Mobile Information Device Profile
for Java™ 2 Micro Edition

Version 2.1

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

Java Community Process
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

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

1.1 Introduction ........................................................................................................... 13
1.2 Scope ..................................................................................................................... 13
1.3 Architecture .......................................................................................................... 14
1.4 Device Requirements ............................................................................................. 15
  1.4.1 Hardware ......................................................................................................... 16
  1.4.2 Software .......................................................................................................... 16
1.5 Specification Requirements .................................................................................... 16

2 Over The Air User Initiated Provisioning Specification ................................. 19

2.1 Over The Air User Initiated Provisioning .............................................................. 20
  2.1.1 Overview and Goals ......................................................................................... 20
  2.1.2 Functional Requirements .............................................................................. 20
  2.1.3 MIDlet Suite Discovery ................................................................................... 20
  2.1.4 MIDlet Suite Installation ............................................................................... 21
  2.1.5 MIDlet Suite Update ...................................................................................... 22
  2.1.6 MIDlet Suite Execution .................................................................................. 23
  2.1.7 MIDlet Suite Removal .................................................................................... 23
  2.1.8 Installation/Deletion Status Reports .............................................................. 23
  2.1.9 Additional Descriptor Attributes .................................................................. 24
  2.1.10 Device Identification and Request Headers ............................................... 25
    2.1.10.1 User-Agent Product Tokens ................................................................. 25
    2.1.10.2 Accept-Language Header .................................................................... 25
    2.1.10.3 Accept Header ....................................................................................... 25
    2.1.10.4 Example: HTTP Request for Application Descriptor .......................... 25
    2.1.10.5 Example: HTTP Request to Install/Update a MIDlet suite ................. 26
    2.1.10.6 Example: Install Status via HTTP Post Request .................................. 26
  2.2 MIDP Provisioning and Networking in the WAP June2000 Environment ........... 26
    2.2.1 Purpose of This Section .............................................................................. 26
    2.2.2 Overview ..................................................................................................... 27
    2.2.3 Terminal Requirements and Recommendations ........................................ 27
    2.2.4 Requirements and recommendations in addition to those in the WAP Specifications ......................................................... 28
    2.2.5 Gateway Requirements and Recommendations ........................................ 28
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.6</td>
<td>MIDlet/MIDlet Suite Recommendations</td>
<td>28</td>
</tr>
<tr>
<td>2.2.7</td>
<td>References</td>
<td>28</td>
</tr>
<tr>
<td>2.2.8</td>
<td>Terms</td>
<td>28</td>
</tr>
<tr>
<td>3</td>
<td>Security for MIDP Applications</td>
<td>30</td>
</tr>
<tr>
<td>3.1</td>
<td>Untrusted MIDlet Suites</td>
<td>30</td>
</tr>
<tr>
<td>3.2</td>
<td>Trusted MIDlet Suite Security</td>
<td>30</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Definition of Terms</td>
<td>31</td>
</tr>
<tr>
<td>3.3</td>
<td>Authorization Model</td>
<td>31</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Assumptions</td>
<td>31</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Permissions</td>
<td>31</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Permissions for Protected Functions</td>
<td>32</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Requesting Permissions for a MIDlet Suite</td>
<td>32</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Permissions on the Device</td>
<td>32</td>
</tr>
<tr>
<td>3.3.6</td>
<td>Protection Domain</td>
<td>32</td>
</tr>
<tr>
<td>3.3.7</td>
<td>User Permission Interaction Modes</td>
<td>32</td>
</tr>
<tr>
<td>3.3.8</td>
<td>Granting Permissions to Trusted MIDlet Suites</td>
<td>33</td>
</tr>
<tr>
<td>3.3.9</td>
<td>Example External Domain Policy Format</td>
<td>34</td>
</tr>
<tr>
<td>4</td>
<td>Trusted MIDlet Suites using X.509 PKI</td>
<td>36</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>36</td>
</tr>
<tr>
<td>4.1.1</td>
<td>References</td>
<td>36</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Definition of Terms</td>
<td>36</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Signing a MIDlet Suite</td>
<td>36</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Creating the Signing Certificate</td>
<td>37</td>
</tr>
<tr>
<td>4.1.5</td>
<td>Insert Certificates into the application descriptor</td>
<td>37</td>
</tr>
<tr>
<td>4.1.6</td>
<td>Creating the RSA SHA-1 signature of the JAR</td>
<td>37</td>
</tr>
<tr>
<td>4.1.7</td>
<td>Authenticating a MIDlet Suite</td>
<td>38</td>
</tr>
<tr>
<td>4.1.8</td>
<td>Verify Signer Certificate</td>
<td>38</td>
</tr>
<tr>
<td>4.1.9</td>
<td>Verify the MIDlet Suite JAR</td>
<td>38</td>
</tr>
<tr>
<td>4.1.9.1</td>
<td>Summary of MIDlet suite source verification results</td>
<td>39</td>
</tr>
<tr>
<td>4.1.10</td>
<td>Caching of Authentication and Authorization Results</td>
<td>39</td>
</tr>
<tr>
<td>4.1.11</td>
<td>Security in Split-VM Implementations</td>
<td>40</td>
</tr>
<tr>
<td>4.2</td>
<td>MIDP X.509 Certificate Profile for Trusted MIDlet Suites</td>
<td>40</td>
</tr>
</tbody>
</table>
Classes ................................................................. 61
CommConnection ....................................................... 62
Connector ................................................................. 66
HttpConnection ........................................................... 71
HttpsConnection ......................................................... 94
PushRegistry ............................................................. 98
SecureConnection ....................................................... 109
SecurityInfo ............................................................. 112
ServerSocketConnection .............................................. 114
SocketConnection ....................................................... 117
UDPDatagramConnection .............................................. 122

8 javax.microedition.lcdui ........................................ 124
8.1 User Interface ...................................................... 124
8.1.1 Structure of the MIDP UI API ................................. 124
8.1.1.1 Class Hierarchy .............................................. 125
8.1.1.2 Class Overview ............................................. 126
8.1.1.3 Interplay with Application Manager ....................... 126
8.1.2 Event Handling .................................................. 126
8.1.2.1 Abstract Commands ....................................... 128
8.1.2.2 Device-Provided Operations .............................. 128
8.1.2.3 High-Level API for Events ................................. 129
8.1.2.4 Low-Level API for Events ................................. 130
8.1.2.5 Interplay of High-Level Commands and the Low-Level API .......................... 131
8.1.3 Graphics and Text in Low-Level API ....................... 132
8.1.3.1 The Redrawing Scheme ................................... 132
8.1.3.2 Drawing Model ............................................. 132
8.1.3.3 Coordinate System ....................................... 133
8.1.3.4 Font Support .............................................. 133
8.1.4 Concurrency ...................................................... 133
8.1.5 Text Truncation in UI Components ......................... 134
8.1.6 Implementation Notes ......................................... 134
Classes ................................................................. 136
9.1.4   Sprite .......................................................... 345
9.1.5   TiledLayer ..................................................... 345
Classes ................................................................. 347
GameCanvas .......................................................... 348
Layer ................................................................. 355
LayerManager ....................................................... 358
Sprite ................................................................. 363
TiledLayer ............................................................ 379

10  javax.microedition.media ............................................ 386
10.1  Introduction ....................................................... 386
10.2  Background of the Media API ................................. 386
10.2.1   Basic Concepts .............................................. 387
10.2.2   API Details .................................................. 387
10.2.3   Tone Generation ............................................. 388
10.2.4   Usage Scenarios ............................................. 388
  10.2.4.1   Scenario 1: Single-Tone Generation .................. 388
  10.2.4.2   Scenario 2: Simple Media Playback with Looping . 388
  10.2.4.3   Scenario 3: Playing Back from Media Stored in JAR . 388
  10.2.4.4   Scenario 4: Tone Sequence Generation .............. 389
Classes ................................................................. 390
Control ............................................................... 391
Controllable .......................................................... 392
Manager ............................................................... 394
MediaException ...................................................... 399
Player ................................................................. 400
PlayerListener ........................................................ 410

11  javax.microedition.media.control ................................ 414
Classes ................................................................. 415
ToneControl .......................................................... 416
VolumeControl ....................................................... 423

