(“All scores”);
rmsgs.printScores();
System.out.println(“Farnsworth's scores”);
RMSGameScores.playerNameFilter = “Farnsworth”;
rmsgs.printScores(“Farnsworth”);
    }
}
```

Since: MIDP 1.0

Class Summary

Interfaces

RecordComparator 471	An interface defining a comparator which compares two records (in an implementation-defined manner) to see if they match or what their relative sort order is.
RecordEnumeration 473	An interface representing a bidirectional record store Record enumerator.
RecordFilter 478	An interface defining a filter which examines a record to see if it matches (based on an application-defined criteria).
RecordListener 479	A listener interface for receiving Record Changed/Added/Deleted events from a record store.

Classes

RecordStore 481	A class representing a record store.

Exceptions

InvalidRecordIDException 469	Thrown to indicate an operation could not be completed because the record ID was invalid.
RecordStoreException 493	Thrown to indicate a general exception occurred in a record store operation.
RecordStoreFullException 495	Thrown to indicate an operation could not be completed because the record store system storage is full.
RecordStoreNotFoundException 497	Thrown to indicate an operation could not be completed because the record store could not be found.
RecordStoreNotOpenException 499	Thrown to indicate that an operation was attempted on a closed record store.

javax.microedition.rms
InvalidRecordIDException

Declaration
```
public class InvalidRecordIDException extends RecordStoreException493

Object
  |
  +--Throwable
        |
        +--Exception
              |
              +--javax.microedition.rms.RecordStoreException493
                    |
                    +--javax.microedition.rms.InvalidRecordIDException
```

Description
Thrown to indicate an operation could not be completed because the record ID was invalid.

Since: MIDP 1.0

Member Summary	
Constructors	
	InvalidRecordIDException()469 InvalidRecordIDException(String message)470

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()`
Methods inherited from class `Throwable`
`getMessage(), printStackTrace(), toString()`

Constructors

InvalidRecordIDException()

Declaration:
```
public InvalidRecordIDException()
```

Description:
Constructs a new `InvalidRecordIDException` with no detail message.

InvalidRecordIDException(String)

Declaration:
```
public InvalidRecordIDException(String message)
```

Description:
Constructs a new `InvalidRecordIDException` with the specified detail message.

Parameters:
`message` - the detail message

javax.microedition.rms
RecordComparator

Declaration
```
public interface RecordComparator
```

Description
An interface defining a comparator which compares two records (in an implementation-defined manner) to see if they match or what their relative sort order is. The application implements this interface to compare two candidate records. The return value must indicate the ordering of the two records. The compare method is called by RecordEnumeration to sort and return records in an application specified order. For example:

```
RecordComparator c = new AddressRecordComparator();
if (c.compare(recordStore.getRecord(rec1), recordStore.getRecord(rec2))
== RecordComparator.PRECEDES)
return rec1;
```

Since: MIDP 1.0

Member Summary	
Fields	
static int	EQUIVALENT 471
static int	FOLLOWS 471
static int	PRECEDES 472
Methods	
int	compare(byte[] rec1, byte[] rec2) 472

Fields

EQUIVALENT

Declaration:
```
public static final int EQUIVALENT
```

Description:
EQUIVALENT means that in terms of search or sort order, the two records are the same. This does not necessarily mean that the two records are identical.

The value of EQUIVALENT is 0.

FOLLOWS

Declaration:
```
public static final int FOLLOWS
```

PRECEDES

Description:
FOLLOWS means that the left (first parameter) record *follows* the right (second parameter) record in terms of search or sort order.

The value of FOLLOWS is 1.

PRECEDES

Declaration:
```
public static final int PRECEDES
```

Description:
PRECEDES means that the left (first parameter) record *precedes* the right (second parameter) record in terms of search or sort order.

The value of PRECEDES is -1.

Methods

compare(byte[], byte[])

Declaration:
```
public int compare(byte[] rec1, byte[] rec2)
```

Description:
Returns `RecordComparator.PRECEDES` if rec1 precedes rec2 in sort order, or `RecordComparator.FOLLOWS` if rec1 follows rec2 in sort order, or `RecordComparator.EQUIVALENT` if rec1 and rec2 are equivalent in terms of sort order.

Parameters:

`rec1` - the first record to use for comparison. Within this method, the application must treat this parameter as read-only.

`rec2` - the second record to use for comparison. Within this method, the application must treat this parameter as read-only.

Returns: `RecordComparator.PRECEDES` if rec1 precedes rec2 in sort order, or `RecordComparator.FOLLOWS` if rec1 follows rec2 in sort order, or `RecordComparator.EQUIVALENT` if rec1 and rec2 are equivalent in terms of sort order

javax.microedition.rms
RecordEnumeration

Declaration
`public interface` **`RecordEnumeration`**

Description
An interface representing a bidirectional record store Record enumerator. The RecordEnumeration logically maintains a sequence of the recordId's of the records in a record store. The enumerator will iterate over all (or a subset, if an optional record filter has been supplied) of the records in an order determined by an optional record comparator.

By using an optional `RecordFilter`, a subset of the records can be chosen that match the supplied filter. This can be used for providing search capabilities.

By using an optional `RecordComparator`, the enumerator can index through the records in an order determined by the comparator. This can be used for providing sorting capabilities.

If, while indexing through the enumeration, some records are deleted from the record store, the recordId's returned by the enumeration may no longer represent valid records. To avoid this problem, the RecordEnumeration can optionally become a listener of the RecordStore and react to record additions and deletions by recreating its internal index. Use special care when using this option however, in that every record addition, change and deletion will cause the index to be rebuilt, which may have serious performance impacts.

If the RecordStore used by this RecordEnumeration is closed, this RecordEnumeration becomes invalid and all subsequent operations performed on it may give invalid results or throw a RecordStoreNotOpenException, even if the same RecordStore is later opened again. In addition, calls to `hasNextElement()` and `hasPreviousElement()` will return false.

The first call to `nextRecord()` returns the record data from the first record in the sequence. Subsequent calls to `nextRecord()` return the next consecutive record's data. To return the record data from the previous consecutive from any given point in the enumeration, call `previousRecord()`. On the other hand, if after creation, the first call is to `previousRecord()`, the record data of the last element of the enumeration will be returned. Each subsequent call to `previousRecord()` will step backwards through the sequence until the beginning is reached.

Final note, to do record store searches, create a RecordEnumeration with no RecordComparator, and an appropriate RecordFilter with the desired search criterion.

Since: MIDP 1.0

Member Summary

Methods

void	destroy()[474]
boolean	hasNextElement()[474]
boolean	hasPreviousElement()[474]
boolean	isKeptUpdated()[474]
void	keepUpdated(boolean keepUpdated)[474]
byte[]	nextRecord()[475]

Member Summary	
int	nextRecordId()[475]
int	numRecords()[476]
byte[]	previousRecord()[476]
int	previousRecordId()[476]
void	rebuild()[476]
void	reset()[477]

Methods

destroy()

Declaration:
```
public void destroy()
```

Description:
Frees internal resources used by this RecordEnumeration. MIDlets should call this method when they are done using a RecordEnumeration. If a MIDlet tries to use a RecordEnumeration after this method has been called, it will throw a `IllegalStateException`. Note that this method is used for manually aiding in the minimization of immediate resource requirements when this enumeration is no longer needed.

hasNextElement()

Declaration:
```
public boolean hasNextElement()
```

Description:
Returns true if more elements exist in the *next* direction.

Returns: true if more elements exist in the *next* direction

hasPreviousElement()

Declaration:
```
public boolean hasPreviousElement()
```

Description:
Returns true if more elements exist in the *previous* direction.

Returns: true if more elements exist in the *previous* direction

isKeptUpdated()

Declaration:
```
public boolean isKeptUpdated()
```

Description:
Returns true if the enumeration keeps its enumeration current with any changes in the records.

Returns: true if the enumeration keeps its enumeration current with any changes in the records

keepUpdated(boolean)

Declaration:
```
public void keepUpdated(boolean keepUpdated)
```

Description:
Used to set whether the enumeration will be keep its internal index up to date with the record store record additions/deletions/changes. Note that this should be used carefully due to the potential performance problems associated with maintaining the enumeration with every change.

Parameters:
`keepUpdated` - if true, the enumerator will keep its enumeration current with any changes in the records of the record store. Use with caution as there are possible performance consequences. Calling `keepUpdated(true)` has the same effect as calling `RecordEnumeration.rebuild`: the enumeration will be updated to reflect the current record set. If false the enumeration will not be kept current and may return recordIds for records that have been deleted or miss records that are added later. It may also return records out of order that have been modified after the enumeration was built. Note that any changes to records in the record store are accurately reflected when the record is later retrieved, either directly or through the enumeration. The thing that is risked by setting this parameter false is the filtering and sorting order of the enumeration when records are modified, added, or deleted.

See Also: `rebuild()` 476

nextRecord()

Declaration:
```
public byte[] nextRecord()
              throws InvalidRecordIDException, RecordStoreNotOpenException, RecordStoreEx
              ception
```

Description:
Returns a copy of the *next* record in this enumeration, where *next* is defined by the comparator and/or filter supplied in the constructor of this enumerator. The byte array returned is a copy of the record. Any changes made to this array will NOT be reflected in the record store. After calling this method, the enumeration is advanced to the next available record.

Returns: the next record in this enumeration

Throws:
`InvalidRecordIDException` 469 - when no more records are available. Subsequent calls to this method will continue to throw this exception until `reset()` has been called to reset the enumeration.

`RecordStoreNotOpenException` 499 - if the record store is not open

`RecordStoreException` 493 - if a general record store exception occurs

nextRecordId()

Declaration:
```
public int nextRecordId()
              throws InvalidRecordIDException
```

Description:
Returns the recordId of the *next* record in this enumeration, where *next* is defined by the comparator and/or filter supplied in the constructor of this enumerator. After calling this method, the enumeration is advanced to the next available record.

Returns: the recordId of the next record in this enumeration

Throws:
`InvalidRecordIDException` 469 - when no more records are available. Subsequent calls to this method will continue to throw this exception until `reset()` has been called to reset the enumeration.

numRecords()

Declaration:
```
public int numRecords()
```

Description:
Returns the number of records available in this enumeration's set. That is, the number of records that have matched the filter criterion. Note that this forces the RecordEnumeration to fully build the enumeration by applying the filter to all records, which may take a non-trivial amount of time if there are a lot of records in the record store.

Returns: the number of records available in this enumeration's set. That is, the number of records that have matched the filter criterion.

previousRecord()

Declaration:
```
public byte[] previousRecord()
            throws InvalidRecordIDException, RecordStoreNotOpenException, RecordStoreEx
            ception
```

Description:
Returns a copy of the *previous* record in this enumeration, where *previous* is defined by the comparator and/or filter supplied in the constructor of this enumerator. The byte array returned is a copy of the record. Any changes made to this array will NOT be reflected in the record store. After calling this method, the enumeration is advanced to the next (previous) available record.

Returns: the previous record in this enumeration

Throws:

`InvalidRecordIDException`[469] - when no more records are available. Subsequent calls to this method will continue to throw this exception until `reset()` has been called to reset the enumeration.

`RecordStoreNotOpenException`[499] - if the record store is not open

`RecordStoreException`[493] - if a general record store exception occurs.

previousRecordId()

Declaration:
```
public int previousRecordId()
            throws InvalidRecordIDException
```

Description:
Returns the recordId of the *previous* record in this enumeration, where *previous* is defined by the comparator and/or filter supplied in the constructor of this enumerator. After calling this method, the enumeration is advanced to the next (previous) available record.

Returns: the recordId of the previous record in this enumeration

Throws:

`InvalidRecordIDException`[469] - when no more records are available. Subsequent calls to this method will continue to throw this exception until `reset()` has been called to reset the enumeration.

rebuild()

Declaration:
```
public void rebuild()
```

Description:
Request that the enumeration be updated to reflect the current record set. Useful for when a MIDlet makes a number of changes to the record store, and then wants an existing RecordEnumeration to enumerate the new changes.

See Also: `keepUpdated(boolean)`*474*

reset()

Declaration:
```
public void reset()
```

Description:
Returns the enumeration index to the same state as right after the enumeration was created.

javax.microedition.rms
RecordFilter

Declaration
```
public interface RecordFilter
```

Description
An interface defining a filter which examines a record to see if it matches (based on an application-defined criteria). The application implements the match() method to select records to be returned by the RecordEnumeration. Returns true if the candidate record is selected by the RecordFilter. This interface is used in the record store for searching or subsetting records. For example:

```
RecordFilter f = new DateRecordFilter(); // class implements RecordFilter
if (f.matches(recordStore.getRecord(theRecordID)) == true)
  DoSomethingUseful(theRecordID);
```

Since: MIDP 1.0

Member Summary	
Methods	
	`boolean matches(byte[] candidate)` *478*

Methods

matches(byte[])

Declaration:
```
public boolean matches(byte[] candidate)
```

Description:
Returns true if the candidate matches the implemented criterion.

Parameters:
`candidate` - the record to consider. Within this method, the application must treat this parameter as read-only.

Returns: true if the candidate matches the implemented criterion

javax.microedition.rms
RecordListener

Declaration
```
public interface RecordListener
```

Description
A listener interface for receiving Record Changed/Added/Deleted events from a record store.

Since: MIDP 1.0

See Also: `RecordStore.addRecordListener(RecordListener)`[483]

Member Summary	
Methods	
void	`recordAdded(RecordStore recordStore, int recordId)`[479]
void	`recordChanged(RecordStore recordStore, int recordId)`[479]
void	`recordDeleted(RecordStore recordStore, int recordId)`[480]

Methods

recordAdded(RecordStore, int)

Declaration:
```
public void recordAdded(javax.microedition.rms.RecordStore[481] recordStore, int recordId)
```

Description:
Called when a record has been added to a record store.

Parameters:
`recordStore` - the RecordStore in which the record is stored

`recordId` - the recordId of the record that has been added

recordChanged(RecordStore, int)

Declaration:
```
public void recordChanged(javax.microedition.rms.RecordStore[481] recordStore,
            int recordId)
```

Description:
Called after a record in a record store has been changed. If the implementation of this method retrieves the record, it will receive the changed version.

Parameters:
`recordStore` - the RecordStore in which the record is stored

`recordId` - the recordId of the record that has been changed

recordDeleted(RecordStore, int)

Declaration:

```
public void recordDeleted(javax.microedition.rms.RecordStore[481] recordStore,
            int recordId)
```

Description:

Called after a record has been deleted from a record store. If the implementation of this method tries to retrieve the record from the record store, an InvalidRecordIDException will be thrown.

Parameters:

`recordStore` - the RecordStore in which the record was stored

`recordId` - the recordId of the record that has been deleted

javax.microedition.rms
RecordStore

Declaration

```
public class RecordStore

Object
  |
  +--javax.microedition.rms.RecordStore
```

Description

A class representing a record store. A record store consists of a collection of records which will remain persistent across multiple invocations of the MIDlet. The platform is responsible for making its best effort to maintain the integrity of the MIDlet's record stores throughout the normal use of the platform, including reboots, battery changes, etc.

Record stores are created in platform-dependent locations, which are not exposed to the MIDlets. The naming space for record stores is controlled at the MIDlet suite granularity. MIDlets within a MIDlet suite are allowed to create multiple record stores, as long as they are each given different names. When a MIDlet suite is removed from a platform all the record stores associated with its MIDlets will also be removed. MIDlets within a MIDlet suite can access each other's record stores directly. New APIs in MIDP 2.0 allow for the explicit sharing of record stores if the MIDlet creating the RecordStore chooses to give such permission.

Sharing is accomplished through the ability to name a RecordStore created by another MIDlet suite.

RecordStores are uniquely named using the unique name of the MIDlet suite plus the name of the RecordStore. MIDlet suites are identified by the MIDlet-Vendor and MIDlet-Name attributes from the application descriptor.

Access controls are defined when RecordStores to be shared are created. Access controls are enforced when RecordStores are opened. The access modes allow private use or shareable with any other MIDlet suite.

Record store names are case sensitive and may consist of any combination of between one and 32 Unicode characters inclusive. Record store names must be unique within the scope of a given MIDlet suite. In other words, MIDlets within a MIDlet suite are not allowed to create more than one record store with the same name, however a MIDlet in one MIDlet suite is allowed to have a record store with the same name as a MIDlet in another MIDlet suite. In that case, the record stores are still distinct and separate.

No locking operations are provided in this API. Record store implementations ensure that all individual record store operations are atomic, synchronous, and serialized, so no corruption will occur with multiple accesses. However, if a MIDlet uses multiple threads to access a record store, it is the MIDlet's responsibility to coordinate this access or unintended consequences may result. Similarly, if a platform performs transparent synchronization of a record store, it is the platform's responsibility to enforce exclusive access to the record store between the MIDlet and synchronization engine.

Records are uniquely identified within a given record store by their recordId, which is an integer value. This recordId is used as the primary key for the records. The first record created in a record store will have recordId equal to one (1). Each subsequent record added to a RecordStore will be assigned a recordId one greater than the record added before it. That is, if two records are added to a record store, and the first has a recordId of 'n', the next will have a recordId of 'n + 1'. MIDlets can create other sequences of the records in the RecordStore by using the `RecordEnumeration` class.

This record store uses long integers for time/date stamps, in the format used by System.currentTimeMillis(). The record store is time stamped with the last time it was modified. The record store also maintains a *version*

number, which is an integer that is incremented for each operation that modifies the contents of the RecordStore. These are useful for synchronization engines as well as other things.

Since: MIDP 1.0

Member Summary

Fields

static int	AUTHMODE_ANY[483]
static int	AUTHMODE_PRIVATE[483]

Methods

int	addRecord(byte[] data, int offset, int numBytes)[483]
void	addRecordListener(RecordListener listener)[483]
void	closeRecordStore()[484]
void	deleteRecord(int recordId)[484]
static void	deleteRecordStore(String recordStoreName)[484]
RecordEnumeration	enumerateRecords(RecordFilter filter, RecordComparator comparator, boolean keepUpdated)[485]
long	getLastModified()[486]
java.lang.String	getName()[486]
int	getNextRecordID()[486]
int	getNumRecords()[486]
byte[]	getRecord(int recordId)[487]
int	getRecord(int recordId, byte[] buffer, int offset)[487]
int	getRecordSize(int recordId)[488]
int	getSize()[488]
int	getSizeAvailable()[488]
int	getVersion()[488]
static java.lang.String[]	listRecordStores()[489]
static RecordStore	openRecordStore(String recordStoreName, boolean createIfNecessary)[489]
static RecordStore	openRecordStore(String recordStoreName, boolean createIfNecessary, int authmode, boolean writable)[490]
static RecordStore	openRecordStore(String recordStoreName, String vendorName, String suiteName)[490]
void	removeRecordListener(RecordListener listener)[491]
void	setMode(int authmode, boolean writable)[491]
void	setRecord(int recordId, byte[] newData, int offset, int numBytes)[492]

Inherited Member Summary

Methods inherited from class `Object`

`equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Fields

AUTHMODE_ANY

Declaration:
```
public static final int AUTHMODE_ANY
```

Description:
Authorization to allow access to any MIDlet suites. AUTHMODE_ANY has a value of 1.

AUTHMODE_PRIVATE

Declaration:
```
public static final int AUTHMODE_PRIVATE
```

Description:
Authorization to allow access only to the current MIDlet suite. AUTHMODE_PRIVATE has a value of 0.

Methods

addRecord(byte[], int, int)

Declaration:
```
public int addRecord(byte[] data, int offset, int numBytes)
            throws RecordStoreNotOpenException, RecordStoreException, RecordStoreFullEx
            ception
```

Description:
Adds a new record to the record store. The recordId for this new record is returned. This is a blocking atomic operation. The record is written to persistent storage before the method returns.

Parameters:

`data` - the data to be stored in this record. If the record is to have zero-length data (no data), this parameter may be null.

`offset` - the index into the data buffer of the first relevant byte for this record

`numBytes` - the number of bytes of the data buffer to use for this record (may be zero)

Returns: the recordId for the new record

Throws:

`RecordStoreNotOpenException`[499] - if the record store is not open

`RecordStoreException`[493] - if a different record store-related exception occurred

`RecordStoreFullException`[495] - if the operation cannot be completed because the record store has no more room

`SecurityException` - if the MIDlet has read-only access to the RecordStore

addRecordListener(RecordListener)

Declaration:
```
public void addRecordListener(javax.microedition.rms.RecordListener[479] listener)
```

Description:
Adds the specified RecordListener. If the specified listener is already registered, it will not be added a second time. When a record store is closed, all listeners are removed.

Parameters:
`listener` - the RecordChangedListener

See Also: `removeRecordListener(RecordListener)`[491]

closeRecordStore()

Declaration:
```
public void closeRecordStore()
            throws RecordStoreNotOpenException, RecordStoreException
```

Description:
This method is called when the MIDlet requests to have the record store closed. Note that the record store will not actually be closed until closeRecordStore() is called as many times as openRecordStore() was called. In other words, the MIDlet needs to make a balanced number of close calls as open calls before the record store is closed.

When the record store is closed, all listeners are removed and all RecordEnumerations associated with it become invalid. If the MIDlet attempts to perform operations on the RecordStore object after it has been closed, the methods will throw a RecordStoreNotOpenException.

Throws:
`RecordStoreNotOpenException`[499] - if the record store is not open

`RecordStoreException`[493] - if a different record store-related exception occurred

deleteRecord(int)

Declaration:
```
public void deleteRecord(int recordId)
            throws RecordStoreNotOpenException, InvalidRecordIDException, RecordStoreEx
            ception
```

Description:
The record is deleted from the record store. The recordId for this record is NOT reused.

Parameters:
`recordId` - the ID of the record to delete

Throws:
`RecordStoreNotOpenException`[499] - if the record store is not open

`InvalidRecordIDException`[469] - if the recordId is invalid

`RecordStoreException`[493] - if a general record store exception occurs

`SecurityException` - if the MIDlet has read-only access to the RecordStore

deleteRecordStore(String)

Declaration:
```
public static void deleteRecordStore(String recordStoreName)
            throws RecordStoreException, RecordStoreNotFoundException
```

Description:
Deletes the named record store. MIDlet suites are only allowed to delete their own record stores. If the named record store is open (by a MIDlet in this suite or a MIDlet in a different MIDlet suite) when this method is called, a RecordStoreException will be thrown. If the named record store does not exist a RecordStoreNotFoundException will be thrown. Calling this method does NOT result in recordDeleted calls to any registered listeners of this RecordStore.

Parameters:
`recordStoreName` - the MIDlet suite unique record store to delete

Throws:
`RecordStoreException`[493] - if a record store-related exception occurred

`RecordStoreNotFoundException`[497] - if the record store could not be found

enumerateRecords(RecordFilter, RecordComparator, boolean)

Declaration:
```
public javax.microedition.rms.RecordEnumeration[473]
            enumerateRecords(javax.microedition.rms.RecordFilter[478] filter,
            javax.microedition.rms.RecordComparator[471] comparator, boolean keepUpdated)
            throws RecordStoreNotOpenException
```

Description:
Returns an enumeration for traversing a set of records in the record store in an optionally specified order.

The filter, if non-null, will be used to determine what subset of the record store records will be used.

The comparator, if non-null, will be used to determine the order in which the records are returned.

If both the filter and comparator is null, the enumeration will traverse all records in the record store in an undefined order. This is the most efficient way to traverse all of the records in a record store. If a filter is used with a null comparator, the enumeration will traverse the filtered records in an undefined order. The first call to `RecordEnumeration.nextRecord()` returns the record data from the first record in the sequence. Subsequent calls to `RecordEnumeration.nextRecord()` return the next consecutive record's data. To return the record data from the previous consecutive from any given point in the enumeration, call `previousRecord()`. On the other hand, if after creation the first call is to `previousRecord()`, the record data of the last element of the enumeration will be returned. Each subsequent call to `previousRecord()` will step backwards through the sequence.

Parameters:
`filter` - if non-null, will be used to determine what subset of the record store records will be used

`comparator` - if non-null, will be used to determine the order in which the records are returned

`keepUpdated` - if true, the enumerator will keep its enumeration current with any changes in the records of the record store. Use with caution as there are possible performance consequences. If false the enumeration will not be kept current and may return recordIds for records that have been deleted or miss records that are added later. It may also return records out of order that have been modified after the enumeration was built. Note that any changes to records in the record store are accurately reflected when the record is later retrieved, either directly or through the enumeration. The thing that is risked by setting this parameter false is the filtering and sorting order of the enumeration when records are modified, added, or deleted.

Returns: an enumeration for traversing a set of records in the record store in an optionally specified order

Throws:
`RecordStoreNotOpenException`[499] - if the record store is not open

See Also: `RecordEnumeration.rebuild()`[476]

getLastModified()

Declaration:

```
public long getLastModified()
            throws RecordStoreNotOpenException
```

Description:
Returns the last time the record store was modified, in the format used by System.currentTimeMillis().

Returns: the last time the record store was modified, in the format used by System.currentTimeMillis()

Throws:

`RecordStoreNotOpenException`[499] - if the record store is not open

getName()

Declaration:

```
public String getName()
            throws RecordStoreNotOpenException
```

Description:
Returns the name of this RecordStore.

Returns: the name of this RecordStore

Throws:

`RecordStoreNotOpenException`[499] - if the record store is not open

getNextRecordID()

Declaration:

```
public int getNextRecordID()
            throws RecordStoreNotOpenException, RecordStoreException
```

Description:
Returns the recordId of the next record to be added to the record store. This can be useful for setting up pseudo-relational relationships. That is, if you have two or more record stores whose records need to refer to one another, you can predetermine the recordIds of the records that will be created in one record store, before populating the fields and allocating the record in another record store. Note that the recordId returned is only valid while the record store remains open and until a call to `addRecord()`.

Returns: the recordId of the next record to be added to the record store

Throws:

`RecordStoreNotOpenException`[499] - if the record store is not open

`RecordStoreException`[493] - if a different record store-related exception occurred

getNumRecords()

Declaration:

```
public int getNumRecords()
            throws RecordStoreNotOpenException
```

Description:
Returns the number of records currently in the record store.

Returns: the number of records currently in the record store

Throws:

`RecordStoreNotOpenException`[499] - if the record store is not open

getRecord(int)

Declaration:

```
public byte[] getRecord(int recordId)
            throws RecordStoreNotOpenException, InvalidRecordIDException, RecordStoreEx
            ception
```

Description:

Returns a copy of the data stored in the given record.

Parameters:

`recordId` - the ID of the record to use in this operation

Returns: the data stored in the given record. Note that if the record has no data, this method will return null.

Throws:

`RecordStoreNotOpenException`[499] - if the record store is not open

`InvalidRecordIDException`[469] - if the recordId is invalid

`RecordStoreException`[493] - if a general record store exception occurs

See Also: `setRecord(int, byte[], int, int)`[492]

getRecord(int, byte[], int)

Declaration:

```
public int getRecord(int recordId, byte[] buffer, int offset)
            throws RecordStoreNotOpenException, InvalidRecordIDException, RecordStoreEx
            ception
```

Description:

Returns the data stored in the given record.

Parameters:

`recordId` - the ID of the record to use in this operation

`buffer` - the byte array in which to copy the data

`offset` - the index into the buffer in which to start copying

Returns: the number of bytes copied into the buffer, starting at index `offset`

Throws:

`RecordStoreNotOpenException`[499] - if the record store is not open

`InvalidRecordIDException`[469] - if the recordId is invalid

`RecordStoreException`[493] - if a general record store exception occurs

`ArrayIndexOutOfBoundsException` - if the record is larger than the buffer supplied

See Also: `setRecord(int, byte[], int, int)`[492]

getRecordSize(int)

Declaration:

```
public int getRecordSize(int recordId)
            throws RecordStoreNotOpenException, InvalidRecordIDException, RecordStoreEx
            ception
```

Description:
Returns the size (in bytes) of the MIDlet data available in the given record.

Parameters:
`recordId` - the ID of the record to use in this operation

Returns: the size (in bytes) of the MIDlet data available in the given record

Throws:
`RecordStoreNotOpenException`[499] - if the record store is not open

`InvalidRecordIDException`[469] - if the recordId is invalid

`RecordStoreException`[493] - if a general record store exception occurs

getSize()

Declaration:

```
public int getSize()
            throws RecordStoreNotOpenException
```

Description:
Returns the amount of space, in bytes, that the record store occupies. The size returned includes any overhead associated with the implementation, such as the data structures used to hold the state of the record store, etc.

Returns: the size of the record store in bytes

Throws:
`RecordStoreNotOpenException`[499] - if the record store is not open

getSizeAvailable()

Declaration:

```
public int getSizeAvailable()
            throws RecordStoreNotOpenException
```

Description:
Returns the amount of additional room (in bytes) available for this record store to grow. Note that this is not necessarily the amount of extra MIDlet-level data which can be stored, as implementations may store additional data structures with each record to support integration with native applications, synchronization, etc.

Returns: the amount of additional room (in bytes) available for this record store to grow

Throws:
`RecordStoreNotOpenException`[499] - if the record store is not open

getVersion()

Declaration:

```
public int getVersion()
            throws RecordStoreNotOpenException
```

Description:
Each time a record store is modified (by `addRecord`, `setRecord`, or `deleteRecord` methods) its *version* is incremented. This can be used by MIDlets to quickly tell if anything has been modified. The initial version number is implementation dependent. The increment is a positive integer greater than 0. The version number increases only when the RecordStore is updated. The increment value need not be constant and may vary with each update.

Returns: the current record store version

Throws:
`RecordStoreNotOpenException`[499] - if the record store is not open

listRecordStores()

Declaration:
```
public static String[] listRecordStores()
```

Description:
Returns an array of the names of record stores owned by the MIDlet suite. Note that if the MIDlet suite does not have any record stores, this function will return null. The order of RecordStore names returned is implementation dependent.

Returns: array of the names of record stores owned by the MIDlet suite. Note that if the MIDlet suite does not have any record stores, this function will return null.

openRecordStore(String, boolean)

Declaration:
```
public static javax.microedition.rms.RecordStore[481] openRecordStore(String
            recordStoreName, boolean createIfNecessary)
            throws RecordStoreException, RecordStoreFullException, RecordStoreNotFoundE
            xception
```

Description:
Open (and possibly create) a record store associated with the given MIDlet suite. If this method is called by a MIDlet when the record store is already open by a MIDlet in the MIDlet suite, this method returns a reference to the same RecordStore object.

Parameters:
`recordStoreName` - the MIDlet suite unique name for the record store, consisting of between one and 32 Unicode characters inclusive.

`createIfNecessary` - if true, the record store will be created if necessary

Returns: `RecordStore` object for the record store

Throws:
`RecordStoreException`[493] - if a record store-related exception occurred

`RecordStoreNotFoundException`[497] - if the record store could not be found

`RecordStoreFullException`[495] - if the operation cannot be completed because the record store is full

`IllegalArgumentException` - if recordStoreName is invalid

openRecordStore(String, boolean, int, boolean)

Declaration:

```
public static javax.microedition.rms.RecordStore481 openRecordStore(String
            recordStoreName, boolean createIfNecessary, int authmode, boolean writable)
            throws RecordStoreException, RecordStoreFullException, RecordStoreNotFoundE
            xception
```

Description:

Open (and possibly create) a record store that can be shared with other MIDlet suites. The RecordStore is owned by the current MIDlet suite. The authorization mode is set when the record store is created, as follows:

- `AUTHMODE_PRIVATE` - Only allows the MIDlet suite that created the RecordStore to access it. This case behaves identically to `openRecordStore(recordStoreName, createIfNecessary)`.
- `AUTHMODE_ANY` - Allows any MIDlet to access the RecordStore. Note that this makes your recordStore accessible by any other MIDlet on the device. This could have privacy and security issues depending on the data being shared. Please use carefully.

The owning MIDlet suite may always access the RecordStore and always has access to write and update the store.

If this method is called by a MIDlet when the record store is already open by a MIDlet in the MIDlet suite, this method returns a reference to the same RecordStore object.

Parameters:

`recordStoreName` - the MIDlet suite unique name for the record store, consisting of between one and 32 Unicode characters inclusive.

`createIfNecessary` - if true, the record store will be created if necessary

`authmode` - the mode under which to check or create access. Must be one of AUTHMODE_PRIVATE or AUTHMODE_ANY. This argument is ignored if the RecordStore exists.

`writable` - true if the RecordStore is to be writable by other MIDlet suites that are granted access. This argument is ignored if the RecordStore exists.

Returns: `RecordStore` object for the record store

Throws:

`RecordStoreException`493 - if a record store-related exception occurred

`RecordStoreNotFoundException`497 - if the record store could not be found

`RecordStoreFullException`495 - if the operation cannot be completed because the record store is full

`IllegalArgumentException` - if authmode or recordStoreName is invalid

Since: MIDP 2.0

openRecordStore(String, String, String)

Declaration:

```
public static javax.microedition.rms.RecordStore481 openRecordStore(String
            recordStoreName, String vendorName, String suiteName)
            throws RecordStoreException, RecordStoreNotFoundException
```

Description:
Open a record store associated with the named MIDlet suite. The MIDlet suite is identified by MIDlet vendor and MIDlet name. Access is granted only if the authorization mode of the RecordStore allows access by the current MIDlet suite. Access is limited by the authorization mode set when the record store was created:

- `AUTHMODE_PRIVATE` - Succeeds only if vendorName and suiteName identify the current MIDlet suite; this case behaves identically to `openRecordStore(recordStoreName, createIfNecessary)`.
- `AUTHMODE_ANY` - Always succeeds. Note that this makes your recordStore accessible by any other MIDlet on the device. This could have privacy and security issues depending on the data being shared. Please use carefully. Untrusted MIDlet suites are allowed to share data but this is not recommended. The authenticity of the origin of untrusted MIDlet suites cannot be verified so shared data may be used unscrupulously.

If this method is called by a MIDlet when the record store is already open by a MIDlet in the MIDlet suite, this method returns a reference to the same RecordStore object.

If a MIDlet calls this method to open a record store from its own suite, the behavior is identical to calling: `openRecordStore(recordStoreName, false)`[489]

Parameters:
`recordStoreName` - the MIDlet suite unique name for the record store, consisting of between one and 32 Unicode characters inclusive.

`vendorName` - the vendor of the owning MIDlet suite

`suiteName` - the name of the MIDlet suite

Returns: `RecordStore` object for the record store

Throws:
`RecordStoreException`[493] - if a record store-related exception occurred

`RecordStoreNotFoundException`[497] - if the record store could not be found

`SecurityException` - if this MIDlet Suite is not allowed to open the specified RecordStore.

`IllegalArgumentException` - if recordStoreName is invalid

Since: MIDP 2.0

removeRecordListener(RecordListener)

Declaration:
```
public void removeRecordListener(javax.microedition.rms.RecordListener[479] listener)
```

Description:
Removes the specified RecordListener. If the specified listener is not registered, this method does nothing.

Parameters:
`listener` - the RecordChangedListener

See Also: `addRecordListener(RecordListener)`[483]

setMode(int, boolean)

Declaration:
```
public void setMode(int authmode, boolean writable)
            throws RecordStoreException
```

setRecord(int, byte[], int, int)

Description:
Changes the access mode for this RecordStore. The authorization mode choices are:

- `AUTHMODE_PRIVATE` - Only allows the MIDlet suite that created the RecordStore to access it. This case behaves identically to `openRecordStore(recordStoreName, createIfNecessary)`.
- `AUTHMODE_ANY` - Allows any MIDlet to access the RecordStore. Note that this makes your recordStore accessible by any other MIDlet on the device. This could have privacy and security issues depending on the data being shared. Please use carefully.

The owning MIDlet suite may always access the RecordStore and always has access to write and update the store. Only the owning MIDlet suite can change the mode of a RecordStore.

Parameters:

`authmode` - the mode under which to check or create access. Must be one of AUTHMODE_PRIVATE or AUTHMODE_ANY.

`writable` - true if the RecordStore is to be writable by other MIDlet suites that are granted access

Throws:

`RecordStoreException`[493] - if a record store-related exception occurred

`SecurityException` - if this MIDlet Suite is not allowed to change the mode of the RecordStore

`IllegalArgumentException` - if authmode is invalid

Since: MIDP 2.0

setRecord(int, byte[], int, int)

Declaration:

```
public void setRecord(int recordId, byte[] newData, int offset, int numBytes)
            throws RecordStoreNotOpenException, InvalidRecordIDException, RecordStoreEx
            ception, RecordStoreFullException
```

Description:
Sets the data in the given record to that passed in. After this method returns, a call to `getRecord(int recordId)` will return an array of numBytes size containing the data supplied here.

Parameters:

`recordId` - the ID of the record to use in this operation

`newData` - the new data to store in the record

`offset` - the index into the data buffer of the first relevant byte for this record

`numBytes` - the number of bytes of the data buffer to use for this record

Throws:

`RecordStoreNotOpenException`[499] - if the record store is not open

`InvalidRecordIDException`[469] - if the recordId is invalid

`RecordStoreException`[493] - if a general record store exception occurs

`RecordStoreFullException`[495] - if the operation cannot be completed because the record store has no more room

`SecurityException` - if the MIDlet has read-only access to the RecordStore

See Also: `getRecord(int, byte[], int)`[487]

javax.microedition.rms
RecordStoreException

Declaration
```
public class RecordStoreException extends Exception

Object
  |
  +--Throwable
        |
        +--Exception
              |
              +--javax.microedition.rms.RecordStoreException
```

Direct Known Subclasses: InvalidRecordIDException[469], RecordStoreFullException[495], RecordStoreNotFoundException[497], RecordStoreNotOpenException[499]

Description
Thrown to indicate a general exception occurred in a record store operation.

Since: MIDP 1.0

Member Summary	
Constructors	
	RecordStoreException()[493] RecordStoreException(String message)[494]

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()`
Methods inherited from class `Throwable`
`getMessage(), printStackTrace(), toString()`

Constructors

RecordStoreException()

Declaration:
```
public RecordStoreException()
```

Description:
Constructs a new `RecordStoreException` with no detail message.

RecordStoreException(String)

Declaration:
```
public RecordStoreException(String message)
```

Description:
Constructs a new `RecordStoreException` with the specified detail message.

Parameters:
`message` - the detail message

javax.microedition.rms
RecordStoreFullException

Declaration
```
public class RecordStoreFullException extends RecordStoreException[493]

Object
  |
  +--Throwable
        |
        +--Exception
              |
              +--javax.microedition.rms.RecordStoreException[493]
                    |
                    +--javax.microedition.rms.RecordStoreFullException
```

Description
Thrown to indicate an operation could not be completed because the record store system storage is full.

Since: MIDP 1.0

Member Summary	
Constructors	
	`RecordStoreFullException()`[495] `RecordStoreFullException(String message)`[496]

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()`
Methods inherited from class `Throwable`
`getMessage(), printStackTrace(), toString()`

Constructors

RecordStoreFullException()

Declaration:
```
public RecordStoreFullException()
```

Description:
Constructs a new `RecordStoreFullException` with no detail message.

RecordStoreFullException(String)

Declaration:
```
public RecordStoreFullException(String message)
```

Description:
Constructs a new `RecordStoreFullException` with the specified detail message.

Parameters:
`message` - the detail message

javax.microedition.rms
RecordStoreNotFoundException

Declaration
```
public class RecordStoreNotFoundException extends RecordStoreException[493]

Object
  |
  +--Throwable
        |
        +--Exception
              |
              +--javax.microedition.rms.RecordStoreException[493]
                    |
                    +--javax.microedition.rms.RecordStoreNotFoundException
```

Description
Thrown to indicate an operation could not be completed because the record store could not be found.

Since: MIDP 1.0

Member Summary	
Constructors	
	`RecordStoreNotFoundException()`[497] `RecordStoreNotFoundException(String message)`[498]

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()`
Methods inherited from class `Throwable`
`getMessage(), printStackTrace(), toString()`

Constructors

RecordStoreNotFoundException()

Declaration:
```
public RecordStoreNotFoundException()
```

Description:
Constructs a new `RecordStoreNotFoundException` with no detail message.

RecordStoreNotFoundException(String)

Declaration:
```
public RecordStoreNotFoundException(String message)
```

Description:
Constructs a new `RecordStoreNotFoundException` with the specified detail message.

Parameters:
`message` - the detail message

javax.microedition.rms
RecordStoreNotOpenException

Declaration
```
public class RecordStoreNotOpenException extends RecordStoreException493