12  javax.microedition.midlet .......................................... 425
12.1  Applications ..................................................... 425
12.1.1 MIDP MIDlet Suite .............................................................. 425
12.2 MIDlet Suite Security ............................................................ 425
12.2.1 MIDP Execution Environment ............................................... 426
12.2.2 MIDlet Suite Packaging ....................................................... 426
12.2.3 MIDlet Attributes .............................................................. 427
   12.2.3.1 Runtime-Execution-Environment ....................................... 429
   12.2.3.2 Version Numbering ...................................................... 429
   12.2.3.3 JAR Manifest ............................................................ 429
   12.2.3.4 MIDlet Classes .......................................................... 430
12.2.4 Application Descriptor ..................................................... 430
12.2.5 Application Lifecycle ....................................................... 433
12.2.6 MIDlet lifecycle ............................................................... 433
12.3 MIDlet Lifecycle Definitions .................................................. 433
12.4 MIDlet States ................................................................. 434
12.5 MIDlet Lifecycle Model ........................................................ 435
12.6 MIDlet Interface ............................................................... 436
12.7 Application Implementation Notes .......................................... 436
12.8 Example MIDlet Application .................................................. 436
Classes ................................................................. 439
   MIDlet ............................................................... 440
   MIDletStateException ........................................................ 446
13 javax.microedition.pki ............................................................ 447
   13.1 MIDP X.509 Certificate Profile ............................................. 447
      13.1.1 Certificate Extensions ................................................. 447
      13.1.2 Certificate Size ....................................................... 448
      13.1.3 Algorithm Support ..................................................... 448
      13.1.4 Certificate Processing for HTTPS .................................. 448
Classes ................................................................. 449
   Certificate ............................................................. 450
   CertificateException ...................................................... 453
14 javax.microedition.rms .......................................................... 458
   14.1 Persistent Storage .......................................................... 458
14.1.1 Record Store ................................................................. 458
14.1.2 Records ................................................................. 459
14.1.3 Example: ................................................................. 459
Classes ................................................................. 464
InvalidRecordIDException ........................................ 465
RecordComparator .................................................. 466
RecordEnumeration ............................................... 468
RecordFilter ............................................................ 472
RecordListener .......................................................... 473
RecordStore ............................................................. 475
RecordStoreException ............................................. 486
RecordStoreFullException ........................................... 488
RecordStoreNotFoundException .................................... 489
RecordStoreNotOpenException ...................................... 490

15 The Recommended Security Policy for GSM/UMTS Compliant Devices 491
15.1 General ................................................................. 492
15.2 Protection Domains ......................................................... 493
15.3 Authorities on Protection Domains ....................................... 493
15.3.1 Manufacturer and Operator Protection Domains ....................... 493
15.3.1.1 Manufacturer Protection Domain .................................. 493
15.3.1.2 Operator Protection Domain ........................................ 494
15.3.2 Third Party Protection Domains ........................................ 497
15.3.2.1 Identified Third Party Protection Domain ......................... 497
15.3.2.2 Unidentified Third Party Protection Domain ...................... 498
15.4 Permissions for Downloaded MIDlet Suites ............................. 499
15.4.1 Mapping MIDP 2.1 Permissions onto Function Groups in Protected Domains 499
15.4.2 Network Access Requirements ...................................... 504
15.4.3 Wireless Messaging Third Party Domain Policy ..................... 504
15.4.4 User Prompting .................................................... 505
15.4.5 Wireless Messaging Interaction Modes ............................... 505
15.4.6 Mobile Media Third Party Domain Policy ............................. 505
15.5 Permissions Granted to a MIDlet Suite by the Authorization Mechanism ........ 506
15.6 Requirements on Restricted APIs ......................................................... 507
15.7 User Prompts and Notifications ............................................................ 508
15.8 MIDlet Download and Execution While Roaming and After Changing the Smart Card 508
15.9 Revocation Checking .......................................................................... 510

**Almanac** .................................................................................................. 512

**Index** .................................................................................................... 544
Preface

These documents define the Final Release of the Mobile Information Device Profile (MIDP) v2.1 for the Java 2 Platform, Micro Edition (J2METM).

This document and all associated documents are subject to the terms of the JCP agreements (i.e. JSPA and/or IEPA).

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-September-2000</td>
<td>MIDP 1.0 Specification</td>
<td>Final MIDP 1.0 specification</td>
</tr>
<tr>
<td>23-October-2001</td>
<td>MIDP 2.0, EG Draft 2</td>
<td>First complete draft published to the Expert Group</td>
</tr>
<tr>
<td>6-November-2001</td>
<td>MIDP 2.0, EG Draft 3</td>
<td>Incorporated changes made during 30-31 October Expert Group meeting</td>
</tr>
<tr>
<td>20-November-2001</td>
<td>MIDP 2.0, EG Draft 4</td>
<td>Incorporated changes discussed on EG mailing lists.</td>
</tr>
<tr>
<td>18-December-2001</td>
<td>MIDP 2.0, EG Draft 5</td>
<td>Incorporated changes made during 5-6 December Expert group meeting and EG mailing lists. Published for Community Review</td>
</tr>
<tr>
<td>12-February-2002</td>
<td>MIDP 2.0, EG Draft 6</td>
<td>Incorporated changes made during Community Review and internal EG mailing lists.</td>
</tr>
<tr>
<td>12-March-2002</td>
<td>MIDP 2.0, EG Draft 7</td>
<td>Incorporated changes made during 20-21 February Expert group meeting and EG mailing lists. Published for Public Review</td>
</tr>
<tr>
<td>09-April-2002</td>
<td>MIDP 2.0, EG Draft 8</td>
<td>Incorporated changes made during 26 March Expert group meeting and EG mailing lists.</td>
</tr>
<tr>
<td>23-April-2002</td>
<td>MIDP 2.0, EG Draft 9</td>
<td>Incorporated changes discussed in the EG mailing lists.</td>
</tr>
<tr>
<td>9-May-2002</td>
<td>MIDP 2.0, EG Draft 10</td>
<td>Incorporated changes discussed in the EG mailing lists.</td>
</tr>
<tr>
<td>29-May-2002</td>
<td>MIDP 2.0, EG Draft 11</td>
<td>Incorporated changes discussed in the EG mailing lists.</td>
</tr>
<tr>
<td>11-June-2002</td>
<td>MIDP 2.0, EG Draft 12</td>
<td>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.</td>
</tr>
<tr>
<td>15-July-2002</td>
<td>MIDP 2.0, EG Draft 13</td>
<td>Incorporated editorial changes and clarifications discussed in the EG mailing lists.</td>
</tr>
</tbody>
</table>
mailing lists. This is final draft candidate 1.

<table>
<thead>
<tr>
<th>Date</th>
<th>Document Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>02-August-2002</td>
<td>MIDP 2.0, EG Draft 14</td>
<td>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.</td>
</tr>
<tr>
<td>04-September-2002</td>
<td>MIDP 2.0, EG Draft 15</td>
<td>Incorporated minor editorial changes and clarifications from the RI and TCK teams and EG.</td>
</tr>
<tr>
<td>27-September-2002</td>
<td>MIDP 2.0, Final Specification</td>
<td>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.</td>
</tr>
<tr>
<td>26-May-2006</td>
<td>MIDP 2.1 ChangeLog</td>
<td>Incorporated all of the Change log items that were accepted during the 2.1 MR. (See table below)</td>
</tr>
</tbody>
</table>

**JSR 118 Maintenance Release 2.1 Change Log**

<table>
<thead>
<tr>
<th>No.</th>
<th>Change</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Make LCDUI layout directives (MIDP 2.0 specification pages 292-294) mandatory</td>
<td>MIDP 2.0 specification Pages 292-294</td>
</tr>
<tr>
<td>3</td>
<td>The behavior of the LayerManager.insert() method MUST comply with the following description:insert(Layer, int)</td>
<td>LayerManager</td>
</tr>
</tbody>
</table>

**Declaration:**

```java
public void insert(javax.microedition.lcdui.game.Layer l, int index)
```

**Description:**

Inserts a new Layer in this LayerManager at the specified index.

Inserting a Layer that was previously added to this LayerManager is equivalent to first removing it with LayerManager.remove() and then adding it with the insert() method at the specified index. Conditions for throwing IndexOutOfBoundsException are checked before LayerManager.remove() is called.

**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
  - If the index is greater than the number of Layers already added to this LayerManager and the Layer has not been added to this LayerManager yet
  - If the index is greater than the number of Layers already added to
4  A StringItem that has item commands and the appearance mode Item.PLAIN MUST always be presented as a StringItem with added command(s) and appearance mode Item.HYPERLINK.

5  If the device keypad can be used in different modes, and some modes restrict user access to certain keys, or if the device has several different keypads that are not meant to be used simultaneously, a mobile handset implementation is NOT REQUIRED to ensure that the mapping between key codes and game action does not change during the execution of the application.

   Consequently, applications SHOULD NOT use the Canvas.getKeyCode() method to learn which key is assigned to a certain game action. Instead, applications SHOULD use the Canvas.getGameAction() method to check which game action is associated with a given key code.

6  Many MIDP LCDUI graphical components can contain text (that is, an alphanumeric string) that is shown to the user. Examples of such components are List, TextBox, Alert, StringItem, Form, and Item. An implementation often needs to truncate such visible text because it does not fit in the designated space of a given UI component. In this case, an implementation MUST use an appropriate visual indication (for example an ellipsis symbol) to signal the user that the text is truncated. The actual symbol or symbols used to represent the truncated text depends on the locale that is currently selected in the device. However, the visual indication SHOULD be consistent with the visual indication used in the device's native UI.

7  Canvas pointer events MUST be supported, if the underlying hardware supports this feature. In this case, the method Canvas.hasPointerEvents() SHALL always return true.

8  Canvas pointer motion events MUST be supported, if the underlying hardware supports this feature. In this case the method Canvas.hasPointerMotionEvents() SHALL always return true.

9  Canvas repeat events MUST be supported. The method Canvas.hasRepeatEvents() SHALL always return true.

10 Double-buffered graphics MUST be supported. The method Canvas.isDoubleBuffered() SHALL always return true.

11 A device can have a 3-way or 5-way jog dial as a control mechanism. A 3-way jog dial is usually a wheel that rotates in two directions (to indicate scrolling) and can also be pressed (to indicate a selection). A 5-way jog dial is typically similar to a 3-way jog dial with the added possibility to tilt the wheel sideways.

   A jog dial wheel might have the ability to be rolled several steps in each direction. Alternatively, a jog dial wheel might only have the ability to be rotated by a limited angle, returning to the base position when released.
are as follows:

- An implementation **MUST** generate key press events in response to jog dial movements. An implementation **MAY** reuse key codes assigned to device keys with functions similar to jog dial functions (for example, key codes assigned to cursor keys can also be used for jog dial events). However, an implementation **MAY** choose to use unique key codes for jog dial events.
- An implementation **MUST** map key press events generated by jog dial movements to game actions. An application can learn game actions via the `Canvas.getGameAction()` method. The mapping of game actions **MUST** always be natural in relation to the spatial orientation of the device (that is, mapping **MUST** be natural from the user point of view. For example, if the jog dial is rolled upwards, UP events are generated). This also means that an implementation **MAY** need to deliver different key events and game actions depending on the spatial orientation of the device. For example, an application running in a mode that requires a certain orientation of the device can get different key codes than an application running in another orientation. In the same vein, a clamshell device **MAY** generate different jog dial events when the clamshell is in OPEN or CLOSED mode.
- A 3-way jog dial **MUST** generate key events with key codes mapping to game actions UP, DOWN and FIRE.
- A 5-way jog dial **MUST** generate key events with key codes mapping to game actions UP, DOWN, LEFT, RIGHT and FIRE.
- A jog dial **MAY** also be used to control high level UI objects.

Note: Depending on the mechanics of the jog wheel, the implementation is not necessarily able to generate key repeat events for some movements (for example, when the wheel is rotated).

12 If Canvas is in full-screen mode, the following requirements apply:

- If CommandListener is set for a Canvas, an implementation **MUST** display the commands in the User Interface when a key normally reserved for the delivery of commands is pressed and deliver the selected command.
- If no CommandListener is set for a Canvas, an implementation **MUST** deliver key code events for key presses that are otherwise reserved for the delivery of commands.

13 Availability of various text input modes (for example, predictive input and numbers-only input) **SHOULD** be consistent across Java and native applications. This means, for example, that if predictive text input mode is available in native applications, it **SHOULD** also be available in Java applications.

15 Correct the first example on page 57 of the MIDP 2.0 specification that contains an error (com0 in the first line **MUST** be COM1). Here is the corrected version of the code:

```java
CommConnection cc = (CommConnection) Connector.open("comm:COM1;baudrate=19200");
int baudrate = cc.getBaudRate();
```
```java
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();
```

Also, "Recommended Port Naming Convention" on pages 57-58 is replaced with the following requirements:

**Port Naming Requirements:**

Ports have to be named consistently among the implementations of this class. Therefore, VM implementations MUST follow the following convention:

Port names MUST contain a text abbreviation indicating port capabilities followed by a sequential number for the port. Port numbering MUST start from 1 without any leading zeros, and numbering MUST be sequential (without any gaps).

The following device name types MUST 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
- USB#, where USB is for USB ports and # is a number assigned to the port
- BT#, where BT is for Bluetooth Serial Port Profile ports and # is a number assigned to the port.

This naming scheme allows API users to generally determine the type of port to use. For example, if an application needs to beam a piece of data, the application could look for IR# ports for opening the connection. The alternative is a trial-and-error approach with all available ports.

If the implementation cannot determine a port type, the same naming convention as for an RS-232 port MUST be used.

| 16 | Every serial port name included in the String value returned by the method System.getProperty("microedition.comports") MUST be accessible via the javax.microedition.io.CommConnection interface. |
| 17 | In every protocol, the AllowedSender field MUST match with the appropriate address field of the incoming event. The address field to use and the exact syntax and semantics of the address depend on the protocol. However, the address and the filter MUST be compared by exact string matching where the strings are compared character by character and the characters need to match exactly except as allowed by the two wildcard characters: asterisk (*) and question mark (?). The semantics of these wildcard characters are defined in the MIDP 2.0 specification. |
For `javax.microedition.io.HttpsConnection` and `javax.microedition.io.SecureConnection` interfaces the SSL V3 protocol MUST be supported as the secure protocol. Other secure protocols, such as TLS and WTLS, MAY also be supported.

The following diagram illustrates the requirements related to the MIDlet lifecycle, as defined in the MIDP Specification:

The MIDP specification requires that the application descriptor contain the following attributes:

- MIDlet-Name
- MIDlet-Version
- MIDlet-Vendor
- MIDlet-Jar-URL
- MIDlet-Jar-Size

In addition, to improve the user experience (by rejecting incompatible suites without needing to download the JAR file), the application descriptor SHOULD contain the following attributes:

- MicroEdition-Profile
- MicroEdition-Configuration
- MIDlet-Permissions

The following JAD / JAR manifest attribute is defined to support specifying the expected runtime execution environment:

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Attribute description</th>
</tr>
</thead>
</table>

Page 436

Page 431-435
<table>
<thead>
<tr>
<th><strong>Runtime-Execution-Environment</strong></th>
<th>This is an optional attribute that indicates the runtime execution environment REQUIRED by the application. This attribute MAY have the value MIDP.CLDC. If the MIDlet suite does not specify the attribute, the implicit default value is MIDP.CLDC. When the value of the Runtime-Execution-Environment attribute is MIDP.CLDC, the behavior of the implementation MUST adhere to the requirements presented in the following more detailed description. Additional values for this attribute MAY be defined in the future.</th>
</tr>
</thead>
</table>

A mobile handset implementation MUST support the Runtime-Execution-Environment attribute.

**Runtime-Execution-Environment**

- When an application defines the Runtime-Execution-Environment attribute value MIDP.CLDC or does not define this attribute, the behavior of the mobile handset implementation during execution of this application MUST adhere to the following requirements:
- The set of supported APIs and the API behavior, as well as the behavior of the underlying virtual machine, MUST comply with the CLDC 1.1 specification.
- A mobile handset implementation MAY alternatively use the CDC configuration of Java ME. However, if CDC is used as the underlying configuration, applications running on top of this platform MUST see an environment that is semantically and functionally equivalent to a CLDC 1.1 platform. CDC-specific APIs or CDC-specific behavior MUST NOT be exposed to applications or application developers.
- When an application defines the Runtime-Execution-Environment attribute value MIDP.CLDC the application MUST also define a CLDC platform in the MicroEdition-Configuration attribute.
- If an application defines a value for the Runtime-Execution-Environment or MicroEdition-Configuration attribute that is not supported by the implementation, the application MUST NOT be installed. All mobile handset implementations MUST support the "MIDP.CLDC" value for the Runtime-Execution-Environment attribute.