Object
  |
  +--Throwable
        |
        +--Exception
              |
              +--javax.microedition.rms.RecordStoreException493
                    |
                    +--javax.microedition.rms.RecordStoreNotOpenException
```

Description
Thrown to indicate that an operation was attempted on a closed record store.

Since: MIDP 1.0

Member Summary	
Constructors	
	RecordStoreNotOpenException()499 RecordStoreNotOpenException(String message)500

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()`
Methods inherited from class `Throwable`
`getMessage(), printStackTrace(), toString()`

Constructors

RecordStoreNotOpenException()

Declaration:
```
public RecordStoreNotOpenException()
```

Description:
Constructs a new `RecordStoreNotOpenException` with no detail message.

RecordStoreNotOpenException(String)

Declaration:
```
public RecordStoreNotOpenException(String message)
```

Description:
Constructs a new `RecordStoreNotOpenException` with the specified detail message.

Parameters:
`message` - the detail message

CHAPTER 15

The Recommended Security Policy for GSM/UMTS Compliant Devices

Addendum to the Mobile Information Device Profile version 2.0

Scope of This Document

This addendum is informative. However, all implementations of MIDP 2.0 on GSM/UMTS compliant devices are expected to comply with this addendum.

MIDP 2.0 defines the framework for authenticating the source of a MIDlet suite and authorizing the MIDlet suite to perform protected functions by granting permissions it may have requested based on the security policy on the device. It also identifies functions that are deemed security vulnerable and defines permissions for those protected functions. Additionally, MIDP 2.0 specifies the common rules for APIs that can be used together with the MIDP but are specified outside the MIDP. MIDP 2.0 specification does not mandate a single trust model but rather allows the model to accord with the device trust policy.

The purpose of this addendum is to extend the base MIDlet suite security framework defined in MIDP 2.0 and to define the following areas:

- The required trust model for GSM/UMTS compliant devices
- The domain number and structure, as reflected in the device security policy
- The mechanism of reading root keys from sources external to the device
- Capabilities of MIDlets based on permissions defined by MIDP 2.0 and other JSRs
- MIDlet behaviour in the roaming network
- MIDlet behaviour when SIM/USIM is changed
- The use of user permission types
- Guidelines on user prompts and notifications

How This Specification Is Organized

This specification is organized as follows:

Sections 2 to 4 establish the relationship between the device security policy, different protection domains, and requirements concerning certificate storage on smart cards. Section 5 specifies the function groups and identifies

the permissions and the APIs that need to be protected using the MIDP 2.0 security framework. Sections 6 and 7 specify rules that must be followed when permissions are granted, and also requirements of user notifications. Finally Section 8 specifies the MIDlet behaviour during roaming and after changing the smart card.

References

1. Connected Limited Device Configuration (CLDC)
   http://jcp.org/jsr/detail/30.jsp (http://jcp.org/jsr/detail/30.jsp)
2. Mobile Information Device Profile (MIDP) 2.0
   http://jcp.org/jsr/detail/118.jsp (http://jcp.org/jsr/detail/118.jsp)
3. HTTP 1.1 Specification
   http://www.ietf.org/rfc/rfc2616.txt (http://www.ietf.org/rfc/rfc2616.txt)
4. WAP Wireless Identity Module Specification (WIM) WAP-260-WIM-20010712-a
   http://www.wapforum.org/what/technical.htm (http://www.wapforum.org/what/technical.htm)
5. WAP Smart Card Provisioning (SCPROV) WAP-186-ProvSC-20010710-a
   http://www.wapforum.org/what/technical.htm (http://www.wapforum.org/what/technical.htm)
6. PKCS#15 v.1.1
   http://www.rsasecurity.com/rsalabs/pkcs/pkcs-15/ (http://www.rsasecurity.com/rsalabs/pkcs/pkcs-15/)
7. USIM, 3GPP TS 31.102: “Characteristics of the USIM applications”
   http://www.3gpp.org (http://www.3gpp.org/)
8. RFC3280
   http://www.ietf.org/rfc (http://www.ietf.org/rfc)

1 General

GSM/UMTS compliant devices implementing this Recommended Security Policy MUST follow the security framework specified in the MIDP 2.0. Additionally, devices that support trusted MIDlets MUST follow the PKI-based authentication scheme as defined in MIDP 2.0 specification.

2 Protection Domains in the Device Security Policy

A protection domain is a way to differentiate between downloaded MIDlet suites based on the entity that signed the MIDlet suite, and to grant or make available to a MIDlet suite a set of permissions. A domain binds a Protection Domain Root Certificate to a set of permissions. The permissions are specified in the protection domain security policy, a policy has as many entries as there are protection domains available on the device. A domain can exist only for a Protection Domain Root Certificate that contain the `id-kp-codeSigning` extended key usage extension. MIDlet suites that authenticate to a trusted Protection Domain Root Certificate are treated as trusted, and assigned to the corresponding protection domain. A MIDlet suite cannot belong to more than one protection domain. The representation of a domain and its security policy is implementation specific.

3 Protection Domains and the Permissions Framework

This document specifies two different requirements as to how the MIDP permissions framework should be used, depending on the protection domain an application executes.

Manufacturer and Operator Domains˝ MIDlet suites SHOULD seek permission from the user when accessing security vulnerable APIs and functions. Permissions defined by MIDP 2.0 and other APIs give the guidelines of which functions are seen as security vulnerable and need protection. It is expected that operator

trusted MIDlets will give prompts and notifications to the user when accessing these security protected functions as required.

Third Party and Untrusted Domains ˝ The device implementation is responsible for prompting the user according to the security policies specified in Tables 1 through 6 in this document.

3.1 Manufacturer Domain

The trusted manufacturer Protection Domain Root Certificate is used to verify manufacturer MIDlet suites. The manufacturer Protection Domain Root Certificate MUST be mapped on to the security policy for the manufacturer domain on the device. A device MUST support the security policy for the manufacturer domain.

If the manufacturer Protection Domain Root Certificate is NOT available on the device, the manufacturer domain MUST be disabled.

The manufacturer Protection Domain Root Certificate can only be deleted or modified by the manufacturer, who may use an update mechanism whose details are outside the scope of this specification. Any new or updated manufacturer Protection Domain Root Certificate MUST be associated with the security policy for the manufacturer domain on the device. MIDlet suites verified by a previous manufacturer Protection Domain Root Certificate MUST be disabled.

Permissions in the Manufacturer domain are all marked as *Allowed* (see MIDP 2.0 for the definition). Permissions granted by the Manufacturer domain as *Allowed* imply that downloaded and authenticated manufacturer MIDlets suites perform consistently with MIDlets suites pre-installed by the manufacturer in terms of security and prompts to the user whenever events that require user acknowledgement occur. Manufacturer MIDlets SHOULD seek permission from the user when accessing security vulnerable APIs and functions. Permissions defined by MIDP 2.0 and other APIs give the guidelines of which functions are seen as security vulnerable and need protection.

At MIDlet suite installation, an implementation MUST present the user with the *Organisation* and *Country* fields within the Subject field of the manufacturer Protection Domain Root Certificate if the *Organisation* and *Country* fields are present. If the *Organisation* and *Country* fields are absent, the implementation MUST present the user with other appropriate information from the Subject field. An implementation MAY also present the user with additional information in the Subject field other than *Organisation* and *Country* in all cases. This user notification MUST take place at application installation.

The Manufacturer domain imposes no restriction on the capabilities specified in the MIDP 2.0 and other JSRs.

3.2 Operator Domain

A trusted operator Protection Domain Root Certificate is used to verify operator MIDlet suites. There is no explicit limitation on the number of operator trusted Protection Domain Root Certificates available at the specified location in the SIM, USIM or WIM. Trusted operator Protection Domain Root Certificates MUST be mapped on to the security policy for the Operator domain on the device. A device MUST support the security policy for the Operator domain.

If an operator Protection Domain Root Certificate is NOT available on the specified location in the SIM, USIM or WIM; the operator domain MUST be disabled.

Trusted Protection Domain Root Certificates are read from the Certificate Directory File (CDF) for trusted certificates [WIM]. Protection Domain Root Certificate found in the trustedCertificates file on the WIM are mapped onto the Operator domain or onto the Trusted Third Party domain, depending on the trustedUsage field in the CommonCertificateAttributes associated with the certificate [PKCS#15]:

> If the trustedUsage field is present and contains the OID for key usage *˝iso(1)org(3)dod(6)internet(1)private(4)enterprises(1)sun(42) products(2)javaXMLsoftware(110)midp(2)spec(2)gsm-policy(2)operator(1)˝*, then the certificate is to be mapped onto the Operator domain.

If the trustedUsage field is not present, or does not contain the OID for key usage ″Operator Domain″, then the certificate is to be mapped onto the Trusted Third Party domain.

Operator trusted Protection Domain Root Certificates may be placed in the trustedCertificates Certificate Directory File (CDF) of a WIM, SIM, or USIM. If operator Protection Domain Root Certificates are stored directly on a SIM or USIM, that is, not under the WIM application, then they shall be stored in the EF trustedCertificates CDF located under DF(PKCS#15), as defined by [SCPROV]. Operator Protection Domain Root Certificates can be obtained only from the trusted CDF (the card holder can not update this directory) and not from any other directory of the smart card.

All operator Protection Domain Root Certificates MUST be mapped onto the same security policy for the operator domain on the device. The Operator domain cannot be deleted or modified by the user or any other party, except by a device provisioned capability.

A signed and authenticated MIDlet suite MUST be authorized to the Operator domain if the MIDlet suite was authenticated to the operator Protection Domain Root Certificate. The operator root public key MUST be obtained from a certificate in the trusted CDF of a currently inserted and enabled smart card and not from any other location on the smart card or on the device. At MIDlet suite installation, an implementation MUST present the user with the *Organisation* and *Country* fields within the *Subject* field of the operator Protection Domain Root Certificate if the *Organisation* and *Country* fields are present. If the *Organisation* and *Country* fields are absent, the implementation MUST present the user with other appropriate information from the *Subject* field. An implementation MAY also present the user with additional information in the *Subject* field other than *Organisation* and *Country* in all cases.This user notification MUST take place at application installation.

The security policy for the operator domain MUST contain all permissions implemented on the device as "*Allowed*". Permissions granted by the Operator domain as *Allowed* imply that downloaded and authenticated operator MIDlets suites perform consistently with other MIDlets suites installed by the operator in terms of security and prompts to the user whenever events that require user acknowledgement occur. Operator MIDlets SHOULD seek permission from the user when accessing security vulnerable APIs and functions. Permissions defined by MIDP 2.0 and other APIs provide guidelines as to which APIs and functions are seen as security vulnerable and need protection. The Operator domain imposes no restriction on the capabilities specified in the MIDP 2.0 and other JSRs.

MIDlet suites installed in the Operator domain MUST store, along with the application itself, a hash of the Protection Domain Root Certificate under which the signing certificate used to sign the application was issued. The hash algorithm to be used is the following, starting with the Protection Domain Root Certificate, compute the 20-byte SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits) of that certificate. This method is commonly used to compute key identifiers, especially to accelerate trust chain building [RFC3280, §4.2.1.2]. The implementation MUST NOT assume for optimization purposes that X.509 key identifiers or PKCS#15 labels are the correct value; and MUST compute the hash themselves. This hash MUST be used by the device to decide when a given MIDlet suite should be disabled, as specified in Section 8.

3.3 Trusted Third Party Domain

A trusted third party Protection Domain Root Certificate is used to verify third party MIDlet suites. There is no explicit limitation on the number of trusted third party Protection Domain Root Certificates available either on the device or at the specified location in the SIM, USIM or WIM (see section 3.2). A trusted third party Protection Domain Root Certificates MUST be mapped on to the security policy for the trusted third party domain on the device. A device MUST support the security policy for the trusted third party domain. If there are no trusted third party Protection Domain Root Certificates available either on the device or at the specified location in the SIM, USIM or WIM; the trusted third party domain MUST be disabled.

Third party Protection Domain Root Certificates downloaded after device manufacture MUST NOT be used for authentication of MIDlet suites. This does NOT prevent obtaining trusted third party Protection Domain Root Certificates from the specified location in the SIM, USIM, WIM.

At MIDlet suite installation, an implementation MUST present the user with the *Organisation* and *Country* fields within the *Subject* field of the signing certificate of a MIDlet suite if the *Organisation* and *Country* fields are present. If the *Organisation* and *Country* fields are absent, the implementation MUST present the user with other appropriate information from the *Subject* field. An implementation MAY also present the user with additional information in the *Subject* field other than *Organisation* and *Country* in all cases. This user notification MUST take place at MIDlet suite installation. When the user is prompted to grant permissions to an application, the prompt MUST identify the trusted source with the appropriate fields within *Subject* field of the signing certificate as stated above.

The user MUST be able to delete or disable trusted third party Protection Domain Root Certificates. If a third party Protection Domain Root Certificate is to be deleted, the implementation SHOULD warn the user of the consequence of the deletion adequately. The user MUST be able to enable a disabled third party Protection Domain Root Certificate. A disabled third party Protection Domain Root Certificate MUST NOT be used to verify downloaded MIDlet suites. Furthermore, if a third party Protection Domain Root Certificate is deleted or disabled (for example, revoked, deleted, or disabled by the user) the Third Party domain MUST no longer be associated with this Protection Domain Root Certificate. If the user chooses to delete or disable the Protection Domain Root Certificate, implementation may provide an option to delete the MIDlet suites authenticated to it.

The security policy for trusted third party domain MUST NOT granted any permissions on the device as *Allowed.* All permissions granted by the Third Party domainMUST be *User* permissions, that is, user interaction is required for permission to be granted. Table 1 specifies the function groups and the available user permission types for MIDlet suites in the Third Party domain. Tables 2 through 6 specify the mapping of permissions and APIs onto different function groups.

3.4 Untrusted Domain

MIDlets suites that are unsigned will belong to the Untrusted domain. The implementation MUST inform the user whenever a new MIDlet suite is installed in the Untrusted domain. The notification MUST indicate that the application does not come from a trusted source. The user must be able to make an informed decision based on the available information before granting permissions to an application.

When the user is prompted to grant permissions to an application, the prompt MUST indicate that the application does not come from a trusted source.

Untrusted MIDlets suites MUST NOT gain read access directly to PIM data through the API defined in JSR 075 (see Tables 1 and 3 in Section 5). Interactions between an untrusted application and the PIM data can be enabled, however, by implementations of the javax.microedition.lcdui package: when the application programmer sets the constraint TextField.PHONENUMBER, an implementation of the TextField class MAY propose that the user look up a number in his or her phone book and copy it to the TextField item. For example, when the TextField item has input focus, the user can access a menu to enter the phone book; when the user selects an entry in the phone book, the contents of the selected entry are “copied and pasted” into the TextField item.

Table 1 specifies the function groups and the available user permissions for MIDlets suites in the Untrusted domain. Tables 2 through 6 specify the mapping of permissions and APIs onto different function groups.

4 Remotely Located Security Policy

The MIDP 2.0 specification defines the generic format for a policy file that can be read from removable media. GSM/UMTS compliant devices are not expected to use it in the first phase, but rather to use security policy resident on the device. The possibility of remotely located security policy files is left for further consideration.

5 Permissions for Downloaded MIDlet Suites

5.1 Mapping MIDP 2.0 Permissions onto Function Groups in Protected Domains

A device with a small display may not be able to present all permissions to the user in a single configuration settings menu in a user friendly manner. Therefore the device is not required to present all individual permissions for user confirmation. Rather, a certain higher-level action triggered by the protected function should be brought to the user for acceptance. The high level functions presented to the user essentially capture and reflect the actions and consequences of the underlying individual permissions. The function groups are as follows:

Network/cost-related groups:

Phone Call ˝ the group represents permissions to any function that results in a voice call.

Net Access ˝ the group represents permissions to any function that results in an active network data connection (for example GSM, GPRS, UMTS, etc.); such functions must be mapped to this group.

Messaging ˝ the group represents permissions to any function that allows sending or receiving messages (for example, SMS, MMS, etc.)

Application Auto Invocation ˝ the group represents permissions to any function that allows a MIDlet suite to be invoked automatically (for example, push, timed MIDlets, etc.)

Local Connectivity ˝ the group represents permissions to any function that activates a local port for further connection (for example, COMM port, IrDa, Bluetooth, etc.)

User-privacy-related groups:

Multimedia recording ˝ the group represents permissions to any function that gives a MIDlet suite the ability to capture still images, or to record video or audio clips.

Read User Data Access ˝ the group represents permissions to any function that gives a MIDlet suite the ability to read a user's phone book, or any other data in a file or directory.

Write User Data Access ˝ the group represents permissions to any function that gives a MIDlet suite the ability to add or modify a user's phone book, or any other data in a file or directory.

Whenever new features are added to the MIDP they should be assigned to the appropriate function group. In addition, APIs that are specified elsewhere (that is, in other JSRs) but rely on the MIDP security framework should also be assigned to an appropriate function group. If none of the function groups defined in this section is able to capture the new feature and reflect it to the user adequately, however, then a new function group MUST be defined in this document.

If a new function group is to be added, the following should be taken into consideration: the group to be added MUST not introduce any redundancy to the existing groups, the new group MUST be capable of protecting a wide range of similar features. The latter requirement is to prevent introducing narrowly scoped groups.

It is the function groups and not the individual permissions that should be presented when the user is prompted. Furthermore, it is the function groups that should be presented to the user in the settings of a given MIDlet suite.

Table 1 presents the policy that must be enforced using the security framework as defined in MIDP 2.0. The table specifies the available permission settings for each function group defined. Settings that are effective at the time the MIDlet suite is invoked for the first time, and remain effective until the user changes them in the MIDlet suite's configuration menu, are called "default settings." Settings available to the user in the configuration menu, to which the user can change from a default setting, are called "other settings." Together, default and other settings form a pool of available configuration settings for the MIDlet suite. Default and other settings are presented for each function group and each protection domain. The naming of the function groups is implementation specific but MUST follow the guidelines of the function group names defined in this document as well as the definitions of these groups.

Tables 2 through 5 present individual permissions defined in the MIDP 2.0 and other JSRs, and map to the function groups specified in this section. An individual permission MUST occur in only one function group.

It is recommended that the manufacturer and operator trusted MIDlets suites adhere to the permission guidelines provided in the tables, and present appropriate prompts to the user for the functions identified as security protected.

Table 1: Function groups and user settings

Function group	**Trusted Third Party domain**		**Untrusted domain**	
Phone Call	default setting	Oneshot	default setting	Oneshot
	other settings	No	other settings	No
Net Access	default setting	Session	default setting	Oneshot
	other settings	Oneshot, Blanket, No	other settings	Session, No
Messaging	default setting	Oneshot	default setting	Oneshot
	other settings	No	other settings	No
Application Auto Invocation	default setting	Session	default setting	Session
	other settings	Oneshot, Session, Blanket, No	other settings	Oneshot, No
Local Connectivity	default setting	Session	default setting	Session
	other settings	Blanket, No	other settings	Blanket, No
Multimedia recording	default setting	Session	default setting	Oneshot
	other settings	Blanket, No	other settings	Session, No
Read User Data Access	default setting	Oneshot	default setting	No
	other settings	Session, Blanket, No	other settings	No
Write User Data Access	default setting	Oneshot	default setting	Oneshot
	other settings	Session, Blanket, No	other settings	No

The device MAY enhance and simplify the user experience by applying a single set of configuration settings (default or other), not just to a single MIDlet suite, but to all MIDlet suites for a given signer. This option MUST NOT compromise the function groups and available settings defined in Table 1. If such an option exists, the user will be prompted to save the settings and reuse them in future for MIDlets suites from the same source. Such a feature MAY also inform the user that a given source has already been accepted and has an alias to the saved configuration settings. For each trusted or untrusted application, the implementation MAY read requested permissions from the MIDlet-Permissions and MIDlet-PermissionsOpt attributes, notify the user which capability the application requires, and prompt the user to accept or reject installation of the application.

Blanket permission given for some combinations of Function groups can lead to higher risks for the user. For MIDlet suites in the Third Party domain the user MUST be notified of the higher risk involved and also acknowledge that this risk is accepted to allow such combinations to be set. The combination of Blanket permission in Function groups where this applies is:

- Any of Net Access, Messaging or Local Connectivity set to Blanket in combination with any of Multimedia recording or Read User Data Access set to Blanket

This restriction need not apply to the Untrusted domain, since these combinations would be forbidden according to table 1.

Additionally, the Blanket setting for Application Auto Invocation and the Blanket setting for Net Access are mutually exclusive. This constraint is to prevent a MIDlet suite from auto-invoking itself, then accessing a chargeable network without the user being aware. If the user attempts to set either the Application Auto Invocation or the Network Function group to “Blanket” when the other Function group is already in “Blanket” mode, the user MUST be prompted as to which of the two Function groups shall be granted “Blanket” and which Function group shall be granted “Session”.

For each Phone Call and Messaging action, the implementation MUST present the user with the destination phone number before the user approves the action. For the Messaging group, if the implementation maps a single API call to more than one message (that is, the implementation supports disassembly/reassembly), the implementation MUST present the user with the number of messages that will actually be sent out. This requirement is to ensure that the user always understands the network costs associated with running the program, whatever API calls are involved.

Table 2: Assigning permissions specified in MIDP 2.0 to function groups

MIDP 2.0 ˝JSR 118		
Permission	**Protocol**	**Function group**
javax.microedition.io.Connector.http	http	Net Access
javax.microedition.io.Connector.https	https	Net Access
javax.microedition.io.Connector.datagram	datagram	Net Access
javax.microedition.io.Connector.datagramreceiver	datagram server (without host)	Net Access
javax.microedition.io.Connector.socket	socket	Net Access
javax.microedition.io.Connector.serversocket	server socket (without host)	Net Access
javax.microedition.io.Connector.ssl	ssl	Net Access
javax.microedition.io.Connector.comm	comm	Local Connectivity
javax.microedition.io.PushRegistry	All	Application Auto Invocation

Table 3: Assigning proposed permissions and API calls specified in the Personal Information Management Package of the PDA Profile to function groups

PDAP PIM Package API (JSR75)		
Security Policy Identifier (Proposed Permission)	**Permitted Java API Calls**	**Function group**
javax.microedition.pim.PIM. contact.readonly	PIM.listContactLists() PIM.openContactList(READ_ONLY) PIM.openContactList(READ_ONLY, listName)	Read User Data Access
javax.microedition.pim.PIM. contact.readwrite	PIM.listContactLists() PIM.openContactList(READ_ONLY) PIM.openContactList(READ_WRITE) PIM.openContactList(READ_ONLY, listName) PIM.openContactList(READ_WRITE, listName)	Write User Data Access
javax.microedition.pim.PIM. event.readonly	PIM.listEventLists() PIM.openEventList(READ_ONLY) PIM.openEventList(READ_ONLY, listName)	Read User Data Access
javax.microedition.pim.PIM. event.readwrite	PIM.listEventLists() PIM.openEventList(READ_ONLY) PIM.openEventList(READ_WRITE) PIM.openEventList(READ_ONLY, listName) PIM.openEventList(READ_WRITE, listName)	Write User Data Access
javax.microedition.pim.PIM. todo.readonly	PIM.listToDoLists() PIM.openToDoList(READ_ONLY) PIM.openToDoList(READ_ONLY, listName)	Read User Data Access
javax.microedition.pim.PIM. todo.readwrite	PIM.listToDoLists() PIM.openToDoList(READ_ONLY) PIM.openToDoList(READ_WRITE) PIM.openToDoList(READ_ONLY, listName) PIM.openToDoList(READ_WRITE, listName)	Write User Data Access

Table 3 Editor's Note: The necessary permissions to protect the PIM API are not specified in the PIM package. This table will be updated once these changes are incorporated into the PIM API package.

The implementation MUST ensure that the user is informed of the nature of the user data an application has access to (for instance, events or to-do lists) before allowing the application access to these functions. Whenever a MIDlet adds, deletes or updates a PIM entry under the Oneshot permission type, the implementation MUST display it to the user for acknowledgement.

Table 4: Assigning proposed permissions and API calls specified in the Bluetooth API to function groups

Bluetooth API˝ JSR 82		
Security Policy Identifier (Proposed Permission)	**Permitted API calls**	**Function group**
javax.microedition.io.Connector.bluetooth.client	Connector.open(˝btspp://<server BD_ADDR>˝˝) Connector.open(˝btl2cap://<server BD_ADDR>˝˝)	Local Connectivity
javax.microedition.io.Connector.obex.client	Connector.open(˝btgoep://<server BD_ADDR>˝˝) Connector.open(˝irdaobex://discover˝˝) Connector.open(˝irdaobex://addr˝˝) Connector.open(˝irdaobex://conn˝˝) Connector.open(˝irdaobex://name˝˝)	Local Connectivity
javax.microedition.io.Connector.obex.client.tcp	Connector.open(˝tcpobex://<server IP_ADDR>˝˝)	Net Access
javax.microedition.io.Connector.bluetooth.server	Connector.open(˝btspp://localhost:˝˝) Connector.open(˝btl2cap://localhost:˝˝)	Local Connectivity
javax.microedition.io.Connector.obex.server	Connector.open(˝btgoep://localhost:˝˝) Connector.open(˝irdaobex://localhost:˝˝)	Local Connectivity
javax.microedition.io.Connector.obex.server.tcp	Connector.open(˝tcpobex://:<PORT>˝) Connector.open(˝tcpobex://˝)	Net Access

Table 4 Editor's Note: The permissions proposed for Bluetooth API are yet to be defined in JSR82.

Table 5: Assigning proposed permissions and API calls specified in the Wireless Messaging API to function groups

Wireless Messaging API ˝ JSR 120		
Security Policy Identifier (Proposed Permission)	**Permitted API calls**	**Function group**
javax.microedition.io.Connector.sms.send	Connector.open("sms://˝", WRITE) Connector.open("sms://˝", WRITE, Bool)	Messaging
javax.microedition.io.Connector.sms.receive	Connector.open("sms://˝", READ) Connector.open("sms://˝", READ, Bool)	Messaging
javax.microedition.io.Connector.sms	Connector.open("sms://˝") Connector.open("sms://˝", READ) Connector.open("sms://˝", READ, Bool) Connector.open("sms://˝", WRITE) Connector.open("sms://˝", WRITE, Bool) Connector.open("sms://˝", READ_WRITE) Connector.open("sms://˝", READ_WRITE, Bool)	Messaging
javax.microedition.io.Connector.cbs.receive	Connector.open("cbs://˝") Connector.open("cbs://˝", READ) Connector.open("cbs://˝", READ, Bool)	Messaging/p>

Table 5 Editor's Note: The permissions for Wireless Messaging API are yet to be defined in JSR120.

Table 6: Assigning proposed permissions and API calls specified in the Mobile Media API to function groups

Mobile Media API˝JSR 135		
Security Policy Identifier (Proposed Permissions)	**Permitted API calls**	**Function group**
javax.microedition.media.RecordControl.startRecord	RecordControl.startRecord ( )	Multimedia recording
javax.microedition.media.VideoControl.getSnapshot	VideoControl.getSnapshot (˝)	Multimedia recording

Table 6 Editor's Note: The permissions for Mobile Media API are yet to be defined in JSR135.

Implementations MUST ensure that I/O access from the Mobile Media API follows the same security requirements as the Generic Connection Framework, as specified in the package documentation for javax.microedition.io. Example methods include javax.microedition.media.Player.start, javax.microedition.media.Player.prefetch, etc. When these methods are used to fetch the content for the player via an HTTP connection, the implementation MUST enforce the security requirements specified for HTTP.

5.2 Implementation notes:

When the user grants permission to a function group, this action effectively grants access to all individual permissions under this function group.

An implementation MUST guarantee that a SecurityException is thrown when the caller does not have the appropriate security permissions.

If a messaging group is granted a Oneshot permission, it translates into a Blanket permission for javax.microedition.io.Connector.sms and javax.microedition.io.Connector.cbs, as well as to permissions that enable receiving the messages. Permission for sending the messages is still Oneshot, however; that is, the user grants permission to each message sent out by the MIDlet suite within an open connection. The same applies to the Session permission: functions related to sending the messages get Session permission, but other functions get Blanket permission.“ Blanket permission and No permission granted to the Messaging group apply to all individual permissions under this group.

If a MIDlet uses the capabilities defined in MIDP and other APIs, the following rules MUST apply:

- All the external API functions that need to be protected by MIDP 2.0 security framework MUST have permissions defined in the subsequent JSRs, and follow the naming rules identified in the MIDP 2.0 Specification, titled “Security for MIDP Applications.”
- The functions that are not deemed security-protected by specification can be accessed explicitly by trusted and untrusted MIDlet suites, as per general MIDP security rules.
- If an external API does not define permissions for security-protected functions because the API specification is released earlier than MIDP 2.0, any functions that relate to network access MUST still have the user prompt implemented by the device.
- A device cannot access the network without appropriate user notification.
- All licensee open classes MUST adhere to the permission framework as defined in this document.

6 Permissions Granted to a MIDlet Suite by the Authorization Mechanism

As defined in the "Security for MIDP Applications" section of the MIDP 2.0 specification, MIDlet suite permissions are effectively the intersection of the domain permissions Midlet-Permission and Midlet-Permission-Opt found in the JAR manifest. The way in which a MIDlet suite's granted permissions are presented to the user is implementation-specific, but the following rules must apply:

- The user must be able to change the default permission setting (provided they in accordance with the implementation notes in section 5.2) to any setting available for a given MIDlet suite permission, with default and available sets of user permission types provided as guides in the tables in Section 5. This latitude will allow the user to upgrade or downgrade the default permissions as required.
- If MIDlet permissions are grouped according to capabilities they represent, permissions granted to a MIDlet suite will be rendered into the function groups to be presented to the user. If function grouping is used, default permission applies to the whole group of permissions under the group. So does the available set of types of user permissions. If the default permission is changed, the change is effective for the entire group at once rather than to the individual permissions under this group.
- A function group cannot be a union of permissions with different default settings and other settings. Therefore the tables in Section 5 follow the convention of having the same default and available settings for all permissions in a single function group. This rule must be taken into account when designing new permissions and policies.

A device MUST maintain security related data for each installed MIDlet suite, in addition to generic MIDlet suite information such as MIDlet suite name and version number. The data MUST include at least the following:

- The signer of the MIDlet suite, i.e. the *Subject* field in the signing certificate, if the MIDlet suite was signed. At least MIDlet-Vendor MUST be stored along with the installed MIDlet suite.
- Data related to the Protection Domain Root Certificate a signed MIDlet was authenticated to; at minimum the Subject field of the Protection Domain Root Certificate.
- Data related to a signer certificate that signed the MIDlet suite; at minimum the certificate's Subject, Issuer, and Serial Number fields. (As an alternative, a device may store the entire certificate chain that came with the MIDlet descriptor file.)
- A list of permissions granted to the MIDlet suite.

A device MUST be able to present information related to the application signer in a user-friendly manner.

7 User Prompts and Notifications

The following rules MUST be followed in order to ensure informed user consent to MIDlet actions:

- Any chargeable event generated by a MIDlet in the Third Party and Untrusted domains MUST be preceded by user notification in accordance with user permission settings, for example, showing the phone number the MIDlet is dialling, the URL being connected to, or the recipient of an SMS.
- Any chargeable event in progress (for example, peer-to-peer connection the user is charged for) MUST be indicated to the user.
- A MIDlet MUST get user approval to connect to the network, in accordance with user permission settings of the policy.
- Any MIDlet permissions must be presented to the user in an intuitive, user-friendly manner.
- A MIDlet MUST not be able to override security prompts and notifications to the user generated by the system or virtual machine.

- A MIDlet MUST not be able to simulate security warnings to mislead the user.
- A MIDlet MUST not be able to simulate key-press events to mislead the user.

8 MIDlet Download and Execution While Roaming and After Changing the Smart Card

All previously authorized and installed MIDlet suites MUST act in accordance with the device policy when the device is roaming, or when the device smart card is changed. Newly downloaded MIDlet suites are authenticated to a Protection Domain Root Certificate currently available either on the device (only for third party applications) or at the specified location in the SIM, USIM or WIM (for operator and third party applications) and are authorized in accordance with the device policy.

If device roaming or a smart card change causes failure to access network resources that the MIDlet was previously authorized to access, then the implementation MUST NOT throw a SecurityException. This failure is not related to MIDlet suite authorization, so the implementation MUST throw an IOException instead.

The permissions assigned to MIDlet suites installed in the Manufacturer, Trusted Third Party, and Untrusted domains are not affected by changes of the (U)ICC [(U)ICC], but MIDlet suites installed in the Operator domain MUST NOT execute if, after a smart card change, the SIM no longer holds the certificate containing the operator root public key that was used to authenticate the MIDlet suite to the Operator domain (see Section 3.2, “Operator Domain”).

Whether a MIDlet suite in the Operator domain can be executed depends on a comparison of “root key hash” values, computed as the 20-byte SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits) of a Protection Domain Root Certificate. The decision process SHOULD follow the following mechanism:

- When a MIDlet is installed in the Operator domain, it is signed by a certificate whose certification chain ends with the authenticating Protection Domain Root Certificate, stored in the smart card with the Operator-domain key-usage field. The 20-byte SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits) from that Protection Domain Root Certificate, termed the “authenticating root key hash” of the MIDlet, is stored in the device along with the MIDlet (as specified in Section 3.2).
- Whenever the smart card is changed, the 20-byte SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits) of each certificate stored in the new smart card with the Operator-domain key-usage field (Operator-domain root key hashes) is computed and stored before any MIDlet in the Operator domain is executed.
- A MIDlet in the Operator domain is disabled if its authenticating root key hash does not correspond to one of the new Operator-domain root key hashes generated after the smart card was changed.

Note: In this mechanism, there are two steps the device performs after the smart card has changed:

1. compute the new Operator domain root key hashes
2. for each MIDlet suite in the Operator domain, check whether its authenticating root key hash matchone of the new Operator domain root key hashes.

An implementation MAY perform these two steps at any time, provided NO Operator domain MIDlet suite is executed after a smartcard change if its authenticating root key hash does NOT correspond to one of the new Operator-domain root key hashes. Step 2 MAY be performed right after Step 1; alternatively, Steps 1 and 2 MAY be separated in time, in which case the implementation SHOULD store the results of Step 1 securely to be used in in Step 2 at a later time.

If the Operator Protection Domain Root Certificate is not present at the specified location, the user MUST be informed that the application cannot be executed without the authorizing Protection Domain Root Certificate. The device SHOULD also give the user the option to get information on the Protection Domain Root Certificate

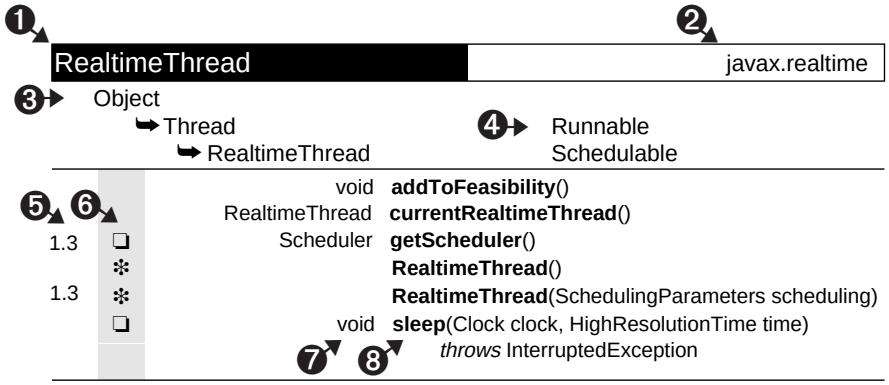
that was used to authenticate the application to the Operator domain. This information SHOULD include the *Subject* field of the root certificate.

Although it is mandatory only to check whether authenticating roots are still present in the smart card when the smart card is changed, an implementation MAY check on more occasions, and accordingly disable MIDlets suites in the Operator domain as specified above. If a MIDlet suite cannot be executed because the authenticating Operator Protection Domain Root Certificate is absent, the device MUST NOT delete the MIDlet suite. The device MAY inform the user in advance via an appropriate mechanism whether a MIDlet suite could execute or not, for example using a “disabled” look and feel in the display. However, the user MUST be able to delete these disabled MIDlets suites.