23 When an uncaught exception occurs in a thread, the thread MUST be stopped in accordance with the Java Language Specification. In addition, further handling of the uncaught exception MAY occur as follows:

- If an implementation can identify the MIDlet that created the thread (the MIDlet to which the thread belongs), the

<p>| MIDlet | MIDletStateChangeException |</p>
<table>
<thead>
<tr>
<th>Page 12-13</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDlet.destroyApp(false) method of this MIDlet MAY be called.</td>
</tr>
<tr>
<td>• If an implementation can only identify the MIDlet suite that created the thread (the MIDlet suite to which the thread belongs), the MIDlet.destroyApp(false) methods of all MIDlets from this suite MAY be called.</td>
</tr>
<tr>
<td>• If an implementation cannot identify either a MIDlet or a MIDlet suite that owns the offending thread, MIDlet.destroyApp(false) methods of all MIDlets executing in this VM instance MAY be called.</td>
</tr>
<tr>
<td>• In all of these cases, the behavior after a call to MIDlet.destroyApp(false) is as follows:</td>
</tr>
<tr>
<td>• If a MIDlet attempts to recover by throwing javax.microedition.midlet.MIDletStateChangeException, the implementation MAY allow the MIDlet to continue running.</td>
</tr>
<tr>
<td>• If a call to MIDlet.destroyApp(false) results in a different exception of any kind, the implementation MUST terminate the MIDlet.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page 433</th>
</tr>
</thead>
<tbody>
<tr>
<td>24 In case of user-initiated OTA provisioning, after a MIDlet suite is successfully installed on the device and if the suite contains any MIDlets (that is, one or more MIDlet-&lt;n&gt; attributes are in the JAD file or JAR manifest), an implementation MUST prompt the user to specify whether to launch a MIDlet from the suite.</td>
</tr>
<tr>
<td>If the user approves launching a MIDlet and the MIDlet suite contains only one MIDlet, that specific MIDlet MUST be launched. If the suite contains several MIDlets, an implementation MUST give the user the opportunity to choose which MIDlet to launch. After the selection, the chosen MIDlet MUST be launched.</td>
</tr>
<tr>
<td>These requirements apply only to MIDlets that are listed in the JAD file or in the JAR manifest by means of MIDlet-&lt;n&gt; attributes.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>25 If an implementation displays MIDlet icons (using associated MIDlet-Icon attribute), the following requirements apply:</td>
</tr>
<tr>
<td>• The application manager MUST always be able to display a MIDlet-Icon of 16 by 16 pixels in the application list.</td>
</tr>
<tr>
<td>• The application manager SHOULD always be able to display a MIDlet-Icon of 24 by 24 pixels in the application list.</td>
</tr>
<tr>
<td>• A MIDlet-Icon larger than 24 by 24 MAY be displayed properly in full size in the application list.</td>
</tr>
<tr>
<td>• A MIDlet-Icon MUST NOT be scaled down in the application list, but the application manager MAY use centered cropping to fit the icon into the list.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>26 If a MIDlet suite contains multiple MIDlets, an implementation MUST make the user aware of the MIDlets and associated RMS record stores that will be removed, and the user MUST be given the chance to abort this action.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>27 In MIDlet suite installations and deletions, if the status report cannot be sent, or if the server reply is not received, an implementation MUST attempt to resend the report at least 5 times. Attempts to resend the status report MUST NOT be performed after a response from the server is received.</td>
</tr>
</tbody>
</table>
The precise conditions for resending the status report (both for installations and deletions) are defined in the MIDP specification.

35 The MIDP subset of MMAPI (classes from the javax.microedition.media package) MUST comply with the MMAPI specification version 1.1 (or later).

36 The microedition.profiles system property MUST NOT contain different versions of the same profile. In particular, MIDP 1.0 and MIDP 2.0 MUST NOT both be listed.

38 Chapter 1, Section PNG Image Format and Chapter 8.

class javax.microedition.lcdui.Image:

- public static Image createImage(byte[] imageData, int imageOffset, int imageLength)
- public static Image createImage(Image image, int x, int y, int width, int height, int transform)
- public static Image createImage(java.io.InputStream stream)
- public static Image createImage(String name)

ISO/IEC JPEG together with JFIF must be supported by a compliant device. The support for ISO/IEC JPEG only applies to baseline DCT, non-differential, Huffman coding, as defined in table B.1, symbol 'SOF0' in [1]. This support extends to the class javax.microedition.lcdui.Image, including the methods outlined

ITU-T Recommendation T.81: "Information technology; Digital compression and coding of continuous-tone still images: Requirements and guidelines" 09/92.

http://www.w3.org/Graphics/JPEG/itu-t81

42 Chapter 8.

- class javax.microedition.lcdui.TextBox
- class javax.microedition.lcdui.TextField

**Requirement Text**

TextBox and TextField with input constraint TextField.ANY must support input of the following set of characters:

<table>
<thead>
<tr>
<th>Glyph</th>
<th>Unicode Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>U+00A</td>
<td>line feed</td>
</tr>
<tr>
<td></td>
<td>U+0020</td>
<td>space</td>
</tr>
<tr>
<td>!</td>
<td>U+0021</td>
<td>exclamation mark</td>
</tr>
<tr>
<td>&quot;</td>
<td>U+0022</td>
<td>quotation mark</td>
</tr>
<tr>
<td>#</td>
<td>U+0023</td>
<td>number sign</td>
</tr>
<tr>
<td>$</td>
<td>U+0024</td>
<td>dollar sign</td>
</tr>
<tr>
<td>%</td>
<td>U+0025</td>
<td>percent</td>
</tr>
</tbody>
</table>

Page 35

Chapter 1 and 8

Chapter 8
<table>
<thead>
<tr>
<th>Character</th>
<th>Unicode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>U+0026</td>
<td>ampersand</td>
</tr>
<tr>
<td>`</td>
<td>U+0027</td>
<td>apostrophe</td>
</tr>
<tr>
<td>(</td>
<td>U+0028</td>
<td>left parenthesis</td>
</tr>
<tr>
<td>)</td>
<td>U+0029</td>
<td>right parenthesis</td>
</tr>
<tr>
<td>*</td>
<td>U+002A</td>
<td>asterisk</td>
</tr>
<tr>
<td>+</td>
<td>U+002B</td>
<td>plus</td>
</tr>
<tr>
<td>,</td>
<td>U+002C</td>
<td>comma</td>
</tr>
<tr>
<td>-</td>
<td>U+002D</td>
<td>hyphen-minus</td>
</tr>
<tr>
<td>.</td>
<td>U+002E</td>
<td>period</td>
</tr>
<tr>
<td>/</td>
<td>U+002F</td>
<td>slash</td>
</tr>
<tr>
<td>0-9</td>
<td>U+0030 - U+0039</td>
<td>digit 0 through 9</td>
</tr>
<tr>
<td>:</td>
<td>U+003A</td>
<td>colon</td>
</tr>
<tr>
<td>;</td>
<td>U+003B</td>
<td>semicolon</td>
</tr>
<tr>
<td>&lt;</td>
<td>U+003C</td>
<td>less-than sign</td>
</tr>
<tr>
<td>=</td>
<td>U+003D</td>
<td>equals sign</td>
</tr>
<tr>
<td>&gt;</td>
<td>U+003E</td>
<td>greater-than sign</td>
</tr>
<tr>
<td>?</td>
<td>U+003F</td>
<td>question mark</td>
</tr>
<tr>
<td>@</td>
<td>U+0040</td>
<td>commercial at</td>
</tr>
<tr>
<td>A - Z</td>
<td>U+0041 - U+005A</td>
<td>Latin capital letter A through Z</td>
</tr>
<tr>
<td>[</td>
<td>U+005B</td>
<td>left square bracket</td>
</tr>
<tr>
<td>\</td>
<td>U+005C</td>
<td>backslash</td>
</tr>
<tr>
<td>]</td>
<td>U+005D</td>
<td>right square bracket</td>
</tr>
<tr>
<td>^</td>
<td>U+005E</td>
<td>circumflex accent</td>
</tr>
<tr>
<td>_</td>
<td>U+005F</td>
<td>spacing underscore</td>
</tr>
<tr>
<td>`</td>
<td>U+0060</td>
<td>grave accent</td>
</tr>
<tr>
<td>a - z</td>
<td>U+0061 - U+007A</td>
<td>Latin small letter a through z</td>
</tr>
<tr>
<td>{</td>
<td>U+007B</td>
<td>left curly bracket</td>
</tr>
<tr>
<td></td>
<td></td>
<td>U+007C</td>
</tr>
<tr>
<td>}</td>
<td>U+007D</td>
<td>right curly bracket</td>
</tr>
<tr>
<td>~</td>
<td>U+007E</td>
<td>tilde</td>
</tr>
<tr>
<td>i</td>
<td>U+00A1</td>
<td>inverted exclamation</td>
</tr>
<tr>
<td>£</td>
<td>U+00A3</td>
<td>pound sign</td>
</tr>
<tr>
<td>¢</td>
<td>U+00A4</td>
<td>currency sign</td>
</tr>
<tr>
<td>¥</td>
<td>U+00A5</td>
<td>yen sign</td>
</tr>
<tr>
<td>§</td>
<td>U+00A7</td>
<td>section mark</td>
</tr>
<tr>
<td>¤</td>
<td>U+00BF</td>
<td>inverted question mark</td>
</tr>
</tbody>
</table>
### Chapter 8

- `class javax.microedition.lcdui.TextBox`
- `class javax.microedition.lcdui.TextField`

**Requirement Text**

Instances of these classes with either of the constraints `TextField.EMAILADDR` and `TextField.URL` must allow the same characters to be input as are allowed for input constraint `TextField.ANY`.