ALMANAC LEGEND

The almanac presents classes and intefaces in alphabetic order, regardless of their package. Fields, methods and constructors are in alphabetic order in a single list.

This almanac is modeled after the style introduced by Patrick Chan in his excellent book *Java Developers Almanac*.



1. Name of the class, interface, nested class or nested interface. Interfaces are italic.
2. Name of the package containing the class or interface.
3. Inheritance hierarchy. In this example, `RealtimeThread` extends `Thread`, which extends `Object`.
4. Implemented interfaces. The interface is to the right of, and on the same line as, the class that implements it. In this example, `Thread` implements `Runnable`, and `RealtimeThread` implements `Schedulable`.
5. The first column above is for the value of the `@since` comment, which indicates the version in which the item was introduced.
6. The second column above is for the following icons. If the "protected" symbol does not appear, the member is public. (Private and package-private modifiers also have no symbols.) One symbol from each group can appear in this column.

Modifiers	**Access Modifiers**	**Constructors and Fields**
❍ abstract	♦protected	❇ constructor
● final		✍ field
❑ static		
■ static final		

7. Return type of a method or declared type of a field. Blank for constructors.
8. Name of the constructor, field or method. Nested classes are listed in 1, not here.

Almanac

Alert	javax.microedition.lcdui

Object
➥Displayable
➥Screen
➥Alert

		void	**addCommand(Command cmd)**
	❄		**Alert(String title)**
	❄		**Alert(String title, String alertText, Image alertImage, AlertType alertType)**
midp 2.0	✍■	Command	**DISMISS_COMMAND**
	✍■	int	**FOREVER**
		int	**getDefaultTimeout()**
		Image	**getImage()**
midp 2.0		Gauge	**getIndicator()**
		String	**getString()**
		int	**getTimeout()**
		AlertType	**getType()**
		void	**removeCommand(Command cmd)**
		void	**setCommandListener(CommandListener l)**
		void	**setImage(Image img)**
midp 2.0		void	**setIndicator(Gauge indicator)**
		void	**setString(String str)**
		void	**setTimeout(int time)**
		void	**setType(AlertType type)**

AlertType	javax.microedition.lcdui

Object
➥AlertType

	✍■	AlertType	**ALARM**
	❄♦		**AlertType()**
	✍■	AlertType	**CONFIRMATION**
	✍■	AlertType	**ERROR**
	✍■	AlertType	**INFO**
		boolean	**playSound(Display display)**
	✍■	AlertType	**WARNING**

Canvas — javax.microedition.lcdui

Object
➥Displayable
➥Canvas

❇✦		**Canvas()**
✍■	int	**DOWN**
✍■	int	**FIRE**
✍■	int	**GAME_A**
✍■	int	**GAME_B**
✍■	int	**GAME_C**
✍■	int	**GAME_D**
	int	**getGameAction(int keyCode)**
	int	**getHeight()**
	int	**getKeyCode(int gameAction)**
	String	**getKeyName(int keyCode)**
	int	**getWidth()**
	boolean	**hasPointerEvents()**
	boolean	**hasPointerMotionEvents()**
	boolean	**hasRepeatEvents()**
✦	void	**hideNotify()**
	boolean	**isDoubleBuffered()**
✍■	int	**KEY_NUM0**
✍■	int	**KEY_NUM1**
✍■	int	**KEY_NUM2**
✍■	int	**KEY_NUM3**
✍■	int	**KEY_NUM4**
✍■	int	**KEY_NUM5**
✍■	int	**KEY_NUM6**
✍■	int	**KEY_NUM7**
✍■	int	**KEY_NUM8**
✍■	int	**KEY_NUM9**
✍■	int	**KEY_POUND**
✍■	int	**KEY_STAR**
✦	void	**keyPressed(int keyCode)**
✦	void	**keyReleased(int keyCode)**
✦	void	**keyRepeated(int keyCode)**
✍■	int	**LEFT**
❍✦	void	**paint(Graphics g)**
✦	void	**pointerDragged(int x, int y)**
✦	void	**pointerPressed(int x, int y)**
✦	void	**pointerReleased(int x, int y)**
●	void	**repaint()**
●	void	**repaint(int x, int y, int width, int height)**
✍■	int	**RIGHT**
●	void	**serviceRepaints()**

midp 2.0		void	**setFullScreenMode(boolean mode)**
	♦	void	**showNotify()**
midp 2.0	♦	void	**sizeChanged(int w, int h)**
	✍■	int	**UP**

Certificate — javax.microedition.pki

Certificate

	String	**getIssuer()**
	long	**getNotAfter()**
	long	**getNotBefore()**
	String	**getSerialNumber()**
	String	**getSigAlgName()**
	String	**getSubject()**
	String	**getType()**
	String	**getVersion()**

CertificateException — javax.microedition.pki

Object
- ➥Throwable
  - ➥Exception
    - ➥java.io.IOException
      - ➥CertificateException

✍■	byte	**BAD_EXTENSIONS**
✍■	byte	**BROKEN_CHAIN**
✍■	byte	**CERTIFICATE_CHAIN_TOO_LONG**
❄		**CertificateException(Certificate certificate, byte status)**
❄		**CertificateException(String message, Certificate certificate, byte status)**
✍■	byte	**EXPIRED**
	Certificate	**getCertificate()**
	byte	**getReason()**
✍■	byte	**INAPPROPRIATE_KEY_USAGE**
✍■	byte	**MISSING_SIGNATURE**
✍■	byte	**NOT_YET_VALID**
✍■	byte	**ROOT_CA_EXPIRED**
✍■	byte	**SITENAME_MISMATCH**
✍■	byte	**UNAUTHORIZED_INTERMEDIATE_CA**
✍■	byte	**UNRECOGNIZED_ISSUER**
✍■	byte	**UNSUPPORTED_PUBLIC_KEY_TYPE**
✍■	byte	**UNSUPPORTED_SIGALG**
✍■	byte	**VERIFICATION_FAILED**

Choice — javax.microedition.lcdui

Choice

		int	**append(String stringPart, Image imagePart)**
		void	**delete(int elementNum)**
midp 2.0		void	**deleteAll()**
	✍■	int	**EXCLUSIVE**
midp 2.0		int	**getFitPolicy()**
midp 2.0		Font	**getFont(int elementNum)**
		Image	**getImage(int elementNum)**
		int	**getSelectedFlags(boolean[] selectedArray_return)**
		int	**getSelectedIndex()**
		String	**getString(int elementNum)**
	✍■	int	**IMPLICIT**
		void	**insert(int elementNum, String stringPart, Image imagePart)**
		boolean	**isSelected(int elementNum)**
	✍■	int	**MULTIPLE**
midp 2.0	✍■	int	**POPUP**
		void	**set(int elementNum, String stringPart, Image imagePart)**
midp 2.0		void	**setFitPolicy(int fitPolicy)**
midp 2.0		void	**setFont(int elementNum, Font font)**
		void	**setSelectedFlags(boolean[] selectedArray)**
		void	**setSelectedIndex(int elementNum, boolean selected)**
		int	**size()**
midp 2.0	✍■	int	**TEXT_WRAP_DEFAULT**
midp 2.0	✍■	int	**TEXT_WRAP_OFF**
midp 2.0	✍■	int	**TEXT_WRAP_ON**

ChoiceGroup — javax.microedition.lcdui

Object
↳Item
↳ChoiceGroup — Choice

		int	**append(String stringPart, Image imagePart)**
	❄		**ChoiceGroup(String label, int choiceType)**
	❄		**ChoiceGroup(String label, int choiceType, String stringElements, Image imageElements)**
		void	**delete(int elementNum)**
		void	**deleteAll()**
midp 2.0		int	**getFitPolicy()**
midp 2.0		Font	**getFont(int elementNum)**
		Image	**getImage(int elementNum)**
		int	**getSelectedFlags(boolean[] selectedArray_return)**
		int	**getSelectedIndex()**
		String	**getString(int elementNum)**
		void	**insert(int elementNum, String stringPart, Image imagePart)**
		boolean	**isSelected(int elementNum)**

		void	**set(int elementNum, String stringPart, Image imagePart)**
midp 2.0		void	**setFitPolicy(int fitPolicy)**
midp 2.0		void	**setFont(int elementNum, Font font)**
		void	**setSelectedFlags(boolean[] selectedArray)**
		void	**setSelectedIndex(int elementNum, boolean selected)**
		int	**size()**

Command — javax.microedition.lcdui

Object
➥Command

	✍■	int	**BACK**
	✍■	int	**CANCEL**
	❇		**Command(String label, int commandType, int priority)**
midp 2.0	❇		**Command(String shortLabel, String longLabel, int commandType, int priority)**
	✍■	int	**EXIT**
		int	**getCommandType()**
		String	**getLabel()**
midp 2.0		String	**getLongLabel()**
		int	**getPriority()**
	✍■	int	**HELP**
	✍■	int	**ITEM**
	✍■	int	**OK**
	✍■	int	**SCREEN**
	✍■	int	**STOP**

CommandListener — javax.microedition.lcdui

CommandListener

		void	**commandAction(Command c, Displayable d)**

CommConnection — javax.microedition.io

CommConnection — StreamConnection

		int	**getBaudRate()**
		int	**setBaudRate(int baudrate)**

Connector — javax.microedition.io

Object
➥Connector

	❑	Connection	**open(String name) *throws* java.io.IOException**
	❑	Connection	**open(String name, int mode) *throws* java.io.IOException**
	❑	Connection	**open(String name, int mode, boolean timeouts) *throws* java.io.IOException**
	❑	java.io.DataInputStream	**openDataInputStream(String name) *throws* java.io.IOException**
	❑	java.io.DataOutputStream	**openDataOutputStream(String name) *throws* java.io.IOException**
	❑	java.io.InputStream	**openInputStream(String name) *throws* java.io.IOException**

❑	java.io.OutputStream	**openOutputStream(String name) *throws* java.io.IOException**
✍■	int	**READ**
✍■	int	**READ_WRITE**
✍■	int	**WRITE**

Control — javax.microedition.media

Control

Controllable — javax.microedition.media

Controllable

	Control	**getControl(String controlType)**
	Control[]	**getControls()**

CustomItem — javax.microedition.lcdui

Object
➥Item
➥CustomItem

❇♦		**CustomItem(String label)**
	int	**getGameAction(int keyCode)**
●♦	int	**getInteractionModes()**
❍♦	int	**getMinContentHeight()**
❍♦	int	**getMinContentWidth()**
❍♦	int	**getPrefContentHeight(int width)**
❍♦	int	**getPrefContentWidth(int height)**
♦	void	**hideNotify()**
●♦	void	**invalidate()**
✍■♦	int	**KEY_PRESS**
✍■♦	int	**KEY_RELEASE**
✍■♦	int	**KEY_REPEAT**
♦	void	**keyPressed(int keyCode)**
♦	void	**keyReleased(int keyCode)**
♦	void	**keyRepeated(int keyCode)**
✍■♦	int	**NONE**
❍♦	void	**paint(Graphics g, int w, int h)**
✍■♦	int	**POINTER_DRAG**
✍■♦	int	**POINTER_PRESS**
✍■♦	int	**POINTER_RELEASE**
♦	void	**pointerDragged(int x, int y)**
♦	void	**pointerPressed(int x, int y)**
♦	void	**pointerReleased(int x, int y)**
●♦	void	**repaint()**
●♦	void	**repaint(int x, int y, int w, int h)**
♦	void	**showNotify()**
♦	void	**sizeChanged(int w, int h)**
✍■♦	int	**TRAVERSE_HORIZONTAL**

	✍■♦	int	**TRAVERSE_VERTICAL**
	♦	boolean	**traverse(int dir, int viewportWidth, int viewportHeight, int[] visRect_inout)**
	♦	void	**traverseOut()**

DateField — javax.microedition.lcdui

Object
➥Item
➥DateField

	✍■	int	**DATE**
	✍■	int	**DATE_TIME**
	❇		**DateField(String label, int mode)**
	❇		**DateField(String label, int mode, java.util.TimeZone timeZone)**
		java.util.Date	**getDate()**
		int	**getInputMode()**
		void	**setDate(java.util.Date date)**
		void	**setInputMode(int mode)**
	✍■	int	**TIME**

Display — javax.microedition.lcdui

Object
➥Display

midp 2.0	✍■	int	**ALERT**
		void	**callSerially(Runnable r)**
midp 2.0	✍■	int	**CHOICE_GROUP_ELEMENT**
midp 2.0	✍■	int	**COLOR_BACKGROUND**
midp 2.0	✍■	int	**COLOR_BORDER**
midp 2.0	✍■	int	**COLOR_FOREGROUND**
midp 2.0	✍■	int	**COLOR_HIGHLIGHTED_BACKGROUND**
midp 2.0	✍■	int	**COLOR_HIGHLIGHTED_BORDER**
midp 2.0	✍■	int	**COLOR_HIGHLIGHTED_FOREGROUND**
midp 2.0		boolean	**flashBacklight(int duration)**
midp 2.0		int	**getBestImageHeight(int imageType)**
midp 2.0		int	**getBestImageWidth(int imageType)**
midp 2.0		int	**getBorderStyle(boolean highlighted)**
midp 2.0		int	**getColor(int colorSpecifier)**
		Displayable	**getCurrent()**
	❑	Display	**getDisplay(javax.microedition.midlet.MIDlet m)**
		boolean	**isColor()**
midp 2.0	✍■	int	**LIST_ELEMENT**
midp 2.0		int	**numAlphaLevels()**
		int	**numColors()**
		void	**setCurrent(Alert alert, Displayable nextDisplayable)**
		void	**setCurrent(Displayable nextDisplayable)**
midp 2.0		void	**setCurrentItem(Item item)**
midp 2.0		boolean	**vibrate(int duration)**

Displayable — javax.microedition.lcdui

Object
➥Displayable

		void	**addCommand(Command cmd)**
midp 2.0		int	**getHeight()**
midp 2.0		Ticker	**getTicker()**
midp 2.0		String	**getTitle()**
midp 2.0		int	**getWidth()**
		boolean	**isShown()**
		void	**removeCommand(Command cmd)**
		void	**setCommandListener(CommandListener l)**
midp 2.0		void	**setTicker(Ticker ticker)**
midp 2.0		void	**setTitle(String s)**
midp 2.0	♦	void	**sizeChanged(int w, int h)**

Font — javax.microedition.lcdui

Object
➥Font

		int	**charsWidth(char[] ch, int offset, int length)**
		int	**charWidth(char ch)**
	✍■	int	**FACE_MONOSPACE**
	✍■	int	**FACE_PROPORTIONAL**
	✍■	int	**FACE_SYSTEM**
midp 2.0	✍■	int	**FONT_INPUT_TEXT**
midp 2.0	✍■	int	**FONT_STATIC_TEXT**
		int	**getBaselinePosition()**
	❑	Font	**getDefaultFont()**
		int	**getFace()**
midp 2.0	❑	Font	**getFont(int fontSpecifier)**
	❑	Font	**getFont(int face, int style, int size)**
		int	**getHeight()**
		int	**getSize()**
		int	**getStyle()**
		boolean	**isBold()**
		boolean	**isItalic()**
		boolean	**isPlain()**
		boolean	**isUnderlined()**
	✍■	int	**SIZE_LARGE**
	✍■	int	**SIZE_MEDIUM**
	✍■	int	**SIZE_SMALL**
		int	**stringWidth(String str)**
	✍■	int	**STYLE_BOLD**
	✍■	int	**STYLE_ITALIC**

	✍■	int	**STYLE_PLAIN**
	✍■	int	**STYLE_UNDERLINED**
		int	**substringWidth(String str, int offset, int len)**

Form — javax.microedition.lcdui

Object
➥Displayable
➥Screen
➥Form

		int	**append(Image img)**
		int	**append(Item item)**
		int	**append(String str)**
		void	**delete(int itemNum)**
midp 2.0		void	**deleteAll()**
	❇		**Form(String title)**
	❇		**Form(String title, Item items)**
		Item	**get(int itemNum)**
midp 2.0		int	**getHeight()**
midp 2.0		int	**getWidth()**
		void	**insert(int itemNum, Item item)**
		void	**set(int itemNum, Item item)**
		void	**setItemStateListener(ItemStateListener iListener)**
		int	**size()**

GameCanvas — javax.microedition.lcdui.game

Object
➥javax.microedition.lcdui.Displayable
➥javax.microedition.lcdui.Canvas
➥GameCanvas

	✍■	int	**DOWN_PRESSED**
	✍■	int	**FIRE_PRESSED**
		void	**flushGraphics()**
		void	**flushGraphics(int x, int y, int width, int height)**
	✍■	int	**GAME_A_PRESSED**
	✍■	int	**GAME_B_PRESSED**
	✍■	int	**GAME_C_PRESSED**
	✍■	int	**GAME_D_PRESSED**
	❇✦		**GameCanvas(boolean suppressKeyEvents)**
	✦	javax.microedition.lcdui.Graphics	**getGraphics()**
		int	**getKeyStates()**
	✍■	int	**LEFT_PRESSED**
		void	**paint(javax.microedition.lcdui.Graphics g)**
	✍■	int	**RIGHT_PRESSED**
	✍■	int	**UP_PRESSED**

Gauge — javax.microedition.lcdui

Object
➥Item
➥Gauge

midp 2.0	✍■	int	**CONTINUOUS_IDLE**
midp 2.0	✍■	int	**CONTINUOUS_RUNNING**
	❇		**Gauge(String label, boolean interactive, int maxValue, int initialValue)**
		int	**getMaxValue()**
		int	**getValue()**
midp 2.0	✍■	int	**INCREMENTAL_IDLE**
midp 2.0	✍■	int	**INCREMENTAL_UPDATING**
midp 2.0	✍■	int	**INDEFINITE**
		boolean	**isInteractive()**
		void	**setMaxValue(int maxValue)**
		void	**setValue(int value)**

Graphics — javax.microedition.lcdui

Object
➥Graphics

	✍■	int	**BASELINE**
	✍■	int	**BOTTOM**
		void	**clipRect(int x, int y, int width, int height)**
midp 2.0		void	**copyArea(int x_src, int y_src, int width, int height, int x_dest, int y_dest, int anchor)**
	✍■	int	**DOTTED**
		void	**drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)**
		void	**drawChar(char character, int x, int y, int anchor)**
		void	**drawChars(char[] data, int offset, int length, int x, int y, int anchor)**
		void	**drawImage(Image img, int x, int y, int anchor)**
		void	**drawLine(int x1, int y1, int x2, int y2)**
		void	**drawRect(int x, int y, int width, int height)**
midp 2.0		void	**drawRegion(Image src, int x_src, int y_src, int width, int height, int transform, int x_dest, int y_dest, int anchor)**
midp 2.0		void	**drawRGB(int[] rgbData, int offset, int scanlength, int x, int y, int width, int height, boolean processAlpha)**
		void	**drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)**
		void	**drawString(String str, int x, int y, int anchor)**
		void	**drawSubstring(String str, int offset, int len, int x, int y, int anchor)**
		void	**fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)**
		void	**fillRect(int x, int y, int width, int height)**
		void	**fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)**
midp 2.0		void	**fillTriangle(int x1, int y1, int x2, int y2, int x3, int y3)**
		int	**getBlueComponent()**
		int	**getClipHeight()**

		int **getClipWidth()**
		int **getClipX()**
		int **getClipY()**
		int **getColor()**
midp 2.0		int **getDisplayColor(int color)**
		Font **getFont()**
		int **getGrayScale()**
		int **getGreenComponent()**
		int **getRedComponent()**
		int **getStrokeStyle()**
		int **getTranslateX()**
		int **getTranslateY()**
	✍■	int **HCENTER**
	✍■	int **LEFT**
	✍■	int **RIGHT**
		void **setClip(int x, int y, int width, int height)**
		void **setColor(int RGB)**
		void **setColor(int red, int green, int blue)**
		void **setFont(Font font)**
		void **setGrayScale(int value)**
		void **setStrokeStyle(int style)**
	✍■	int **SOLID**
	✍■	int **TOP**
		void **translate(int x, int y)**
	✍■	int **VCENTER**

HttpConnection — javax.microedition.io

HttpConnection — ContentConnection

	✍■	String **GET**
		long **getDate() *throws* java.io.IOException**
		long **getExpiration() *throws* java.io.IOException**
		String **getFile()**
		String **getHeaderField(int n) *throws* java.io.IOException**
		String **getHeaderField(String name) *throws* java.io.IOException**
		long **getHeaderFieldDate(String name, long def) *throws* java.io.IOException**
		int **getHeaderFieldInt(String name, int def) *throws* java.io.IOException**
		String **getHeaderFieldKey(int n) *throws* java.io.IOException**
		String **getHost()**
		long **getLastModified() *throws* java.io.IOException**
		int **getPort()**
		String **getProtocol()**
		String **getQuery()**
		String **getRef()**
		String **getRequestMethod()**
		String **getRequestProperty(String key)**

	int	**getResponseCode() *throws* java.io.IOException**
	String	**getResponseMessage() *throws* java.io.IOException**
	String	**getURL()**
✍■	String	**HEAD**
✍■	int	**HTTP_ACCEPTED**
✍■	int	**HTTP_BAD_GATEWAY**
✍■	int	**HTTP_BAD_METHOD**
✍■	int	**HTTP_BAD_REQUEST**
✍■	int	**HTTP_CLIENT_TIMEOUT**
✍■	int	**HTTP_CONFLICT**
✍■	int	**HTTP_CREATED**
✍■	int	**HTTP_ENTITY_TOO_LARGE**
✍■	int	**HTTP_EXPECT_FAILED**
✍■	int	**HTTP_FORBIDDEN**
✍■	int	**HTTP_GATEWAY_TIMEOUT**
✍■	int	**HTTP_GONE**
✍■	int	**HTTP_INTERNAL_ERROR**
✍■	int	**HTTP_LENGTH_REQUIRED**
✍■	int	**HTTP_MOVED_PERM**
✍■	int	**HTTP_MOVED_TEMP**
✍■	int	**HTTP_MULT_CHOICE**
✍■	int	**HTTP_NO_CONTENT**
✍■	int	**HTTP_NOT_ACCEPTABLE**
✍■	int	**HTTP_NOT_AUTHORITATIVE**
✍■	int	**HTTP_NOT_FOUND**
✍■	int	**HTTP_NOT_IMPLEMENTED**
✍■	int	**HTTP_NOT_MODIFIED**
✍■	int	**HTTP_OK**
✍■	int	**HTTP_PARTIAL**
✍■	int	**HTTP_PAYMENT_REQUIRED**
✍■	int	**HTTP_PRECON_FAILED**
✍■	int	**HTTP_PROXY_AUTH**
✍■	int	**HTTP_REQ_TOO_LONG**
✍■	int	**HTTP_RESET**
✍■	int	**HTTP_SEE_OTHER**
✍■	int	**HTTP_TEMP_REDIRECT**
✍■	int	**HTTP_UNAUTHORIZED**
✍■	int	**HTTP_UNAVAILABLE**
✍■	int	**HTTP_UNSUPPORTED_RANGE**
✍■	int	**HTTP_UNSUPPORTED_TYPE**
✍■	int	**HTTP_USE_PROXY**
✍■	int	**HTTP_VERSION**
✍■	String	**POST**
	void	**setRequestMethod(String method) *throws* java.io.IOException**
	void	**setRequestProperty(String key, String value) *throws* java.io.IOException**

HttpsConnection — javax.microedition.io

HttpsConnection
HttpConnection

		int	**getPort()**
		SecurityInfo	**getSecurityInfo() *throws* java.io.IOException**

IllegalStateException — java.lang

Object
➥Throwable
➥Exception
➥RuntimeException
➥IllegalStateException

	❇		**IllegalStateException()**
	❇		**IllegalStateException(String s)**

Image — javax.microedition.lcdui

Object
➥Image

	❑	Image	**createImage(byte[] imageData, int imageOffset, int imageLength)**
	❑	Image	**createImage(Image source)**
midp 2.0	❑	Image	**createImage(Image image, int x, int y, int width, int height, int transform)**
midp 2.0	❑	Image	**createImage(java.io.InputStream stream) *throws* java.io.IOException**
	❑	Image	**createImage(int width, int height)**
	❑	Image	**createImage(String name) *throws* java.io.IOException**
midp 2.0	❑	Image	**createRGBImage(int[] rgb, int width, int height, boolean processAlpha)**
		Graphics	**getGraphics()**
		int	**getHeight()**
midp 2.0		void	**getRGB(int[] rgbData, int offset, int scanlength, int x, int y, int width, int height)**
		int	**getWidth()**
		boolean	**isMutable()**

ImageItem — javax.microedition.lcdui

Object
➥Item
➥ImageItem

		String	**getAltText()**
midp 2.0		int	**getAppearanceMode()**
		Image	**getImage()**
		int	**getLayout()**
	❇		**ImageItem(String label, Image img, int layout, String altText)**
midp 2.0	❇		**ImageItem(String label, Image image, int layout, String altText, int appearanceMode)**
	✍■	int	**LAYOUT_CENTER**
	✍■	int	**LAYOUT_DEFAULT**

	✍■	int	**LAYOUT_LEFT**
	✍■	int	**LAYOUT_NEWLINE_AFTER**
	✍■	int	**LAYOUT_NEWLINE_BEFORE**
	✍■	int	**LAYOUT_RIGHT**
		void	**setAltText(String text)**
		void	**setImage(Image img)**
		void	**setLayout(int layout)**

InvalidRecordIDException — javax.microedition.rms

Object
➥Throwable
➥Exception
➥RecordStoreException
➥InvalidRecordIDException

	❊		**InvalidRecordIDException()**
	❊		**InvalidRecordIDException(String message)**

Item — javax.microedition.lcdui

Object
➥Item

midp 2.0		void	**addCommand(Command cmd)**
midp 2.0	✍■	int	**BUTTON**
		String	**getLabel()**
midp 2.0		int	**getLayout()**
midp 2.0		int	**getMinimumHeight()**
midp 2.0		int	**getMinimumWidth()**
midp 2.0		int	**getPreferredHeight()**
midp 2.0		int	**getPreferredWidth()**
midp 2.0	✍■	int	**HYPERLINK**
midp 2.0	✍■	int	**LAYOUT_2**
midp 2.0	✍■	int	**LAYOUT_BOTTOM**
midp 2.0	✍■	int	**LAYOUT_CENTER**
midp 2.0	✍■	int	**LAYOUT_DEFAULT**
midp 2.0	✍■	int	**LAYOUT_EXPAND**
midp 2.0	✍■	int	**LAYOUT_LEFT**
midp 2.0	✍■	int	**LAYOUT_NEWLINE_AFTER**
midp 2.0	✍■	int	**LAYOUT_NEWLINE_BEFORE**
midp 2.0	✍■	int	**LAYOUT_RIGHT**
midp 2.0	✍■	int	**LAYOUT_SHRINK**
midp 2.0	✍■	int	**LAYOUT_TOP**
midp 2.0	✍■	int	**LAYOUT_VCENTER**
midp 2.0	✍■	int	**LAYOUT_VEXPAND**
midp 2.0	✍■	int	**LAYOUT_VSHRINK**
midp 2.0		void	**notifyStateChanged()**
midp 2.0	✍■	int	**PLAIN**
midp 2.0		void	**removeCommand(Command cmd)**

midp 2.0		void	**setDefaultCommand(Command cmd)**
midp 2.0		void	**setItemCommandListener(ItemCommandListener l)**
		void	**setLabel(String label)**
midp 2.0		void	**setLayout(int layout)**
midp 2.0		void	**setPreferredSize(int width, int height)**

ItemCommandListener — javax.microedition.lcdui

ItemCommandListener

		void	**commandAction(Command c, Item item)**

ItemStateListener — javax.microedition.lcdui

ItemStateListener

		void	**itemStateChanged(Item item)**

Layer — javax.microedition.lcdui.game

Object
➥Layer

	●	int	**getHeight()**
	●	int	**getWidth()**
	●	int	**getX()**
	●	int	**getY()**
	●	boolean	**isVisible()**
		void	**move(int dx, int dy)**
	❍	void	**paint(javax.microedition.lcdui.Graphics g)**
		void	**setPosition(int x, int y)**
		void	**setVisible(boolean visible)**

LayerManager — javax.microedition.lcdui.game

Object
➥LayerManager

		void	**append(Layer l)**
		Layer	**getLayerAt(int index)**
		int	**getSize()**
		void	**insert(Layer l, int index)**
	❉		**LayerManager()**
		void	**paint(javax.microedition.lcdui.Graphics g, int x, int y)**
		void	**remove(Layer l)**
		void	**setViewWindow(int x, int y, int width, int height)**

List — javax.microedition.lcdui

Object
➥Displayable
➥Screen
➥List
Choice

		int	**append(String stringPart, Image imagePart)**
		void	**delete(int elementNum)**
		void	**deleteAll()**
midp 2.0		int	**getFitPolicy()**
midp 2.0		Font	**getFont(int elementNum)**
		Image	**getImage(int elementNum)**
		int	**getSelectedFlags(boolean[] selectedArray_return)**
		int	**getSelectedIndex()**
		String	**getString(int elementNum)**
		void	**insert(int elementNum, String stringPart, Image imagePart)**
		boolean	**isSelected(int elementNum)**
	❇		**List(String title, int listType)**
	❇		**List(String title, int listType, String stringElements, Image imageElements)**
midp 2.0		void	**removeCommand(Command cmd)**
	✍■	Command	**SELECT_COMMAND**
		void	**set(int elementNum, String stringPart, Image imagePart)**
midp 2.0		void	**setFitPolicy(int fitPolicy)**
midp 2.0		void	**setFont(int elementNum, Font font)**
midp 2.0		void	**setSelectCommand(Command command)**
		void	**setSelectedFlags(boolean[] selectedArray)**
		void	**setSelectedIndex(int elementNum, boolean selected)**
		int	**size()**

Manager — javax.microedition.media

Object
➥Manager

	❑	Player	**createPlayer(java.io.InputStream stream, String type) *throws* java.io.IOException, MediaException**
	❑	Player	**createPlayer(String locator) *throws* java.io.IOException, MediaException**
	❑	String[]	**getSupportedContentTypes(String protocol)**
	❑	String[]	**getSupportedProtocols(String content_type)**
	❑	void	**playTone(int note, int duration, int volume) *throws* MediaException**
	✍■	String	**TONE_DEVICE_LOCATOR**

MediaException — javax.microedition.media

Object
➥Throwable
➥Exception
➥MediaException

	❊	**MediaException()**
	❊	**MediaException(String reason)**

MIDlet — javax.microedition.midlet

Object
➥MIDlet

midp 2.0	●	int	**checkPermission(String permission)**
	❍♦	void	**destroyApp(boolean unconditional)** *throws* **MIDletStateChangeException**
	●	String	**getAppProperty(String key)**
	❊♦		**MIDlet()**
	●	void	**notifyDestroyed()**
	●	void	**notifyPaused()**
	❍♦	void	**pauseApp()**
midp 2.0	●	boolean	**platformRequest(String URL)** *throws* **javax.microedition.io.ConnectionNotFoundException**
	●	void	**resumeRequest()**
	❍♦	void	**startApp()** *throws* **MIDletStateChangeException**

MIDletStateChangeException — javax.microedition.midlet

Object
➥Throwable
➥Exception
➥MIDletStateChangeException

	❊	**MIDletStateChangeException()**
	❊	**MIDletStateChangeException(String s)**

Player — javax.microedition.media

Player — Controllable

		void	**addPlayerListener(PlayerListener playerListener)**
		void	**close()**
	✍■	int	**CLOSED**
		void	**deallocate()**
		String	**getContentType()**
		long	**getDuration()**
		long	**getMediaTime()**
		int	**getState()**
		void	**prefetch()** *throws* **MediaException**
	✍■	int	**PREFETCHED**
		void	**realize()** *throws* **MediaException**

✍■	int	**REALIZED**
	void	**removePlayerListener(PlayerListener playerListener)**
	void	**setLoopCount(int count)**
	long	**setMediaTime(long now)** ***throws*** **MediaException**
	void	**start()** ***throws*** **MediaException**
✍■	int	**STARTED**
	void	**stop()** ***throws*** **MediaException**
✍■	long	**TIME_UNKNOWN**
✍■	int	**UNREALIZED**

PlayerListener — javax.microedition.media

PlayerListener

✍■	String	**CLOSED**
✍■	String	**DEVICE_AVAILABLE**
✍■	String	**DEVICE_UNAVAILABLE**
✍■	String	**DURATION_UPDATED**
✍■	String	**END_OF_MEDIA**
✍■	String	**ERROR**
	void	**playerUpdate(Player player, String event, Object eventData)**
✍■	String	**STARTED**
✍■	String	**STOPPED**
✍■	String	**VOLUME_CHANGED**

PushRegistry — javax.microedition.io

Object
➥PushRegistry

❑	String	**getFilter(String connection)**
❑	String	**getMIDlet(String connection)**
❑	String[]	**listConnections(boolean available)**
❑	long	**registerAlarm(String midlet, long time)** ***throws*** **ClassNotFoundException, ConnectionNotFoundException**
❑	void	**registerConnection(String connection, String midlet, String filter)** ***throws*** **ClassNotFoundException, java.io.IOException**
❑	boolean	**unregisterConnection(String connection)**

RecordComparator — javax.microedition.rms

RecordComparator

	int	**compare(byte[] rec1, byte[] rec2)**
✍■	int	**EQUIVALENT**
✍■	int	**FOLLOWS**
✍■	int	**PRECEDES**

RecordEnumeration — javax.microedition.rms

RecordEnumeration

void	**destroy()**
boolean	**hasNextElement()**
boolean	**hasPreviousElement()**
boolean	**isKeptUpdated()**
void	**keepUpdated(boolean keepUpdated)**
byte[]	**nextRecord() *throws* InvalidRecordIDException, RecordStoreNotOpenException, RecordStoreException**
int	**nextRecordId() *throws* InvalidRecordIDException**
int	**numRecords()**
byte[]	**previousRecord() *throws* InvalidRecordIDException, RecordStoreNotOpenException, RecordStoreException**
int	**previousRecordId() *throws* InvalidRecordIDException**
void	**rebuild()**
void	**reset()**

RecordFilter — javax.microedition.rms

RecordFilter

boolean	**matches(byte[] candidate)**

RecordListener — javax.microedition.rms

RecordListener

void	**recordAdded(RecordStore recordStore, int recordId)**
void	**recordChanged(RecordStore recordStore, int recordId)**
void	**recordDeleted(RecordStore recordStore, int recordId)**

RecordStore — javax.microedition.rms

Object
➥RecordStore

int	**addRecord(byte[] data, int offset, int numBytes) *throws* RecordStoreNotOpenException, RecordStoreException, RecordStoreFullException**
void	**addRecordListener(RecordListener listener)**
✍■ int	**AUTHMODE_ANY**
✍■ int	**AUTHMODE_PRIVATE**
void	**closeRecordStore() *throws* RecordStoreNotOpenException, RecordStoreException**
void	**deleteRecord(int recordId) *throws* RecordStoreNotOpenException, InvalidRecordIDException, RecordStoreException**
❑ void	**deleteRecordStore(String recordStoreName) *throws* RecordStoreException, RecordStoreNotFoundException**
RecordEnumeration	**enumerateRecords(RecordFilter filter, RecordComparator comparator, boolean keepUpdated) *throws* RecordStoreNotOpenException**
long	**getLastModified() *throws* RecordStoreNotOpenException**

		String	**getName() *throws* RecordStoreNotOpenException**
		int	**getNextRecordID() *throws* RecordStoreNotOpenException, RecordStoreException**
		int	**getNumRecords() *throws* RecordStoreNotOpenException**
		byte[]	**getRecord(int recordId) *throws* RecordStoreNotOpenException, InvalidRecordIDException, RecordStoreException**
		int	**getRecord(int recordId, byte[] buffer, int offset) *throws* RecordStoreNotOpenException, InvalidRecordIDException, RecordStoreException**
		int	**getRecordSize(int recordId) *throws* RecordStoreNotOpenException, InvalidRecordIDException, RecordStoreException**
		int	**getSize() *throws* RecordStoreNotOpenException**
		int	**getSizeAvailable() *throws* RecordStoreNotOpenException**
		int	**getVersion() *throws* RecordStoreNotOpenException**
	❑	String[]	**listRecordStores()**
	❑	RecordStore	**openRecordStore(String recordStoreName, boolean createIfNecessary) *throws* RecordStoreException, RecordStoreFullException, RecordStoreNotFoundException**
midp 2.0	❑	RecordStore	**openRecordStore(String recordStoreName, boolean createIfNecessary, int authmode, boolean writable) *throws* RecordStoreException, RecordStoreFullException, RecordStoreNotFoundException**
midp 2.0	❑	RecordStore	**openRecordStore(String recordStoreName, String vendorName, String suiteName) *throws* RecordStoreException, RecordStoreNotFoundException**
		void	**removeRecordListener(RecordListener listener)**
midp 2.0		void	**setMode(int authmode, boolean writable) *throws* RecordStoreException**
		void	**setRecord(int recordId, byte[] newData, int offset, int numBytes) *throws* RecordStoreNotOpenException, InvalidRecordIDException, RecordStoreException, RecordStoreFullException**

RecordStoreException — javax.microedition.rms

Object
➥Throwable
➥Exception
➥RecordStoreException

	❊	**RecordStoreException()**
	❊	**RecordStoreException(String message)**

RecordStoreFullException — javax.microedition.rms

Object
➥Throwable
➥Exception
➥RecordStoreException
➥RecordStoreFullException

	❊	**RecordStoreFullException()**
	❊	**RecordStoreFullException(String message)**

RecordStoreNotFoundException — javax.microedition.rms

Object
➥Throwable
➥Exception
➥RecordStoreException
➥RecordStoreNotFoundException

❊	**RecordStoreNotFoundException()**
❊	**RecordStoreNotFoundException(String message)**

RecordStoreNotOpenException — javax.microedition.rms

Object
➥Throwable
➥Exception
➥RecordStoreException
➥RecordStoreNotOpenException

❊	**RecordStoreNotOpenException()**
❊	**RecordStoreNotOpenException(String message)**

Screen — javax.microedition.lcdui

Object
➥Displayable
➥Screen

SecureConnection — javax.microedition.io

SecureConnection — SocketConnection

SecurityInfo	**getSecurityInfo() *throws* java.io.IOException**

SecurityInfo — javax.microedition.io

SecurityInfo

String	**getCipherSuite()**
String	**getProtocolName()**
String	**getProtocolVersion()**
javax.microedition.pki.Certificate	**getServerCertificate()**

ServerSocketConnection — javax.microedition.io

ServerSocketConnection — StreamConnectionNotifier

String	**getLocalAddress() *throws* java.io.IOException**
int	**getLocalPort() *throws* java.io.IOException**

SocketConnection — javax.microedition.io

SocketConnection
➥StreamConnection

✍■	byte	**DELAY**
	String	**getAddress()** ***throws*** **java.io.IOException**
	String	**getLocalAddress()** ***throws*** **java.io.IOException**
	int	**getLocalPort()** ***throws*** **java.io.IOException**
	int	**getPort()** ***throws*** **java.io.IOException**
	int	**getSocketOption(byte option)** ***throws*** **IllegalArgumentException, java.io.IOException**
✍■	byte	**KEEPALIVE**
✍■	byte	**LINGER**
✍■	byte	**RCVBUF**
	void	**setSocketOption(byte option, int value)** ***throws*** **IllegalArgumentException, java.io.IOException**
✍■	byte	**SNDBUF**

Spacer — javax.microedition.lcdui

Object
➥Item
➥Spacer

	void	**addCommand(Command cmd)**
	void	**setDefaultCommand(Command cmd)**
	void	**setLabel(String label)**
	void	**setMinimumSize(int minWidth, int minHeight)**
❄		**Spacer(int minWidth, int minHeight)**

Sprite — javax.microedition.lcdui.game

Object
➥Layer
➥Sprite

●	boolean	**collidesWith(javax.microedition.lcdui.Image image, int x, int y, boolean pixelLevel)**
●	boolean	**collidesWith(Sprite s, boolean pixelLevel)**
●	boolean	**collidesWith(TiledLayer t, boolean pixelLevel)**
	void	**defineCollisionRectangle(int x, int y, int width, int height)**
	void	**defineReferencePixel(int x, int y)**
●	int	**getFrame()**
	int	**getFrameSequenceLength()**
	int	**getRawFrameCount()**
	int	**getRefPixelX()**
	int	**getRefPixelY()**
	void	**nextFrame()**
●	void	**paint(javax.microedition.lcdui.Graphics g)**
	void	**prevFrame()**
	void	**setFrame(int sequenceIndex)**
	void	**setFrameSequence(int[] sequence)**

		void	**setImage(javax.microedition.lcdui.Image img, int frameWidth, int frameHeight)**
		void	**setRefPixelPosition(int x, int y)**
		void	**setTransform(int transform)**
	❊		**Sprite(javax.microedition.lcdui.Image image)**
	❊		**Sprite(javax.microedition.lcdui.Image image, int frameWidth, int frameHeight)**
	❊		**Sprite(Sprite s)**
	✍■	int	**TRANS_MIRROR**
	✍■	int	**TRANS_MIRROR_ROT180**
	✍■	int	**TRANS_MIRROR_ROT270**
	✍■	int	**TRANS_MIRROR_ROT90**
	✍■	int	**TRANS_NONE**
	✍■	int	**TRANS_ROT180**
	✍■	int	**TRANS_ROT270**
	✍■	int	**TRANS_ROT90**

StringItem — javax.microedition.lcdui

Object
➥Item
➥StringItem

midp 2.0		int	**getAppearanceMode()**
midp 2.0		Font	**getFont()**
		String	**getText()**
midp 2.0		void	**setFont(Font font)**
		void	**setText(String text)**
	❊		**StringItem(String label, String text)**
midp 2.0	❊		**StringItem(String label, String text, int appearanceMode)**

TextBox — javax.microedition.lcdui

Object
➥Displayable
➥Screen
➥TextBox

		void	**delete(int offset, int length)**
		int	**getCaretPosition()**
		int	**getChars(char[] data)**
		int	**getConstraints()**
		int	**getMaxSize()**
		String	**getString()**
		void	**insert(char[] data, int offset, int length, int position)**
		void	**insert(String src, int position)**
		void	**setChars(char[] data, int offset, int length)**
		void	**setConstraints(int constraints)**
midp 2.0		void	**setInitialInputMode(String characterSubset)**
		int	**setMaxSize(int maxSize)**

		void	**setString(String text)**
		int	**size()**
	❊		**TextBox(String title, String text, int maxSize, int constraints)**

TextField — javax.microedition.lcdui

Object
➥Item
➥TextField

	✍■	int	**ANY**
	✍■	int	**CONSTRAINT_MASK**
midp 2.0	✍■	int	**DECIMAL**
		void	**delete(int offset, int length)**
	✍■	int	**EMAILADDR**
		int	**getCaretPosition()**
		int	**getChars(char[] data)**
		int	**getConstraints()**
		int	**getMaxSize()**
		String	**getString()**
midp 2.0	✍■	int	**INITIAL_CAPS_SENTENCE**
midp 2.0	✍■	int	**INITIAL_CAPS_WORD**
		void	**insert(char[] data, int offset, int length, int position)**
		void	**insert(String src, int position)**
midp 2.0	✍■	int	**NON_PREDICTIVE**
	✍■	int	**NUMERIC**
	✍■	int	**PASSWORD**
	✍■	int	**PHONENUMBER**
midp 2.0	✍■	int	**SENSITIVE**
		void	**setChars(char[] data, int offset, int length)**
		void	**setConstraints(int constraints)**
midp 2.0		void	**setInitialInputMode(String characterSubset)**
		int	**setMaxSize(int maxSize)**
		void	**setString(String text)**
		int	**size()**
	❊		**TextField(String label, String text, int maxSize, int constraints)**
midp 2.0	✍■	int	**UNEDITABLE**
	✍■	int	**URL**

Ticker — javax.microedition.lcdui

Object
➥Ticker

		String	**getString()**
		void	**setString(String str)**
	❊		**Ticker(String str)**

TiledLayer — javax.microedition.lcdui.game

Object
➥Layer
➥TiledLayer

	int	**createAnimatedTile(int staticTileIndex)**
	void	**fillCells(int col, int row, int numCols, int numRows, int tileIndex)**
	int	**getAnimatedTile(int animatedTileIndex)**
	int	**getCell(int col, int row)**
●	int	**getCellHeight()**
●	int	**getCellWidth()**
●	int	**getColumns()**
●	int	**getRows()**
●	void	**paint(javax.microedition.lcdui.Graphics g)**
	void	**setAnimatedTile(int animatedTileIndex, int staticTileIndex)**
	void	**setCell(int col, int row, int tileIndex)**
	void	**setStaticTileSet(javax.microedition.lcdui.Image image, int tileWidth, int tileHeight)**
❊		**TiledLayer(int columns, int rows, javax.microedition.lcdui.Image image, int tileWidth, int tileHeight)**

Timer — java.util

Object
➥Timer

	void	**cancel()**
	void	**schedule(TimerTask task, Date time)**
	void	**schedule(TimerTask task, Date firstTime, long period)**
	void	**schedule(TimerTask task, long delay)**
	void	**schedule(TimerTask task, long delay, long period)**
	void	**scheduleAtFixedRate(TimerTask task, Date firstTime, long period)**
	void	**scheduleAtFixedRate(TimerTask task, long delay, long period)**
❊		**Timer()**

TimerTask — java.util

Object
➥TimerTask — Runnable

	boolean	**cancel()**
❍	void	**run()**
	long	**scheduledExecutionTime()**
❊✦		**TimerTask()**

ToneControl	**javax.microedition.media.control**
ToneControl	javax.microedition.media.Control
byte	**BLOCK_END**
byte	**BLOCK_START**
byte	**C4**
byte	**PLAY_BLOCK**
byte	**REPEAT**
byte	**RESOLUTION**
byte	**SET_VOLUME**
void	**setSequence(byte[] sequence)**
byte	**SILENCE**
byte	**TEMPO**
byte	**VERSION**

UDPDatagramConnection	**javax.microedition.io**
UDPDatagramConnection	DatagramConnection
String	**getLocalAddress() *throws* java.io.IOException**
int	**getLocalPort() *throws* java.io.IOException**

VolumeControl	**javax.microedition.media.control**
VolumeControl	javax.microedition.media.Control
int	**getLevel()**
boolean	**isMuted()**
int	**setLevel(int level)**
void	**setMute(boolean mute)**

Index

A

addCommand(Command)
of javax.microedition.lcdui.Alert 132
of javax.microedition.lcdui.Displayable 219
of javax.microedition.lcdui.Item 295
of javax.microedition.lcdui.Spacer 317
addPlayerListener(PlayerListener)
of javax.microedition.media.Player 411
addRecord(byte[], int, int)
of javax.microedition.rms.RecordStore 483
addRecordListener(RecordListener)
of javax.microedition.rms.RecordStore 483
ALARM
of javax.microedition.lcdui.AlertType 137
ALERT
of javax.microedition.lcdui.Display 207
Alert
of javax.microedition.lcdui 128
Alert(String)
of javax.microedition.lcdui.Alert 131
Alert(String, String, Image, AlertType)
of javax.microedition.lcdui.Alert 131
AlertType
of javax.microedition.lcdui 136
AlertType()
of javax.microedition.lcdui.AlertType 137
ANY
of javax.microedition.lcdui.TextField 334
append(Image)
of javax.microedition.lcdui.Form 236
append(Item)
of javax.microedition.lcdui.Form 236
append(Layer)
of javax.microedition.lcdui.game.LayerManager 362
append(String)
of javax.microedition.lcdui.Form 237
append(String, Image)
of javax.microedition.lcdui.Choice 159
of javax.microedition.lcdui.ChoiceGroup 168
of javax.microedition.lcdui.List 307
AUTHMODE_ANY
of javax.microedition.rms.RecordStore 483
AUTHMODE_PRIVATE
of javax.microedition.rms.RecordStore 483

B

BACK
of javax.microedition.lcdui.Command 178
BAD_EXTENSIONS
of javax.microedition.pki.CertificateException 459
BASELINE
of javax.microedition.lcdui.Graphics 253
BLOCK_END
of javax.microedition.media.control.ToneControl 425
BLOCK_START
of javax.microedition.media.control.ToneControl 425
BOTTOM
of javax.microedition.lcdui.Graphics 253
BROKEN_CHAIN
of javax.microedition.pki.CertificateException 459
BUTTON
of javax.microedition.lcdui.Item 291

C

C4
of javax.microedition.media.control.ToneControl 425
callSerially(Runnable)
of javax.microedition.lcdui.Display 210
CANCEL
of javax.microedition.lcdui.Command 178
cancel()
of java.util.Timer 41
of java.util.TimerTask 47
Canvas
of javax.microedition.lcdui 139
Canvas()
of javax.microedition.lcdui.Canvas 147
Certificate
of javax.microedition.pki 455
CERTIFICATE_CHAIN_TOO_LONG
of javax.microedition.pki.CertificateException 459
CertificateException
of javax.microedition.pki 458
CertificateException(Certificate, byte)
of javax.microedition.pki.CertificateException 461

CertificateException(String, Certificate, byte)
of javax.microedition.pki.CertificateException 461
charsWidth(char[], int, int)
of javax.microedition.lcdui.Font 226
charWidth(char)
of javax.microedition.lcdui.Font 227
checkPermission(String)
of javax.microedition.midlet.MIDlet 445
Choice
of javax.microedition.lcdui 155
CHOICE_GROUP_ELEMENT
of javax.microedition.lcdui.Display 208
ChoiceGroup
of javax.microedition.lcdui 166
ChoiceGroup(String, int)
of javax.microedition.lcdui.ChoiceGroup 167
ChoiceGroup(String, int, String[], Image[])
of javax.microedition.lcdui.ChoiceGroup 168
clipRect(int, int, int, int)
of javax.microedition.lcdui.Graphics 254
close()
of javax.microedition.media.Player 411
CLOSED
of javax.microedition.media.Player 409
of javax.microedition.media.PlayerListener 417
closeRecordStore()
of javax.microedition.rms.RecordStore 484
collidesWith(Image, int, int, boolean)
of javax.microedition.lcdui.game.Sprite 374
collidesWith(Sprite, boolean)
of javax.microedition.lcdui.game.Sprite 374
collidesWith(TiledLayer, boolean)
of javax.microedition.lcdui.game.Sprite 375
COLOR_BACKGROUND
of javax.microedition.lcdui.Display 208
COLOR_BORDER
of javax.microedition.lcdui.Display 208
COLOR_FOREGROUND
of javax.microedition.lcdui.Display 208
COLOR_HIGHLIGHTED_BACKGROUND
of javax.microedition.lcdui.Display 209
COLOR_HIGHLIGHTED_BORDER
of javax.microedition.lcdui.Display 209
COLOR_HIGHLIGHTED_FOREGROUND
of javax.microedition.lcdui.Display 209
Command
of javax.microedition.lcdui 175
Command(String, int, int)
of javax.microedition.lcdui.Command 180
Command(String, String, int, int)
of javax.microedition.lcdui.Command 181
commandAction(Command, Displayable)
of javax.microedition.lcdui.CommandListener 183
commandAction(Command, Item)
of javax.microedition.lcdui.ItemCommandListener 300
CommandListener
of javax.microedition.lcdui 183
CommConnection
of javax.microedition.io 55
compare(byte[], byte[])
of javax.microedition.rms.RecordComparator 472
CONFIRMATION
of javax.microedition.lcdui.AlertType 137
Connector
of javax.microedition.io 60
CONSTRAINT_MASK
of javax.microedition.lcdui.TextField 335
CONTINUOUS_IDLE
of javax.microedition.lcdui.Gauge 242
CONTINUOUS_RUNNING
of javax.microedition.lcdui.Gauge 242
Control
of javax.microedition.media 396
Controllable
of javax.microedition.media 397
copyArea(int, int, int, int, int, int, int)
of javax.microedition.lcdui.Graphics 255
createAnimatedTile(int)
of javax.microedition.lcdui.game.TiledLayer 386
createImage(byte[], int, int)
of javax.microedition.lcdui.Image 273