<table>
<thead>
<tr>
<th>43</th>
<th>Chapter 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>Recommended Security Policy - corrected typos and minor edits found since 2.0.1</td>
</tr>
</tbody>
</table>

**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**

- Connected, Limited Device Configuration (JSR-30), Sun Microsystems, Inc.
- Mobile, Information Device Profile (JSR-37), Sun Microsystems, Inc.
- Connected, Limited Device Configuration 1.1 (JSR-139), Sun Microsystems, Inc.

**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**.
Specification Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUST</td>
<td>The associated definition is an absolute requirement of this specification.</td>
</tr>
<tr>
<td>MUST NOT</td>
<td>The definition is an absolute prohibition of this specification.</td>
</tr>
<tr>
<td>SHOULD</td>
<td>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.</td>
</tr>
<tr>
<td>SHOULD NOT</td>
<td>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.</td>
</tr>
<tr>
<td>MAY</td>
<td>Indicates that an item is truly optional.</td>
</tr>
</tbody>
</table>

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
1.1 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), 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), and presumably any newer versions. It is anticipated that most MIDP 2.0 implementations will be based on CLDC 1.1.

1.2 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.

1.3 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

<table>
<thead>
<tr>
<th>Application Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP</td>
<td>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.</td>
</tr>
<tr>
<td>OEM-Specific</td>
<td>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.</td>
</tr>
<tr>
<td>Native</td>
<td>A native application is one that is not written in Java and is built on top of the MID's existing, native system software.</td>
</tr>
</tbody>
</table>

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

## 1.4 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,
1.4.1 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).

1.4.2 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.

1.5 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.
- MUST support ISO/IEC JPEG together with JFIF.
- 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 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

# Package Summary

## User Interface Package

**javax.microedition.lcdui**  
The UI API provides a set of features for implementation of user interfaces for MIDP applications.

## Game Package

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

## Networking Package

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

## Public Key Package

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

## Sound

**javax.microedition.media**  
The MIDP 2.1 Media API is a directly compatible building block of the Mobile Media API (JSR-135) specification.

**javax.microedition.media.control**  
This package defines the specific *Control* types that can be used with a *Player*.

## Persistence Package

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

## Core Packages

**java.lang**  
MID Profile Language Classes included from Java 2 Standard Edition.

**java.util**  
MID Profile Utility Classes included from Java 2 Standard Edition.
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. 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:

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

References

2. Mobile Information Device Profile (MIDP 1.0) http://jcp.org/jsr/detail/37.jsp
3. Mobile Information Device Profile 2.0 (MIDP 2.0) http://jcp.org/jsr/detail/118.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.

2.1 Over The Air User Initiated Provisioning

2.1.1 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. BluetoothTM wireless technology, serial cable, IrDATM, 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.

2.1.2 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.

2.1.3 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.

2.1.4 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.

In case of user-initiated OTA provisioning, after a MIDlet suite is successfully installed on the device and if the suite contains any MIDlets (that is, one or more MIDlet-<n> attributes are in the JAD file or JAR manifest), an implementation MUST prompt the user to specify whether to launch a MIDlet from the suite. If the user approves launching a MIDlet and the MIDlet suite contains only one MIDlet, that specific MIDlet MUST be launched. If the suite contains several MIDlets, an implementation MUST give the user the opportunity to choose which MIDlet to launch. After the selection, the chosen MIDlet MUST be launched. These requirements apply only to MIDlets that are listed in the JAD file or in the JAR manifest by means of MIDlet-<n> attributes.
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:
   4. If there is insufficient memory to store the MIDlet suite on the device, the device MUST return Status Code 901 in the Status Report.
   5. 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.
   6. 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.
   7. If the manifest or any other file cannot be extracted from the JAR, the device MUST return Status Code 907 in the Status Report.
   8. If the 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.
   9. 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.
   10. 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.
   11. If the application failed to be authenticated, the device MUST return Status Code 909 in the Status Report.
   12. 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.
   13. 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.
   14. 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.
   15. 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).

5. 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.

6. 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.

2.1.5 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.

2.1.6 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.

2.1.7 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. If a MIDlet suite contains multiple MIDlets, an implementation MUST make the user aware of the MIDlets and associated RMS record stores that will be removed, and the user MUST be given the chance to abort this action.

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.

2.1.8 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, an implementation MUST attempt to resend the report at least 5 times. Attempts to resend the status report MUST NOT be performed after a response from the server is received. 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, an implementation MUST attempt to resend the report at least 5 times. Attempts to resend the status report MUST NOT be performed after a response from the server is received. 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

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Status Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>900</td>
<td>Success</td>
</tr>
<tr>
<td>901</td>
<td>Insufficient Memory</td>
</tr>
<tr>
<td>902</td>
<td>User Cancelled</td>
</tr>
<tr>
<td>903</td>
<td>Loss of Service</td>
</tr>
<tr>
<td>904</td>
<td>JAR size mismatch</td>
</tr>
<tr>
<td>905</td>
<td>Attribute Mismatch</td>
</tr>
<tr>
<td>906</td>
<td>Invalid Descriptor</td>
</tr>
<tr>
<td>907</td>
<td>Invalid JAR</td>
</tr>
<tr>
<td>908</td>
<td>Incompatible Configuration or Profile</td>
</tr>
<tr>
<td>909</td>
<td>Application authentication failure</td>
</tr>
<tr>
<td>910</td>
<td>Application authorization failure</td>
</tr>
<tr>
<td>911</td>
<td>Push registration failure</td>
</tr>
<tr>
<td>912</td>
<td>Deletion Notification</td>
</tr>
</tbody>
</table>

#### 2.1.9 Additional Descriptor Attributes

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

MIDlet Attributes
<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDlet-Install-Notify</td>
<td>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.</td>
</tr>
<tr>
<td>MIDlet-Delete-Notify</td>
<td>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.</td>
</tr>
<tr>
<td>MIDlet-Delete-Confirm</td>
<td>A text message to be provided to the user when prompted to confirm deletion of this MIDlet suite</td>
</tr>
</tbody>
</table>

### 2.1.10 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.

#### 2.1.10.1 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.

#### 2.1.10.2 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.

#### 2.1.10.3 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.

#### 2.1.10.4 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
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

**2.1.10.5 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

**2.1.10.6 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

2.2 MIDP Provisioning and Networking in the WAP June2000 Environment

**2.2.1 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.

### 2.2.2 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.

### 2.2.3 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.

### 2.2.4 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).

### 2.2.5 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.

### 2.2.6 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.

### 2.2.7 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
5. WAP_WDPS

### 2.2.8 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.

3.1 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:

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.microedition.rms</td>
<td>RMS APIs</td>
</tr>
<tr>
<td>javax.microedition.midlet</td>
<td>MIDlet Lifecycle APIs</td>
</tr>
<tr>
<td>javax.microedition.lcdui</td>
<td>User Interface APIs</td>
</tr>
<tr>
<td>javax.microedition.lcdui.game</td>
<td>The Game APIs</td>
</tr>
<tr>
<td>javax.microedition.media</td>
<td>The multi-media APIs for playback of sound</td>
</tr>
<tr>
<td>javax.microedition.media.control</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>API</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.microedition.io.HttpConnection</td>
<td>http</td>
</tr>
<tr>
<td>javax.microedition.io.HttpsConnection</td>
<td>https</td>
</tr>
</tbody>
</table>

3.2 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.