createImage(Image)
of javax.microedition.lcdui.Image 274
createImage(Image, int, int, int, int, int)
of javax.microedition.lcdui.Image 274
createImage(InputStream)
of javax.microedition.lcdui.Image 275
createImage(int, int)
of javax.microedition.lcdui.Image 276
createImage(String)
of javax.microedition.lcdui.Image 276
createPlayer(InputStream, String)
of javax.microedition.media.Manager 401

createPlayer(String)
of javax.microedition.media.Manager 401
createRGBImage(int[], int, int, boolean)
of javax.microedition.lcdui.Image 276
CustomItem
of javax.microedition.lcdui 184
CustomItem(String)
of javax.microedition.lcdui.CustomItem 190

D

DATE
of javax.microedition.lcdui.DateField 202
DATE_TIME
of javax.microedition.lcdui.DateField 202
DateField
of javax.microedition.lcdui 201
DateField(String, int)
of javax.microedition.lcdui.DateField 203
DateField(String, int, TimeZone)
of javax.microedition.lcdui.DateField 203
deallocate()
of javax.microedition.media.Player 411
DECIMAL
of javax.microedition.lcdui.TextField 335
defineCollisionRectangle(int, int, int, int)
of javax.microedition.lcdui.game.Sprite 375
defineReferencePixel(int, int)
of javax.microedition.lcdui.game.Sprite 376
DELAY
of javax.microedition.io.SocketConnection 110
delete(int)
of javax.microedition.lcdui.Choice 160
of javax.microedition.lcdui.ChoiceGroup 169
of javax.microedition.lcdui.Form 237
of javax.microedition.lcdui.List 307
delete(int, int)
of javax.microedition.lcdui.TextBox 325
of javax.microedition.lcdui.TextField 339
deleteAll()
of javax.microedition.lcdui.Choice 160
of javax.microedition.lcdui.ChoiceGroup 169
of javax.microedition.lcdui.Form 237
of javax.microedition.lcdui.List 308
deleteRecord(int)
of javax.microedition.rms.RecordStore 484
deleteRecordStore(String)
of javax.microedition.rms.RecordStore 484
destroy()
of javax.microedition.rms.RecordEnumeration 474
destroyApp(boolean)
of javax.microedition.midlet.MIDlet 445
DEVICE_AVAILABLE
of javax.microedition.media.PlayerListener 417
DEVICE_UNAVAILABLE
of javax.microedition.media.PlayerListener 417
DISMISS_COMMAND
of javax.microedition.lcdui.Alert 130
Display
of javax.microedition.lcdui 205
Displayable
of javax.microedition.lcdui 218
DOTTED
of javax.microedition.lcdui.Graphics 253
DOWN
of javax.microedition.lcdui.Canvas 143
DOWN_PRESSED
ofjavax.microedition.lcdui.game.GameCanvas 351
drawArc(int, int, int, int, int, int)
of javax.microedition.lcdui.Graphics 256
drawChar(char, int, int, int)
of javax.microedition.lcdui.Graphics 256
drawChars(char[], int, int, int, int, int)
of javax.microedition.lcdui.Graphics 257
drawImage(Image, int, int, int)
of javax.microedition.lcdui.Graphics 257
drawLine(int, int, int, int)
of javax.microedition.lcdui.Graphics 258
drawRect(int, int, int, int)
of javax.microedition.lcdui.Graphics 258
drawRegion(Image, int, int, int, int, int, int, int, int)
of javax.microedition.lcdui.Graphics 258
drawRGB(int[], int, int, int, int, int, int, boolean)
of javax.microedition.lcdui.Graphics 260
drawRoundRect(int, int, int, int, int, int)
of javax.microedition.lcdui.Graphics 261
drawString(String, int, int, int)
of javax.microedition.lcdui.Graphics 261
drawSubstring(String, int, int, int, int, int)
of javax.microedition.lcdui.Graphics 262
DURATION_UPDATED
of javax.microedition.media.PlayerListener 418

E

EMAILADDR
of javax.microedition.lcdui.TextField 335
END_OF_MEDIA
of javax.microedition.media.PlayerListener 418
enumerateRecords(RecordFilter, RecordComparator, boolean)
of javax.microedition.rms.RecordStore 485
EQUIVALENT
of javax.microedition.rms.RecordComparator 471
ERROR
of javax.microedition.lcdui.AlertType 137
of javax.microedition.media.PlayerListener 418
EXCLUSIVE
of javax.microedition.lcdui.Choice 158
EXIT
of javax.microedition.lcdui.Command 179
EXPIRED
of javax.microedition.pki.CertificateException 459

F

FACE_MONOSPACE
of javax.microedition.lcdui.Font 224
FACE_PROPORTIONAL
of javax.microedition.lcdui.Font 224
FACE_SYSTEM
of javax.microedition.lcdui.Font 225
fillArc(int, int, int, int, int, int)
of javax.microedition.lcdui.Graphics 262
fillCells(int, int, int, int, int)
of javax.microedition.lcdui.game.TiledLayer 386
fillRect(int, int, int, int)
of javax.microedition.lcdui.Graphics 263
fillRoundRect(int, int, int, int, int, int)
of javax.microedition.lcdui.Graphics 263
fillTriangle(int, int, int, int, int, int)
of javax.microedition.lcdui.Graphics 264
FIRE
of javax.microedition.lcdui.Canvas 144
FIRE_PRESSED
of javax.microedition.lcdui.game.GameCanvas 351
flashBacklight(int)
of javax.microedition.lcdui.Display 210
flushGraphics()
of javax.microedition.lcdui.game.GameCanvas 353
flushGraphics(int, int, int, int)
of javax.microedition.lcdui.game.GameCanvas 353
FOLLOWS
of javax.microedition.rms.RecordComparator 471
Font
of javax.microedition.lcdui 223
FONT_INPUT_TEXT
of javax.microedition.lcdui.Font 225
FONT_STATIC_TEXT
of javax.microedition.lcdui.Font 225
FOREVER
of javax.microedition.lcdui.Alert 131
Form
of javax.microedition.lcdui 231
Form(String)
of javax.microedition.lcdui.Form 235
Form(String, Item[])
of javax.microedition.lcdui.Form 236

G

GAME_A
of javax.microedition.lcdui.Canvas 144
GAME_A_PRESSED
of javax.microedition.lcdui.game.GameCanvas 351
GAME_B
of javax.microedition.lcdui.Canvas 144
GAME_B_PRESSED
of javax.microedition.lcdui.game.GameCanvas 351
GAME_C
of javax.microedition.lcdui.Canvas 144
GAME_C_PRESSED
of javax.microedition.lcdui.game.GameCanvas 351
GAME_D
of javax.microedition.lcdui.Canvas 144
GAME_D_PRESSED
of javax.microedition.lcdui.game.GameCanvas 351
GameCanvas
of javax.microedition.lcdui.game 349

GameCanvas(boolean)
of javax.microedition.lcdui.game.GameCanvas 352
Gauge
of javax.microedition.lcdui 240
Gauge(String, boolean, int, int)
of javax.microedition.lcdui.Gauge 244
GET
of javax.microedition.io.HttpConnection 72
get(int)
of javax.microedition.lcdui.Form 237
getAddress()
of javax.microedition.io.SocketConnection 110
getAltText()
of javax.microedition.lcdui.ImageItem 285
getAnimatedTile(int)
of javax.microedition.lcdui.game.TiledLayer 386
getAppearanceMode()
of javax.microedition.lcdui.ImageItem 285
of javax.microedition.lcdui.StringItem 321
getAppProperty(String)
of javax.microedition.midlet.MIDlet 446
getBaselinePosition()
of javax.microedition.lcdui.Font 227
getBaudRate()
of javax.microedition.io.CommConnection 58
getBestImageHeight(int)
of javax.microedition.lcdui.Display 211
getBestImageWidth(int)
of javax.microedition.lcdui.Display 212
getBlueComponent()
of javax.microedition.lcdui.Graphics 264
getBorderStyle(boolean)
of javax.microedition.lcdui.Display 212
getCaretPosition()
of javax.microedition.lcdui.TextBox 325
of javax.microedition.lcdui.TextField 339
getCell(int, int)
of javax.microedition.lcdui.game.TiledLayer 387
getCellHeight()
of javax.microedition.lcdui.game.TiledLayer 387
getCellWidth()
of javax.microedition.lcdui.game.TiledLayer 387
getCertificate()
of javax.microedition.pki.CertificateException 461
getChars(char[])
of javax.microedition.lcdui.TextBox 325
of javax.microedition.lcdui.TextField 340
getCipherSuite()
of javax.microedition.io.SecurityInfo 103
getClipHeight()
of javax.microedition.lcdui.Graphics 264
getClipWidth()
of javax.microedition.lcdui.Graphics 264
getClipX()
of javax.microedition.lcdui.Graphics 265
getClipY()
of javax.microedition.lcdui.Graphics 265
getColor()
of javax.microedition.lcdui.Graphics 265
getColor(int)
of javax.microedition.lcdui.Display 212
getColumns()
of javax.microedition.lcdui.game.TiledLayer 387
getCommandType()
of javax.microedition.lcdui.Command 181
getConstraints()
of javax.microedition.lcdui.TextBox 326
of javax.microedition.lcdui.TextField 340
getContentType()
of javax.microedition.media.Player 411
getControl(String)
of javax.microedition.media.Controllable 397
getControls()
of javax.microedition.media.Controllable 398
getCurrent()
of javax.microedition.lcdui.Display 213
getDate()
of javax.microedition.io.HttpConnection 78
of javax.microedition.lcdui.DateField 203
getDefaultFont()
of javax.microedition.lcdui.Font 227
getDefaultTimeout()
of javax.microedition.lcdui.Alert 132
getDisplay(MIDlet)
of javax.microedition.lcdui.Display 213
getDisplayColor(int)
of javax.microedition.lcdui.Graphics 265
getDuration()
of javax.microedition.media.Player 412
getExpiration()
of javax.microedition.io.HttpConnection 79

getFace()
of javax.microedition.lcdui.Font 228
getFile()
of javax.microedition.io.HttpConnection 79
getFilter(String)
of javax.microedition.io.PushRegistry 96
getFitPolicy()
of javax.microedition.lcdui.Choice 160
of javax.microedition.lcdui.ChoiceGroup 169
of javax.microedition.lcdui.List 308
getFont()
of javax.microedition.lcdui.Graphics 266
of javax.microedition.lcdui.StringItem 321
getFont(int)
of javax.microedition.lcdui.Choice 160
of javax.microedition.lcdui.ChoiceGroup 169
of javax.microedition.lcdui.Font 228
of javax.microedition.lcdui.List 308
getFont(int, int, int)
of javax.microedition.lcdui.Font 228
getFrame()
of javax.microedition.lcdui.game.Sprite 376
getFrameSequenceLength()
of javax.microedition.lcdui.game.Sprite 377
getGameAction(int)
of javax.microedition.lcdui.Canvas 147
of javax.microedition.lcdui.CustomItem 190
getGraphics()
of javax.microedition.lcdui.game.GameCanvas 353
of javax.microedition.lcdui.Image 277
getGrayScale()
of javax.microedition.lcdui.Graphics 266
getGreenComponent()
of javax.microedition.lcdui.Graphics 266
getHeaderField(int)
of javax.microedition.io.HttpConnection 79
getHeaderField(String)
of javax.microedition.io.HttpConnection 79
getHeaderFieldDate(String, long)
of javax.microedition.io.HttpConnection 80
getHeaderFieldInt(String, int)
of javax.microedition.io.HttpConnection 80
getHeaderFieldKey(int)
of javax.microedition.io.HttpConnection 80
getHeight()
of javax.microedition.lcdui.Canvas 148
of javax.microedition.lcdui.Displayable 219
of javax.microedition.lcdui.Font 228
of javax.microedition.lcdui.Form 238
of javax.microedition.lcdui.game.Layer 357
of javax.microedition.lcdui.Image 278
getHost()
of javax.microedition.io.HttpConnection 81
getImage()
of javax.microedition.lcdui.Alert 132
of javax.microedition.lcdui.ImageItem 285
getImage(int)
of javax.microedition.lcdui.Choice 161
of javax.microedition.lcdui.ChoiceGroup 170
of javax.microedition.lcdui.List 309
getIndicator()
of javax.microedition.lcdui.Alert 132
getInputMode()
of javax.microedition.lcdui.DateField 204
getInteractionModes()
of javax.microedition.lcdui.CustomItem 191
getIssuer()
of javax.microedition.pki.Certificate 456
getKeyCode(int)
of javax.microedition.lcdui.Canvas 148
getKeyName(int)
of javax.microedition.lcdui.Canvas 148
getKeyStates()
of javax.microedition.lcdui.game.GameCanvas 354
getLabel()
of javax.microedition.lcdui.Command 181
of javax.microedition.lcdui.Item 295
getLastModified()
of javax.microedition.io.HttpConnection 81
of javax.microedition.rms.RecordStore 486
getLayerAt(int)
of javax.microedition.lcdui.game.LayerManager 362
getLayout()
of javax.microedition.lcdui.ImageItem 285
of javax.microedition.lcdui.Item 295
getLevel()
of javax.microedition.media.control.VolumeControl 429
getLocalAddress()
of javax.microedition.io.ServerSocketConnection 106
of javax.microedition.io.SocketConnection 111
of javax.microedition.io.UDPDatagramConnection 114
getLocalPort()
of javax.microedition.io.ServerSocketConnec-

tion 107
of javax.microedition.io.SocketConnection 111
of javax.microedition.io.UDPDatagramConnection 114
getLongLabel()
of javax.microedition.lcdui.Command 181
getMaxSize()
of javax.microedition.lcdui.TextBox 326
of javax.microedition.lcdui.TextField 340
getMaxValue()
of javax.microedition.lcdui.Gauge 244
getMediaTime()
of javax.microedition.media.Player 412
getMIDlet(String)
of javax.microedition.io.PushRegistry 97
getMinContentHeight()
of javax.microedition.lcdui.CustomItem 191
getMinContentWidth()
of javax.microedition.lcdui.CustomItem 191
getMinimumHeight()
of javax.microedition.lcdui.Item 296
getMinimumWidth()
of javax.microedition.lcdui.Item 296
getName()
of javax.microedition.rms.RecordStore 486
getNextRecordID()
of javax.microedition.rms.RecordStore 486
getNotAfter()
of javax.microedition.pki.Certificate 456
getNotBefore()
of javax.microedition.pki.Certificate 456
getNumRecords()
of javax.microedition.rms.RecordStore 486
getPort()
of javax.microedition.io.HttpConnection 81
of javax.microedition.io.HttpsConnection 87
of javax.microedition.io.SocketConnection 111
getPrefContentHeight(int)
of javax.microedition.lcdui.CustomItem 191
getPrefContentWidth(int)
of javax.microedition.lcdui.CustomItem 192
getPreferredHeight()
of javax.microedition.lcdui.Item 296
getPreferredWidth()
of javax.microedition.lcdui.Item 296
getPriority()
of javax.microedition.lcdui.Command 182
getProtocol()
of javax.microedition.io.HttpConnection 81
getProtocolName()
of javax.microedition.io.SecurityInfo 104
getProtocolVersion()
of javax.microedition.io.SecurityInfo 104
getQuery()
of javax.microedition.io.HttpConnection 81
getRawFrameCount()
of javax.microedition.lcdui.game.Sprite 377
getReason()
of javax.microedition.pki.CertificateException 462
getRecord(int)
of javax.microedition.rms.RecordStore 487
getRecord(int, byte[], int)
of javax.microedition.rms.RecordStore 487
getRecordSize(int)
of javax.microedition.rms.RecordStore 488
getRedComponent()
of javax.microedition.lcdui.Graphics 266
getRef()
of javax.microedition.io.HttpConnection 82
getRefPixelX()
of javax.microedition.lcdui.game.Sprite 377
getRefPixelY()
of javax.microedition.lcdui.game.Sprite 377
getRequestMethod()
of javax.microedition.io.HttpConnection 82
getRequestProperty(String)
of javax.microedition.io.HttpConnection 82
getResponseCode()
of javax.microedition.io.HttpConnection 82
getResponseMessage()
of javax.microedition.io.HttpConnection 83
getRGB(int[], int, int, int, int, int, int)
of javax.microedition.lcdui.Image 278
getRows()
of javax.microedition.lcdui.game.TiledLayer 388
getSecurityInfo()
of javax.microedition.io.HttpsConnection 88
of javax.microedition.io.SecureConnection 102
getSelectedFlags(boolean[])
of javax.microedition.lcdui.Choice 161
of javax.microedition.lcdui.ChoiceGroup 170
of javax.microedition.lcdui.List 309
getSelectedIndex()
of javax.microedition.lcdui.Choice 162

of javax.microedition.lcdui.ChoiceGroup 171
of javax.microedition.lcdui.List 309
getSerialNumber()
of javax.microedition.pki.Certificate 457
getServerCertificate()
of javax.microedition.io.SecurityInfo 104
getSigAlgName()
of javax.microedition.pki.Certificate 457
getSize()
of javax.microedition.lcdui.Font 229
of javax.microedition.lcdui.game.LayerManager 362
of javax.microedition.rms.RecordStore 488
getSizeAvailable()
of javax.microedition.rms.RecordStore 488
getSocketOption(byte)
of javax.microedition.io.SocketConnection 112
getState()
of javax.microedition.media.Player 412
getString()
of javax.microedition.lcdui.Alert 133
of javax.microedition.lcdui.TextBox 326
of javax.microedition.lcdui.TextField 340
of javax.microedition.lcdui.Ticker 346
getString(int)
of javax.microedition.lcdui.Choice 162
of javax.microedition.lcdui.ChoiceGroup 171
of javax.microedition.lcdui.List 310
getStrokeStyle()
of javax.microedition.lcdui.Graphics 266
getStyle()
of javax.microedition.lcdui.Font 229
getSubject()
of javax.microedition.pki.Certificate 457
getSupportedContentTypes(String)
of javax.microedition.media.Manager 402
getSupportedProtocols(String)
of javax.microedition.media.Manager 402
getText()
of javax.microedition.lcdui.StringItem 321
getTicker()
of javax.microedition.lcdui.Displayable 219
getTimeout()
of javax.microedition.lcdui.Alert 133
getTitle()
of javax.microedition.lcdui.Displayable 220
getTranslateX()
of javax.microedition.lcdui.Graphics 267
getTranslateY()
of javax.microedition.lcdui.Graphics 267
getType()
of javax.microedition.lcdui.Alert 133
of javax.microedition.pki.Certificate 457
getURL()
of javax.microedition.io.HttpConnection 83
getValue()
of javax.microedition.lcdui.Gauge 245
getVersion()
of javax.microedition.pki.Certificate 457
of javax.microedition.rms.RecordStore 488
getWidth()
of javax.microedition.lcdui.Canvas 149
of javax.microedition.lcdui.Displayable 220
of javax.microedition.lcdui.Form 238
of javax.microedition.lcdui.game.Layer 357
of javax.microedition.lcdui.Image 279
getX()
of javax.microedition.lcdui.game.Layer 357
getY()
of javax.microedition.lcdui.game.Layer 357
Graphics
of javax.microedition.lcdui 247

H

hasNextElement()
of javax.microedition.rms.RecordEnumeration 474
hasPointerEvents()
of javax.microedition.lcdui.Canvas 149
hasPointerMotionEvents()
of javax.microedition.lcdui.Canvas 149
hasPreviousElement()
of javax.microedition.rms.RecordEnumeration 474
hasRepeatEvents()
of javax.microedition.lcdui.Canvas 149
HCENTER
of javax.microedition.lcdui.Graphics 253
HEAD
of javax.microedition.io.HttpConnection 72
HELP
of javax.microedition.lcdui.Command 179
hideNotify()
of javax.microedition.lcdui.Canvas 149
of javax.microedition.lcdui.CustomItem 192
HTTP_ACCEPTED
of javax.microedition.io.HttpConnection 72

HTTP_BAD_GATEWAY
of javax.microedition.io.HttpConnection 72
HTTP_BAD_METHOD
of javax.microedition.io.HttpConnection 73
HTTP_BAD_REQUEST
of javax.microedition.io.HttpConnection 73
HTTP_CLIENT_TIMEOUT
of javax.microedition.io.HttpConnection 73
HTTP_CONFLICT
of javax.microedition.io.HttpConnection 73
HTTP_CREATED
of javax.microedition.io.HttpConnection 73
HTTP_ENTITY_TOO_LARGE
of javax.microedition.io.HttpConnection 73
HTTP_EXPECT_FAILED
of javax.microedition.io.HttpConnection 73
HTTP_FORBIDDEN
of javax.microedition.io.HttpConnection 74
HTTP_GATEWAY_TIMEOUT
of javax.microedition.io.HttpConnection 74
HTTP_GONE
of javax.microedition.io.HttpConnection 74
HTTP_INTERNAL_ERROR
of javax.microedition.io.HttpConnection 74
HTTP_LENGTH_REQUIRED
of javax.microedition.io.HttpConnection 74
HTTP_MOVED_PERM
of javax.microedition.io.HttpConnection 74
HTTP_MOVED_TEMP
of javax.microedition.io.HttpConnection 75
HTTP_MULT_CHOICE
of javax.microedition.io.HttpConnection 75
HTTP_NO_CONTENT
of javax.microedition.io.HttpConnection 75
HTTP_NOT_ACCEPTABLE
of javax.microedition.io.HttpConnection 75
HTTP_NOT_AUTHORITATIVE
of javax.microedition.io.HttpConnection 75
HTTP_NOT_FOUND
of javax.microedition.io.HttpConnection 75
HTTP_NOT_IMPLEMENTED
of javax.microedition.io.HttpConnection 76
HTTP_NOT_MODIFIED
of javax.microedition.io.HttpConnection 76
HTTP_OK
of javax.microedition.io.HttpConnection 76
HTTP_PARTIAL
of javax.microedition.io.HttpConnection 76
HTTP_PAYMENT_REQUIRED
of javax.microedition.io.HttpConnection 76
HTTP_PRECON_FAILED
of javax.microedition.io.HttpConnection 76
HTTP_PROXY_AUTH
of javax.microedition.io.HttpConnection 76
HTTP_REQ_TOO_LONG
of javax.microedition.io.HttpConnection 77
HTTP_RESET
of javax.microedition.io.HttpConnection 77
HTTP_SEE_OTHER
of javax.microedition.io.HttpConnection 77
HTTP_TEMP_REDIRECT
of javax.microedition.io.HttpConnection 77
HTTP_UNAUTHORIZED
of javax.microedition.io.HttpConnection 77
HTTP_UNAVAILABLE
of javax.microedition.io.HttpConnection 77
HTTP_UNSUPPORTED_RANGE
of javax.microedition.io.HttpConnection 77
HTTP_UNSUPPORTED_TYPE
of javax.microedition.io.HttpConnection 78
HTTP_USE_PROXY
of javax.microedition.io.HttpConnection 78
HTTP_VERSION
of javax.microedition.io.HttpConnection 78
HttpConnection
of javax.microedition.io 65
HttpsConnection
of javax.microedition.io 85
HYPERLINK
of javax.microedition.lcdui.Item 292

I

IllegalStateException
of java.lang 37
IllegalStateException()
of java.lang.IllegalStateException 37
IllegalStateException(String)
of java.lang.IllegalStateException 38
Image
of javax.microedition.lcdui 270
ImageItem
of javax.microedition.lcdui 281
ImageItem(String, Image, int, String)
of javax.microedition.lcdui.ImageItem 283
ImageItem(String, Image, int, String, int)
of javax.microedition.lcdui.ImageItem 284
IMPLICIT
of javax.microedition.lcdui.Choice 158

INAPPROPRIATE_KEY_USAGE
of javax.microedition.pki.CertificateException 459
INCREMENTAL_IDLE
of javax.microedition.lcdui.Gauge 243
INCREMENTAL_UPDATING
of javax.microedition.lcdui.Gauge 243
INDEFINITE
of javax.microedition.lcdui.Gauge 243
INFO
of javax.microedition.lcdui.AlertType 137
INITIAL_CAPS_SENTENCE
of javax.microedition.lcdui.TextField 336
INITIAL_CAPS_WORD
of javax.microedition.lcdui.TextField 336
insert(char[], int, int, int)
of javax.microedition.lcdui.TextBox 326
of javax.microedition.lcdui.TextField 341
insert(int, Item)
of javax.microedition.lcdui.Form 238
insert(int, String, Image)
of javax.microedition.lcdui.Choice 162
of javax.microedition.lcdui.ChoiceGroup 171
of javax.microedition.lcdui.List 310
insert(Layer, int)
of javax.microedition.lcdui.game.LayerManager 363
insert(String, int)
of javax.microedition.lcdui.TextBox 327
of javax.microedition.lcdui.TextField 341
invalidate()
of javax.microedition.lcdui.CustomItem 192
InvalidRecordIDException
of javax.microedition.rms 469
InvalidRecordIDException()
of javax.microedition.rms.InvalidRecordIDException 469
InvalidRecordIDException(String)
of javax.microedition.rms.InvalidRecordIDException 470
isBold()
of javax.microedition.lcdui.Font 229
isColor()
of javax.microedition.lcdui.Display 213
isDoubleBuffered()
of javax.microedition.lcdui.Canvas 150
isInteractive()
of javax.microedition.lcdui.Gauge 245
isItalic()
of javax.microedition.lcdui.Font 229
isKeptUpdated()
of javax.microedition.rms.RecordEnumeration 474
isMutable()
of javax.microedition.lcdui.Image 280
isMuted()
of javax.microedition.media.control.VolumeControl 429
isPlain()
of javax.microedition.lcdui.Font 229
isSelected(int)