3.2.1 Definition of Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protection Domain</td>
<td>A set of Allowed and User permissions that may be granted to a MIDlet suite</td>
</tr>
<tr>
<td>Permission</td>
<td>A named permission defined by an API or function to prevent it from being used without authorization</td>
</tr>
<tr>
<td>Trusted MIDlet Suite</td>
<td>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</td>
</tr>
</tbody>
</table>

3.3 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

3.3.1 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.

3.3.2 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.

3.3.3 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.

3.3.4 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.

3.3.5 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.

3.3.6 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.

3.3.7 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.

3.3.8 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 then they are removed from the requested permissions.
- 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.
3.3.9 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 = permission_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_permission_levels = highest_level ["(" default_level ")"]
   highest_level = user_permission_level
   default_level = user_permission_level ; cannot be greater the highest_level
   user_permission_level = blanket | session | oneshot
   blanket = "allow" ; allow access without asking the user.
   session = "session" ; Allow access, ask again at next MIDlet suite startup.
   oneshot = "oneshot" ; Allow access, ask again at next use.
   alias_def = "alias:" *WS alias_name *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.UDPDataagramConnection
oneshot(oneshot): javax.microedition.io.CommConnection

domain: allnet
blanket(session): client_connections
oneshot: javax.microedition.io.CommConnection
Trusted MIDlet Suites using X.509 PKI

4.1 Introduction

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.

4.1.1 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
- [RFC2459] - Internet X.509 Public Key Infrastructure

4.1.2 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:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protection Domain</td>
<td>A certificate associated with a protection domain that the device implicitly trusts to verify and authorize downloaded MIDlet suites</td>
</tr>
<tr>
<td>Root Certificate</td>
<td></td>
</tr>
</tbody>
</table>

4.1.3 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.

4.1.4 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.

4.1.5 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.

4.1.6 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.
4.1.7 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.

4.1.8 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.

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

4.1.9 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.

4.1.9.1 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

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

4.1.10 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.
4.1.11 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.

4.2 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.

4.2.1 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.

4.2.2 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.

4.2.3 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.
**4.2.3.1 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

**4.2.3.2 Example - Protection Domain Stakeholder Owns Signing Certificate**

This encodes the signer's 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.IllegalArgumentException

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

5.1 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.

5.2 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:

5.2.1 System Properties Defined by MIDP

<table>
<thead>
<tr>
<th>System Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>microedition.locale</td>
<td>The current locale of the device, may be null</td>
</tr>
</tbody>
</table>
| microedition.profiles    | is a blank (Unicode U+0020) separated list of the J2ME profiles that this device supports. For MIDP 2.1 devices, this property MUST contain at least "MIDP-2.1"

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 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). )

Property microedition.profiles

The microedition.profiles system property MUST NOT contain different versions of the same profile. In particular, MIDP1.0 and MIDP2.x MUST NOT both be listed.

The language codes MUST be the lower-case, two-letter codes as defined by ISO-639.

The country code MUST be the upper-case, two-letter codes as defined by ISO-3166.

5.3 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 occurrences of "/./" are replaced with "/".
- All occurrences 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.

**5.4 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`.

**5.5 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
<table>
<thead>
<tr>
<th>Class Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Exceptions</strong></td>
</tr>
<tr>
<td><strong>IllegalStateException</strong></td>
</tr>
</tbody>
</table>
IllegalStateException

Declaration

public class IllegalStateException extends RuntimeException

Object

+-Throwable
  +--Exception
    +--RuntimeException
      +--java.lang.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

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IllegalStateException()</td>
<td>Constructs an IllegalStateException with no detail message.</td>
</tr>
<tr>
<td>public IllegalStateException(String s)</td>
<td>Constructs an IllegalStateException with the specified detail message. A detail message is a String that describes this particular exception.</td>
</tr>
</tbody>
</table>

Methods inherited from class Throwable

getMessage, printStackTrace, toString

Methods inherited from class Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

IllegalStateException

public IllegalStateException()

Constructs an IllegalStateException with no detail message.

IllegalStateException

public IllegalStateException(String s)

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
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 tasks 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

<table>
<thead>
<tr>
<th>Classes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Timer</strong></td>
<td>A facility for threads to schedule tasks for future execution in a background thread.</td>
</tr>
<tr>
<td><strong>TimerTask</strong></td>
<td>A task that can be scheduled for one-time or repeated execution by a Timer.</td>
</tr>
</tbody>
</table>
java.util

Timer

Declaration

public class Timer

Object

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, Object.wait(long)

Constructor Summary

<table>
<thead>
<tr>
<th>public Timer()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creates a new timer.</td>
</tr>
</tbody>
</table>

Method Summary

| void cancel() |
| Terminates this timer, discarding any currently scheduled tasks. |

| void schedule(TimerTask task, Date time) |
| Schedules the specified task for execution at the specified time. |

| void schedule(TimerTask task, Date firstTime, long period) |
| Schedules the specified task for repeated fixed-delay execution, beginning at the specified time. |
void schedule(TimerTask task, long delay)
Schedules the specified task for execution after the specified delay.

void schedule(TimerTask task, long delay, long period)
Schedules the specified task for repeated fixed-delay execution, beginning after the specified delay.

void scheduleAtFixedRate(TimerTask task, Date firstTime, long period)
Schedules the specified task for repeated fixed-rate execution, beginning at the specified time.

void scheduleAtFixedRate(TimerTask task, long delay, long period)
Schedules the specified task for repeated fixed-rate execution, beginning after the specified delay.

Methods inherited from class Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

Timer
public Timer()

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

See Also: java.lang.Thread, Timer.cancel()

Methods

cancel
public void cancel()

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
public void schedule(TimerTask task, Date time)

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 - if task was already scheduled or cancelled, timer was cancelled, or timer thread terminated.

**schedule**

```java
public void schedule(TimerTask task,
                    Date firstTime,
                    long period)
```

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` - if task was already scheduled or cancelled, timer was cancelled, or timer thread terminated.

**schedule**

```java
public void schedule(TimerTask task,
                    long delay)
```

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` - if task was already scheduled or cancelled, or timer was cancelled.
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 - if task was already scheduled or cancelled, timer was cancelled, or timer thread terminated.

---

`scheduleAtFixedRate`

```java
public void scheduleAtFixedRate(TimerTask task,
                                Date firstTime,
                                long period)
```

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 - if task was already scheduled or cancelled, timer was cancelled, or timer thread terminated.

---

`scheduleAtFixedRate`

```java
public void scheduleAtFixedRate(TimerTask task,
                                long delay,
                                long period)
```

-51-
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 - if task was already scheduled or cancelled, timer was cancelled, or timer thread terminated.
TimerTask

Declaration
public abstract class TimerTask implements Runnable

Description
A task that can be scheduled for one-time or repeated execution by a Timer.
Since: MIDP 1.0
See Also: Timer

Constructor Summary

<table>
<thead>
<tr>
<th>protected</th>
<th>TimerTask()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Creates a new timer task.</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>boolean</th>
<th>cancel()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cancels this timer task.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>abstract void</th>
<th>run()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The action to be performed by this timer task.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>long</th>
<th>scheduledExecutionTime()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the scheduled execution time of the most recent actual execution of this task.</td>
</tr>
</tbody>
</table>

Methods inherited from class Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from interface Runnable
run

Constructors

TimerTask

<table>
<thead>
<tr>
<th>protected</th>
<th>TimerTask()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Creates a new timer task.</td>
</tr>
</tbody>
</table>
Methods

cancel

public boolean cancel()

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

public abstract void run()

The action to be performed by this timer task.

scheduledExecutionTime

public long scheduledExecutionTime()

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:

```java
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.

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
javafx.microedition.io

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

7.1 HTTP Networking

In addition to the javafx.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.

- javafx.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 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.

7.1.1 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.)

**7.1.2 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.

7.1.3 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.

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

<table>
<thead>
<tr>
<th>System Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>microedition.profiles</td>
<td>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 &quot;MIDP-2.0&quot;.</td>
</tr>
<tr>
<td>microedition.configuration</td>
<td>The J2ME configuration supported by this device. For example, &quot;CLDC-1.0.&quot;</td>
</tr>
<tr>
<td>microedition.locale</td>
<td>The name of the current locale on this device. For example, &quot;en-US.&quot;</td>
</tr>
</tbody>
</table>

7.1.4.1 HTTP Request Header Example

User-Agent: Profile/MIDP-2.0 Configuration/CLDC-1.0
Accept-Language: en-US

7.2 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 executing 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.

7.3 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

7.4 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 address (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 UDPPacketConnection 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.UDPPacketConnection

7.5 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

7.6 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**

### 7.7 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 to 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.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>javax.microedition.io.Connector.http</code></td>
<td>http</td>
</tr>
<tr>
<td><code>javax.microedition.io.Connector.https</code></td>
<td>https</td>
</tr>
<tr>
<td><code>javax.microedition.io.Connector.datagram</code></td>
<td>datagram</td>
</tr>
<tr>
<td><code>javax.microedition.io.Connector.datagramreceiver</code></td>
<td>datagram server (without host)</td>
</tr>
<tr>
<td><code>javax.microedition.io.Connector.socket</code></td>
<td>socket</td>
</tr>
<tr>
<td><code>javax.microedition.io.Connector.serversocket</code></td>
<td>server socket (without host)</td>
</tr>
<tr>
<td><code>javax.microedition.io.Connector.ssl</code></td>
<td>ssl</td>
</tr>
<tr>
<td><code>javax.microedition.io.Connector.comm</code></td>
<td>comm</td>
</tr>
</tbody>
</table>

### 7.8 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.serversocket
```

Since: MIDP 1.0
## Class Summary

### Interfaces

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CommConnection</td>
<td>This interface defines a logical serial port connection.</td>
</tr>
<tr>
<td>HttpConnection</td>
<td>This interface defines the necessary methods and constants for an HTTP connection.</td>
</tr>
<tr>
<td>HttpsConnection</td>
<td>This interface defines the necessary methods and constants to establish a secure network connection.</td>
</tr>
<tr>
<td>SecureConnection</td>
<td>This interface defines the secure socket stream connection.</td>
</tr>
<tr>
<td>SecurityInfo</td>
<td>This interface defines methods to access information about a secure network connection.</td>
</tr>
<tr>
<td>ServerSocketConnection</td>
<td>This interface defines the server socket stream connection.</td>
</tr>
<tr>
<td>SocketConnection</td>
<td>This interface defines the socket stream connection.</td>
</tr>
<tr>
<td>UDPDatagramConnection</td>
<td>This interface defines a datagram connection which knows it's local end point address.</td>
</tr>
</tbody>
</table>

### Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connector</td>
<td>Factory class for creating new Connection objects.</td>
</tr>
<tr>
<td>PushRegistry</td>
<td>The PushRegistry maintains a list of inbound connections.</td>
</tr>
</tbody>
</table>
javax.microedition.io

CommConnection

Declaration

public interface CommConnection extends 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.)

Every serial port name included in the String value returned by the method System.getProperty("microedition.commports") MUST be accessible via the javax.microedition.io.CommConnection interface.

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baudrate</td>
<td>platform dependent</td>
<td>The speed of the port.</td>
</tr>
<tr>
<td>bitsperchar</td>
<td>8</td>
<td>The number bits per character(7 or 8).</td>
</tr>
<tr>
<td>stopbits</td>
<td>1</td>
<td>The number of stop bits per char(1 or 2)</td>
</tr>
<tr>
<td>parity</td>
<td>none</td>
<td>The parity can be odd, even, or none.</td>
</tr>
<tr>
<td>blocking</td>
<td>on</td>
<td>If on, wait for a full buffer when reading.</td>
</tr>
<tr>
<td>autocts</td>
<td>on</td>
<td>If on, wait for the CTS line to be on before writing.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autorts</td>
<td>on</td>
<td>If on, turn on the RTS line when the input buffer is not full. If off, the RTS line is always on.</td>
</tr>
</tbody>
</table>

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

```java
CommConnection cc = (CommConnection) Connector.open("comm:COM1;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();
```

```java
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;
}
```

**Port Naming Requirements:**

Ports have to be named consistently among the implementations of this class. Therefore, VM implementations MUST follow the following convention: Port names MUST contain a text abbreviation indicating port capabilities followed by a sequential number for the port. Port numbering MUST start from 1 without any leading zeros, and numbering MUST be sequential (without any gaps). The following device name types MUST 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
- USB#, where USB is for USB ports and # is a number assigned to the port
- BT#, where BT is for Bluetooth Serial Port Profile ports and # is a number assigned to the port.

This naming scheme allows API users to generally determine the type of port to use. For example, if an application needs to beam a piece of data, the application could look for IR# ports for opening the connection. The alternative is a trial-and-error approach with all available ports. If the implementation cannot determine a port type, the same naming convention as for an RS-232 port MUST be used.

**Since:** MIDP 2.0

---

| Method Summary |
### Methods

<table>
<thead>
<tr>
<th>int</th>
<th><code>getBaudRate()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets the baudrate for the serial port connection.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int</th>
<th><code>setBaudRate(int baudrate)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sets the baudrate for the serial port connection.</td>
</tr>
</tbody>
</table>

**Methods inherited from interface `InputConnection`**

openDataInputStream, openInputStream

**Methods inherited from interface `Connection`**

close

**Methods inherited from interface `OutputConnection`**

openDataOutputStream, openOutputStream

**Methods inherited from interface `Connection`**

close

---

**Methods**

### `getBaudRate`

**public int `getBaudRate()`**

Gets the baudrate for the serial port connection.

**Returns:**

the baudrate of the connection

**See Also:** `CommConnection.setBaudRate(int)`

### `setBaudRate`

**public int `setBaudRate(int baudrate)`**

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:** `CommConnection.getBaudRate()`
javax.microedition.io

Connector

**Declaration**

```java
public class Connector
```

Object

```java
+-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

**Field Summary**

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Access Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final</td>
<td>READ</td>
<td>Access mode READ. Value: 1</td>
</tr>
<tr>
<td>public static final</td>
<td>READ_WRITE</td>
<td>Access mode READ_WRITE. Value: 3</td>
</tr>
<tr>
<td>public static final</td>
<td>WRITE</td>
<td>Access mode WRITE. Value: 2</td>
</tr>
</tbody>
</table>
Method Summary

<table>
<thead>
<tr>
<th>static javax.microedition.io.Connection</th>
<th>open(String name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create and open a Connection.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static javax.microedition.io.Connection</th>
<th>open(String name, int mode)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create and open a Connection.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static javax.microedition.io.Connection</th>
<th>open(String name, int mode, boolean timeouts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create and open a Connection.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.io.DataInputStream</th>
<th>openDataInputStream(String name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create and open a connection input stream.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.io.DataOutputStream</th>
<th>openDataOutputStream(String name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create and open a connection output stream.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.io.InputStream</th>
<th>openInputStream(String name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create and open a connection input stream.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.io.OutputStream</th>
<th>openOutputStream(String name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create and open a connection output stream.</td>
<td></td>
</tr>
</tbody>
</table>

Methods inherited from class Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Fields

READ

public static final int READ

Access mode READ.

The value 1 is assigned to READ.
Constant value: 1

READ_WRITE

public static final int READ_WRITE

Access mode READ_WRITE.

The value 3 is assigned to READ_WRITE.
Constant value: 3

WRITE

public static final int WRITE

Access mode WRITE.

The value 2 is assigned to WRITE.
Constant value: 2
Methods

open

public static javax.microedition.io.Connection open(String name) throws java.io.IOException

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.
    IOException - If some other kind of I/O error occurs.
    SecurityException - If a requested protocol handler is not permitted.

open

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

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.
    IOException - If some other kind of I/O error occurs.
    SecurityException - If a requested protocol handler is not permitted.

open

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

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.
    IOException - If some other kind of I/O error occurs.
openDataInputStream

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

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.
IOException - If some other kind of I/O error occurs.
SecurityException - If access to the requested stream is not permitted.

openDataOutputStream

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

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.
IOException - If some other kind of I/O error occurs.
SecurityException - If access to the requested stream is not permitted.

openInputStream

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

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.
IOException - If some other kind of I/O error occurs.
SecurityException - If access to the requested stream is not permitted.

openOutputStream

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

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.
   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 Subinterfaces:
javax.microedition.io.HttpsConnection

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
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 distinguish which data is retrieved 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.