of javax.microedition.lcdui.Choice 163
of javax.microedition.lcdui.ChoiceGroup 172
of javax.microedition.lcdui.List 310
isShown()
of javax.microedition.lcdui.Displayable 220
isUnderlined()
of javax.microedition.lcdui.Font 230
isVisible()
of javax.microedition.lcdui.game.Layer 357
ITEM
of javax.microedition.lcdui.Command 179
Item
of javax.microedition.lcdui 287
ItemCommandListener
of javax.microedition.lcdui 300
itemStateChanged(Item)
of javax.microedition.lcdui.ItemStateListener 301
ItemStateListener
of javax.microedition.lcdui 301

J

java.applet - package 515

K

KEEPALIVE
of javax.microedition.io.SocketConnection 110
keepUpdated(boolean)
of javax.microedition.rms.RecordEnumeration 474
KEY_NUM0
of javax.microedition.lcdui.Canvas 144
KEY_NUM1
of javax.microedition.lcdui.Canvas 145
KEY_NUM2
of javax.microedition.lcdui.Canvas 145

KEY_NUM3
of javax.microedition.lcdui.Canvas 145
KEY_NUM4
of javax.microedition.lcdui.Canvas 145
KEY_NUM5
of javax.microedition.lcdui.Canvas 145
KEY_NUM6
of javax.microedition.lcdui.Canvas 145
KEY_NUM7
of javax.microedition.lcdui.Canvas 146
KEY_NUM8
of javax.microedition.lcdui.Canvas 146
KEY_NUM9
of javax.microedition.lcdui.Canvas 146
KEY_POUND
of javax.microedition.lcdui.Canvas 146
KEY_PRESS
of javax.microedition.lcdui.CustomItem 188
KEY_RELEASE
of javax.microedition.lcdui.CustomItem 188
KEY_REPEAT
of javax.microedition.lcdui.CustomItem 189
KEY_STAR
of javax.microedition.lcdui.Canvas 146
keyPressed(int)
of javax.microedition.lcdui.Canvas 150
of javax.microedition.lcdui.CustomItem 193
keyReleased(int)
of javax.microedition.lcdui.Canvas 150
of javax.microedition.lcdui.CustomItem 193
keyRepeated(int)
of javax.microedition.lcdui.Canvas 150
of javax.microedition.lcdui.CustomItem 193

L

Layer
of javax.microedition.lcdui.game 356
LayerManager
of javax.microedition.lcdui.game 360
LayerManager()
of javax.microedition.lcdui.game.LayerManager 362
LAYOUT_2
of javax.microedition.lcdui.Item 292
LAYOUT_BOTTOM
of javax.microedition.lcdui.Item 292
LAYOUT_CENTER
of javax.microedition.lcdui.ImageItem 282
of javax.microedition.lcdui.Item 292
LAYOUT_DEFAULT
of javax.microedition.lcdui.ImageItem 282
of javax.microedition.lcdui.Item 292
LAYOUT_EXPAND
of javax.microedition.lcdui.Item 293
LAYOUT_LEFT
of javax.microedition.lcdui.ImageItem 283
of javax.microedition.lcdui.Item 293
LAYOUT_NEWLINE_AFTER
of javax.microedition.lcdui.ImageItem 283
of javax.microedition.lcdui.Item 293
LAYOUT_NEWLINE_BEFORE
of javax.microedition.lcdui.ImageItem 283
of javax.microedition.lcdui.Item 293
LAYOUT_RIGHT
of javax.microedition.lcdui.ImageItem 283
of javax.microedition.lcdui.Item 293
LAYOUT_SHRINK
of javax.microedition.lcdui.Item 294
LAYOUT_TOP
of javax.microedition.lcdui.Item 294
LAYOUT_VCENTER
of javax.microedition.lcdui.Item 294
LAYOUT_VEXPAND
of javax.microedition.lcdui.Item 294
LAYOUT_VSHRINK
of javax.microedition.lcdui.Item 294
LEFT
of javax.microedition.lcdui.Canvas 146
of javax.microedition.lcdui.Graphics 253
LEFT_PRESSED
of javax.microedition.lcdui.game.GameCanvas 352
LINGER
of javax.microedition.io.SocketConnection 110
List
of javax.microedition.lcdui 303
List(String, int)
of javax.microedition.lcdui.List 306
List(String, int, String[], Image[])
of javax.microedition.lcdui.List 306
LIST_ELEMENT
of javax.microedition.lcdui.Display 209
listConnections(boolean)
of javax.microedition.io.PushRegistry 97
listRecordStores()
of javax.microedition.rms.RecordStore 489

M

Manager
of javax.microedition.media 399
matches(byte[])
of javax.microedition.rms.RecordFilter 478
MediaException
of javax.microedition.media 404
MediaException()
of javax.microedition.media.MediaException 404
MediaException(String)
of javax.microedition.media.MediaException 404
MIDlet
of javax.microedition.midlet 444
MIDlet()
of javax.microedition.midlet.MIDlet 445
MIDletStateChangeException
of javax.microedition.midlet 450
MIDletStateChangeException()
of javax.microedition.midlet.MIDletStateChangeException 450
MIDletStateChangeException(String)
of javax.microedition.midlet.MIDletStateChangeException 451
MISSING_SIGNATURE
of javax.microedition.pki.CertificateException 459
move(int, int)
of javax.microedition.lcdui.game.Layer 358
MULTIPLE
of javax.microedition.lcdui.Choice 158

N

nextFrame()
of javax.microedition.lcdui.game.Sprite 378
nextRecord()
of javax.microedition.rms.RecordEnumeration 475
nextRecordId()
of javax.microedition.rms.RecordEnumeration 475
NON_PREDICTIVE
of javax.microedition.lcdui.TextField 336
NONE
of javax.microedition.lcdui.CustomItem 189
NOT_YET_VALID
of javax.microedition.pki.CertificateException 460
notifyDestroyed()
of javax.microedition.midlet.MIDlet 446
notifyPaused()
of javax.microedition.midlet.MIDlet 446
notifyStateChanged()
of javax.microedition.lcdui.Item 297
numAlphaLevels()
of javax.microedition.lcdui.Display 213
numColors()
of javax.microedition.lcdui.Display 214
NUMERIC
of javax.microedition.lcdui.TextField 337
numRecords()
of javax.microedition.rms.RecordEnumeration 476

O

OK
of javax.microedition.lcdui.Command 179
open(String)
of javax.microedition.io.Connector 62
open(String, int)
of javax.microedition.io.Connector 62
open(String, int, boolean)
of javax.microedition.io.Connector 62
openDataInputStream(String)
of javax.microedition.io.Connector 63
openDataOutputStream(String)
of javax.microedition.io.Connector 63
openInputStream(String)
of javax.microedition.io.Connector 64
openOutputStream(String)
of javax.microedition.io.Connector 64
openRecordStore(String, boolean)
of javax.microedition.rms.RecordStore 489
openRecordStore(String, boolean, int, boolean)
of javax.microedition.rms.RecordStore 490
openRecordStore(String, String, String)
of javax.microedition.rms.RecordStore 490

P

paint(Graphics)
of javax.microedition.lcdui.Canvas 151
of javax.microedition.lcdui.game.GameCanvas 355
of javax.microedition.lcdui.game.Layer 358
of javax.microedition.lcdui.game.Sprite 378

of javax.microedition.lcdui.game.TiledLayer 388
paint(Graphics, int, int)
of javax.microedition.lcdui.CustomItem 193
of javax.microedition.lcdui.game.LayerManager 363
PASSWORD
of javax.microedition.lcdui.TextField 337
pauseApp()
of javax.microedition.midlet.MIDlet 447
PHONENUMBER
of javax.microedition.lcdui.TextField 337
PLAIN
of javax.microedition.lcdui.Item 295
platformRequest(String)
of javax.microedition.midlet.MIDlet 447
PLAY_BLOCK
of javax.microedition.media.control.ToneControl 426
Player
of javax.microedition.media 406
PlayerListener
of javax.microedition.media 416
playerUpdate(Player, String, Object)
of javax.microedition.media.PlayerListener 419
playSound(Display)
of javax.microedition.lcdui.AlertType 138
playTone(int, int, int)
of javax.microedition.media.Manager 402
POINTER_DRAG
of javax.microedition.lcdui.CustomItem 189
POINTER_PRESS
of javax.microedition.lcdui.CustomItem 189
POINTER_RELEASE
of javax.microedition.lcdui.CustomItem 189
pointerDragged(int, int)
of javax.microedition.lcdui.Canvas 152
of javax.microedition.lcdui.CustomItem 194
pointerPressed(int, int)
of javax.microedition.lcdui.Canvas 152
of javax.microedition.lcdui.CustomItem 194
pointerReleased(int, int)
of javax.microedition.lcdui.Canvas 152
of javax.microedition.lcdui.CustomItem 195
POPUP
of javax.microedition.lcdui.Choice 158
POST
of javax.microedition.io.HttpConnection 78
PRECEDES
of javax.microedition.rms.RecordComparator 472
prefetch()
of javax.microedition.media.Player 412
PREFETCHED
of javax.microedition.media.Player 410
prevFrame()
of javax.microedition.lcdui.game.Sprite 378
previousRecord()
of javax.microedition.rms.RecordEnumeration 476
previousRecordId()
of javax.microedition.rms.RecordEnumeration 476
PushRegistry
of javax.microedition.io 89

R

RCVBUF
of javax.microedition.io.SocketConnection 110
READ
of javax.microedition.io.Connector 61
READ_WRITE
of javax.microedition.io.Connector 61
realize()
of javax.microedition.media.Player 413
REALIZED
of javax.microedition.media.Player 410
rebuild()
of javax.microedition.rms.RecordEnumeration 476
recordAdded(RecordStore, int)
of javax.microedition.rms.RecordListener 479
recordChanged(RecordStore, int)
of javax.microedition.rms.RecordListener 479
RecordComparator
of javax.microedition.rms 471
recordDeleted(RecordStore, int)
of javax.microedition.rms.RecordListener 480
RecordEnumeration
of javax.microedition.rms 473
RecordFilter
of javax.microedition.rms 478
RecordListener
of javax.microedition.rms 479
RecordStore
of javax.microedition.rms 481

RecordStoreException
of javax.microedition.rms 493
RecordStoreException()
of javax.microedition.rms.RecordStoreException 493
RecordStoreException(String)
of javax.microedition.rms.RecordStoreException 494
RecordStoreFullException
of javax.microedition.rms 495
RecordStoreFullException()
of javax.microedition.rms.RecordStoreFullException 495
RecordStoreFullException(String)
of javax.microedition.rms.RecordStoreFullException 496
RecordStoreNotFoundException
of javax.microedition.rms 497
RecordStoreNotFoundException()
of javax.microedition.rms.RecordStoreNotFoundException 497
RecordStoreNotFoundException(String)
of javax.microedition.rms.RecordStoreNotFoundException 498
RecordStoreNotOpenException
of javax.microedition.rms 499
RecordStoreNotOpenException()
of javax.microedition.rms.RecordStoreNotOpenException 499
RecordStoreNotOpenException(String)
of javax.microedition.rms.RecordStoreNotOpenException 500
registerAlarm(String, long)
of javax.microedition.io.PushRegistry 97
registerConnection(String, String, String)
of javax.microedition.io.PushRegistry 98
remove(Layer)
of javax.microedition.lcdui.game.LayerManager 364
removeCommand(Command)
of javax.microedition.lcdui.Alert 133
of javax.microedition.lcdui.Displayable 220
of javax.microedition.lcdui.Item 297
of javax.microedition.lcdui.List 311
removePlayerListener(PlayerListener)
of javax.microedition.media.Player 413
removeRecordListener(RecordListener)
of javax.microedition.rms.RecordStore 491
repaint()
of javax.microedition.lcdui.Canvas 152
of javax.microedition.lcdui.CustomItem 195
repaint(int, int, int, int)
of javax.microedition.lcdui.Canvas 153
of javax.microedition.lcdui.CustomItem 195
REPEAT
of javax.microedition.media.control.ToneControl 426
reset()
of javax.microedition.rms.RecordEnumeration 477
RESOLUTION
of javax.microedition.media.control.ToneControl 426
resumeRequest()
of javax.microedition.midlet.MIDlet 448
RIGHT
of javax.microedition.lcdui.Canvas 147
of javax.microedition.lcdui.Graphics 254
RIGHT_PRESSED
of javax.microedition.lcdui.game.GameCanvas 352
ROOT_CA_EXPIRED
of javax.microedition.pki.CertificateException 460
run()
of java.util.TimerTask 47

S

schedule(TimerTask, Date)
of java.util.Timer 41
schedule(TimerTask, Date, long)
of java.util.Timer 42
schedule(TimerTask, long)
of java.util.Timer 42
schedule(TimerTask, long, long)
of java.util.Timer 43
scheduleAtFixedRate(TimerTask, Date, long)
of java.util.Timer 43
scheduleAtFixedRate(TimerTask, long, long)
of java.util.Timer 44
scheduledExecutionTime()
of java.util.TimerTask 47
SCREEN
of javax.microedition.lcdui.Command 180
Screen
of javax.microedition.lcdui 315
SecureConnection
of javax.microedition.io 100

SecurityInfo
of javax.microedition.io 103
SELECT_COMMAND
of javax.microedition.lcdui.List 306
SENSITIVE
of javax.microedition.lcdui.TextField 338
ServerSocketConnection
of javax.microedition.io 105
serviceRepaints()
of javax.microedition.lcdui.Canvas 153
set(int, Item)
of javax.microedition.lcdui.Form 239
set(int, String, Image)
of javax.microedition.lcdui.Choice 163
of javax.microedition.lcdui.ChoiceGroup 172
of javax.microedition.lcdui.List 311
SET_VOLUME
of javax.microedition.media.control.ToneControl 426
setAltText(String)
of javax.microedition.lcdui.ImageItem 285
setAnimatedTile(int, int)
of javax.microedition.lcdui.game.TiledLayer 388
setBaudRate(int)
of javax.microedition.io.CommConnection 58
setCell(int, int, int)
of javax.microedition.lcdui.game.TiledLayer 389
setChars(char[], int, int)
of javax.microedition.lcdui.TextBox 328
of javax.microedition.lcdui.TextField 342
setClip(int, int, int, int)
of javax.microedition.lcdui.Graphics 267
setColor(int)
of javax.microedition.lcdui.Graphics 267
setColor(int, int, int)
of javax.microedition.lcdui.Graphics 267
setCommandListener(CommandListener)
of javax.microedition.lcdui.Alert 134
of javax.microedition.lcdui.Displayable 221
setConstraints(int)
of javax.microedition.lcdui.TextBox 328
of javax.microedition.lcdui.TextField 342
setCurrent(Alert, Displayable)
of javax.microedition.lcdui.Display 214
setCurrent(Displayable)
of javax.microedition.lcdui.Display 214
setCurrentItem(Item)
of javax.microedition.lcdui.Display 216
setDate(Date)
of javax.microedition.lcdui.DateField 204
setDefaultCommand(Command)
of javax.microedition.lcdui.Item 298
of javax.microedition.lcdui.Spacer 318
setFitPolicy(int)
of javax.microedition.lcdui.Choice 163
of javax.microedition.lcdui.ChoiceGroup 172
of javax.microedition.lcdui.List 312
setFont(Font)
of javax.microedition.lcdui.Graphics 268
of javax.microedition.lcdui.StringItem 322
setFont(int, Font)
of javax.microedition.lcdui.Choice 164
of javax.microedition.lcdui.ChoiceGroup 173
of javax.microedition.lcdui.List 312
setFrame(int)
of javax.microedition.lcdui.game.Sprite 378
setFrameSequence(int[])
of javax.microedition.lcdui.game.Sprite 379
setFullScreenMode(boolean)
of javax.microedition.lcdui.Canvas 153
setGrayScale(int)
of javax.microedition.lcdui.Graphics 268
setImage(Image)
of javax.microedition.lcdui.Alert 134
of javax.microedition.lcdui.ImageItem 286
setImage(Image, int, int)
of javax.microedition.lcdui.game.Sprite 379
setIndicator(Gauge)
of javax.microedition.lcdui.Alert 134
setInitialInputMode(String)
of javax.microedition.lcdui.TextBox 328
of javax.microedition.lcdui.TextField 343
setInputMode(int)
of javax.microedition.lcdui.DateField 204
setItemCommandListener(ItemCommandListener)
of javax.microedition.lcdui.Item 298
setItemStateListener(ItemStateListener)
of javax.microedition.lcdui.Form 239
setLabel(String)
of javax.microedition.lcdui.Item 298
of javax.microedition.lcdui.Spacer 318
setLayout(int)
of javax.microedition.lcdui.ImageItem 286
of javax.microedition.lcdui.Item 299
setLevel(int)
of javax.microedition.media.control.VolumeControl 429

setLoopCount(int)
of javax.microedition.media.Player 414
setMaxSize(int)
of javax.microedition.lcdui.TextBox 329
of javax.microedition.lcdui.TextField 343
setMaxValue(int)
of javax.microedition.lcdui.Gauge 245
setMediaTime(long)
of javax.microedition.media.Player 414
setMinimumSize(int, int)
of javax.microedition.lcdui.Spacer 318
setMode(int, boolean)
of javax.microedition.rms.RecordStore 491
setMute(boolean)
of javax.microedition.media.control.VolumeControl 429
setPosition(int, int)
of javax.microedition.lcdui.game.Layer 358
setPreferredSize(int, int)
of javax.microedition.lcdui.Item 299
setRecord(int, byte[], int, int)
of javax.microedition.rms.RecordStore 492
setRefPixelPosition(int, int)
of javax.microedition.lcdui.game.Sprite 380
setRequestMethod(String)
of javax.microedition.io.HttpConnection 83
setRequestProperty(String, String)
of javax.microedition.io.HttpConnection 84
setSelectCommand(Command)
of javax.microedition.lcdui.List 312
setSelectedFlags(boolean[])
of javax.microedition.lcdui.Choice 164
of javax.microedition.lcdui.ChoiceGroup 173
of javax.microedition.lcdui.List 313
setSelectedIndex(int, boolean)
of javax.microedition.lcdui.Choice 164
of javax.microedition.lcdui.ChoiceGroup 174
of javax.microedition.lcdui.List 313
setSequence(byte[])
of javax.microedition.media.control.ToneControl 427
setSocketOption(byte, int)
of javax.microedition.io.SocketConnection 112
setStaticTileSet(Image, int, int)
of javax.microedition.lcdui.game.TiledLayer 389
setString(String)
of javax.microedition.lcdui.Alert 135
of javax.microedition.lcdui.TextBox 329
of javax.microedition.lcdui.TextField 343
of javax.microedition.lcdui.Ticker 346
setStrokeStyle(int)
of javax.microedition.lcdui.Graphics 268
setText(String)
of javax.microedition.lcdui.StringItem 322
setTicker(Ticker)
of javax.microedition.lcdui.Displayable 221
setTimeout(int)
of javax.microedition.lcdui.Alert 135
setTitle(String)
of javax.microedition.lcdui.Displayable 221
setTransform(int)
of javax.microedition.lcdui.game.Sprite 380
setType(AlertType)
of javax.microedition.lcdui.Alert 135
setValue(int)
of javax.microedition.lcdui.Gauge 246
setViewWindow(int, int, int, int)
of javax.microedition.lcdui.game.LayerManager 364
setVisible(boolean)
of javax.microedition.lcdui.game.Layer 358
showNotify()
of javax.microedition.lcdui.Canvas 154
of javax.microedition.lcdui.CustomItem 195
SILENCE
of javax.microedition.media.control.ToneControl 426
SITENAME_MISMATCH
of javax.microedition.pki.CertificateException 460
size()
of javax.microedition.lcdui.Choice 165
of javax.microedition.lcdui.ChoiceGroup 174
of javax.microedition.lcdui.Form 239
of javax.microedition.lcdui.List 314
of javax.microedition.lcdui.TextBox 329
of javax.microedition.lcdui.TextField 343
SIZE_LARGE
of javax.microedition.lcdui.Font 225
SIZE_MEDIUM
of javax.microedition.lcdui.Font 225
SIZE_SMALL
of javax.microedition.lcdui.Font 226
sizeChanged(int, int)
of javax.microedition.lcdui.Canvas 154
of javax.microedition.lcdui.CustomItem 196
of javax.microedition.lcdui.Displayable 222

SNDBUF
of javax.microedition.io.SocketConnection 110
SocketConnection
of javax.microedition.io 108
SOLID
of javax.microedition.lcdui.Graphics 254
Spacer
of javax.microedition.lcdui 316
Spacer(int, int)
of javax.microedition.lcdui.Spacer 317
Sprite
of javax.microedition.lcdui.game 365
Sprite(Image)
of javax.microedition.lcdui.game.Sprite 373
Sprite(Image, int, int)
of javax.microedition.lcdui.game.Sprite 373
Sprite(Sprite)
of javax.microedition.lcdui.game.Sprite 374
start()
of javax.microedition.media.Player 415
startApp()
of javax.microedition.midlet.MIDlet 448
STARTED
of javax.microedition.media.Player 410
of javax.microedition.media.PlayerListener 418
STOP
of javax.microedition.lcdui.Command 180
stop()
of javax.microedition.media.Player 415
STOPPED
of javax.microedition.media.PlayerListener 418
StringItem
of javax.microedition.lcdui 319
StringItem(String, String)
of javax.microedition.lcdui.StringItem 320
StringItem(String, String, int)
of javax.microedition.lcdui.StringItem 320
stringWidth(String)
of javax.microedition.lcdui.Font 230
STYLE_BOLD
of javax.microedition.lcdui.Font 226
STYLE_ITALIC
of javax.microedition.lcdui.Font 226
STYLE_PLAIN
of javax.microedition.lcdui.Font 226
STYLE_UNDERLINED
of javax.microedition.lcdui.Font 226
substringWidth(String, int, int)
of javax.microedition.lcdui.Font 230

T

TEMPO
of javax.microedition.media.control.ToneControl 426
TEXT_WRAP_DEFAULT
of javax.microedition.lcdui.Choice 158
TEXT_WRAP_OFF
of javax.microedition.lcdui.Choice 159
TEXT_WRAP_ON
of javax.microedition.lcdui.Choice 159
TextBox
of javax.microedition.lcdui 323
TextBox(String, String, int, int)
of javax.microedition.lcdui.TextBox 324
TextField
of javax.microedition.lcdui 330
TextField(String, String, int, int)
of javax.microedition.lcdui.TextField 338
Ticker
of javax.microedition.lcdui 345
Ticker(String)
of javax.microedition.lcdui.Ticker 346
TiledLayer
of javax.microedition.lcdui.game 382
TiledLayer(int, int, Image, int, int)
of javax.microedition.lcdui.game.TiledLayer 385
TIME
of javax.microedition.lcdui.DateField 202
TIME_UNKNOWN
of javax.microedition.media.Player 410
Timer
of java.util 40
Timer()
of java.util.Timer 41
TimerTask
of java.util 46
TimerTask()
of java.util.TimerTask 46
TONE_DEVICE_LOCATOR
of javax.microedition.media.Manager 400
ToneControl
of javax.microedition.media.control 422
TOP
of javax.microedition.lcdui.Graphics 254

TRANS_MIRROR
of javax.microedition.lcdui.game.Sprite 371
TRANS_MIRROR_ROT180
of javax.microedition.lcdui.game.Sprite 371
TRANS_MIRROR_ROT270
of javax.microedition.lcdui.game.Sprite 372
TRANS_MIRROR_ROT90
of javax.microedition.lcdui.game.Sprite 372
TRANS_NONE
of javax.microedition.lcdui.game.Sprite 372
TRANS_ROT180
of javax.microedition.lcdui.game.Sprite 372
TRANS_ROT270
of javax.microedition.lcdui.game.Sprite 372
TRANS_ROT90
of javax.microedition.lcdui.game.Sprite 372
translate(int, int)
of javax.microedition.lcdui.Graphics 269
traverse(int, int, int, int[])
of javax.microedition.lcdui.CustomItem 196
TRAVERSE_HORIZONTAL
of javax.microedition.lcdui.CustomItem 190
TRAVERSE_VERTICAL
of javax.microedition.lcdui.CustomItem 190
traverseOut()
of javax.microedition.lcdui.CustomItem 199

U

UDPDatagramConnection
of javax.microedition.io 113
UNAUTHORIZED_INTERMEDIATE_CA
of javax.microedition.pki.CertificateException 460
UNEDITABLE
of javax.microedition.lcdui.TextField 338
UNREALIZED
of javax.microedition.media.Player 410
UNRECOGNIZED_ISSUER
of javax.microedition.pki.CertificateException 460
unregisterConnection(String)
of javax.microedition.io.PushRegistry 99
UNSUPPORTED_PUBLIC_KEY_TYPE
of javax.microedition.pki.CertificateException 460
UNSUPPORTED_SIGALG
of javax.microedition.pki.CertificateException 460
UP
of javax.microedition.lcdui.Canvas 147
UP_PRESSED
of javax.microedition.lcdui.game.GameCanvas 352
URL
of javax.microedition.lcdui.TextField 338

V

VCENTER
of javax.microedition.lcdui.Graphics 254
VERIFICATION_FAILED
of javax.microedition.pki.CertificateException 461
VERSION
of javax.microedition.media.control.ToneControl 427
vibrate(int)
of javax.microedition.lcdui.Display 216
VOLUME_CHANGED
of javax.microedition.media.PlayerListener 419
VolumeControl
of javax.microedition.media.control 428

W

WARNING
of javax.microedition.lcdui.AlertType 137
WRITE
of javax.microedition.io.Connector 61