```java
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.
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 Content Type
        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 (IOException e) {
        throw e;
    }
}

// Get the response code will open the connection,
// send the request, and read the HTTP response headers.
// The headers are stored until requested.
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:

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

Since: MIDP 1.0

<table>
<thead>
<tr>
<th>Field Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>HTTP Get method. Value: GET</td>
</tr>
<tr>
<td>Method</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>HEAD</td>
</tr>
<tr>
<td>HTTP_ACCEPTED</td>
</tr>
<tr>
<td>HTTP_BAD_GATEWAY</td>
</tr>
<tr>
<td>HTTP_BAD_METHOD</td>
</tr>
<tr>
<td>HTTP_BAD_REQUEST</td>
</tr>
<tr>
<td>HTTP_CLIENT_TIMEOUT</td>
</tr>
<tr>
<td>HTTP_CONFLICT</td>
</tr>
<tr>
<td>HTTP_CREATED</td>
</tr>
<tr>
<td>HTTP_ENTITY_TOO_LARGE</td>
</tr>
<tr>
<td>HTTP_EXPECT_FAILED</td>
</tr>
<tr>
<td>HTTP_FORBIDDEN</td>
</tr>
<tr>
<td>HTTP_GATEWAY_TIMEOUT</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>HTTP_INTERNAL_ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>500: The server encountered an unexpected condition which prevented it from fulfilling the request.</td>
</tr>
<tr>
<td></td>
<td>Value: <strong>500</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>HTTP_LENGTH_REQUIRED</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>411: The server refuses to accept the request without a defined Content-Length.</td>
</tr>
<tr>
<td></td>
<td>Value: <strong>411</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>HTTP_MOVED_PERM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Value: <strong>301</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>HTTP_MOVED_TEMP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>302: The requested resource resides temporarily under a different URI.</td>
</tr>
<tr>
<td></td>
<td>Value: <strong>302</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>HTTP_MULT_CHOICE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Value: <strong>300</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>HTTP_NO_CONTENT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>204: The server has fulfilled the request but does not need to return an entity-body, and might want to return updated meta-information.</td>
</tr>
<tr>
<td></td>
<td>Value: <strong>204</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>HTTP_NOT_ACCEPTABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Value: <strong>406</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>HTTP_NOT_AUTHORITATIVE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>203: The returned meta-information in the entity-header is not the definitive set as available from the origin server.</td>
</tr>
<tr>
<td></td>
<td>Value: <strong>203</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>HTTP_NOT_FOUND</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>404: The server has not found anything matching the Request-URI.</td>
</tr>
<tr>
<td></td>
<td>Value: <strong>404</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>HTTP_NOT_IMPLEMENTED</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>501: The server does not support the functionality required to fulfill the request.</td>
</tr>
<tr>
<td></td>
<td>Value: <strong>501</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>HTTP_NOT_MODIFIED</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Value: <strong>304</strong></td>
</tr>
</tbody>
</table>
| public static final | HTTP_OK         | 200: The request has succeeded.  
|                     |                 | Value: 200                        |
| public static final | HTTP_PARTIAL    | 206: The server has fulfilled the partial GET request for the resource.  
|                     |                 | Value: 206                        |
| public static final | HTTP_PAYMENT_REQUIRED | 402: This code is reserved for future use.  
|                     |                 | Value: 402                        |
| public static final | HTTP_PRECON_FAILED | 412: The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server.  
|                     |                 | Value: 412                        |
| public static final | HTTP_PROXY_AUTH | 407: This code is similar to 401 (Unauthorized), but indicates that the client must first authenticate itself with the proxy.  
|                     |                 | Value: 407                        |
| public static final | HTTP_REQ_TOO_LONG | 414: The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret.  
|                     |                 | Value: 414                        |
| public static final | HTTP_RESET      | 205: The server has fulfilled the request and the user agent SHOULD reset the document view which caused the request to be sent.  
|                     |                 | Value: 205                        |
| public static final | HTTP_SEE_OTHER  | 303: The response to the request can be found under a different URI and SHOULD be retrieved using a GET method on that resource.  
|                     |                 | Value: 303                        |
| public static final | HTTP_TEMP_REDIRECT | 307: The requested resource resides temporarily under a different URI.  
|                     |                 | Value: 307                        |
| public static final | HTTP_UNAUTHORIZED | 401: The request requires user authentication.  
|                     |                 | Value: 401                        |
| public static final | HTTP_UNAVAILABLE | 503: The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.  
|                     |                 | Value: 503                        |
| public static final | HTTP_UNSUPPORTED_RANGE | 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.  
<p>|                     |                 | Value: 416                        |</p>
<table>
<thead>
<tr>
<th>public static final</th>
<th>HTTP_UNSUPPORTED_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Value: <strong>415</strong></td>
</tr>
<tr>
<td>public static final</td>
<td>HTTP_USE_PROXY</td>
</tr>
<tr>
<td></td>
<td>305: The requested resource MUST be accessed through the proxy given by the Location field.</td>
</tr>
<tr>
<td></td>
<td>Value: <strong>305</strong></td>
</tr>
<tr>
<td>public static final</td>
<td>HTTP_VERSION</td>
</tr>
<tr>
<td></td>
<td>505: The server does not support, or refuses to support, the HTTP protocol version that was used in the request message.</td>
</tr>
<tr>
<td></td>
<td>Value: <strong>505</strong></td>
</tr>
<tr>
<td>public static final</td>
<td>POST</td>
</tr>
<tr>
<td></td>
<td>HTTP Post method.</td>
</tr>
<tr>
<td></td>
<td>Value: <strong>POST</strong></td>
</tr>
</tbody>
</table>

**Method Summary**

<table>
<thead>
<tr>
<th>long</th>
<th>getDate()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the value of the date header field.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>long</th>
<th>getExpiration()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the value of the expires header field.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getFile()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the file portion of the URL of this HttpConnection.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getHeaderField(int n)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets a header field value by index.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getHeaderField(String name)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the value of the named header field.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>long</th>
<th>getHeaderFieldDate(String name, long def)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the value of the named field parsed as date.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int</th>
<th>getHeaderFieldInt(String name, int def)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the value of the named field parsed as a number.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getHeaderFieldKey(int n)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets a header field key by index.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getHost()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the host information of the URL of this HttpConnection.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>long</th>
<th>getLastModified()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the value of the last-modified header field.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int</th>
<th>getPort()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the network port number of the URL for this HttpConnection.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getProtocol()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the protocol name of the URL of this HttpConnection.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getQuery()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the query portion of the URL of this HttpConnection.</td>
</tr>
</tbody>
</table>
java.lang.String getRef()
   Returns the ref portion of the URL of this HttpConnection.

java.lang.String getRequestMethod()
   Get the current request method.

java.lang.String getRequestProperty(String key)
   Returns the value of the named general request property for this connection.

int getResponseCode()
   Returns the HTTP response status code.

java.lang.String getResponseMessage()
   Gets the HTTP response message, if any, returned along with the response code from a server.

java.lang.String getURL()
   Return a string representation of the URL for this connection.

void setRequestMethod(String method)
   Set the method for the URL request, one of:

void setRequestProperty(String key, String value)
   Sets the general request property.

Methods inherited from interface ContentConnection
getEncoding, getLength, getType

Methods inherited from interface InputConnection
openDataInputStream, openInputStream

Methods inherited from interface Connection
close

Methods inherited from interface OutputConnection
openDataOutputStream, openOutputStream

Methods inherited from interface Connection
close

Fields

GET

public static final java.lang.String GET
   HTTP Get method.
   Constant value: GET

HEAD

public static final java.lang.String HEAD
   HTTP Head method.
Constant value: HEAD

HTTP_ACCEPTED
public static final int HTTP_ACCEPTED

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

HTTP_BAD_GATEWAY
public static final int HTTP_BAD_GATEWAY

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.
Constant value: 502

HTTP_BAD_METHOD
public static final int HTTP_BAD_METHOD

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

HTTP_BAD_REQUEST
public static final int HTTP_BAD_REQUEST

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

HTTP_CLIENT_TIMEOUT
public static final int HTTP_CLIENT_TIMEOUT

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.
Constant value: 408

HTTP_CONFLICT
public static final int HTTP_CONFLICT

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

HTTP_CREATED
public static final int HTTP_CREATED

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

HTTP_ENTITY_TOO_LARGE
public static final int HTTP_ENTITY_TOO_LARGE

413: The server is refusing to process a request because the request entity is larger than the server is willing
or able to process.
Constant value: 413
<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Description</th>
<th>Constant Value</th>
</tr>
</thead>
</table>
| HTTP_EXPECT_FAILED | public static final int HTTP_EXPECT_FAILED  
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. | 417            |
| HTTP_FORBIDDEN   | public static final int HTTP_FORBIDDEN  
403: The server understood the request, but is refusing to fulfill it. Authorization will not help and the request SHOULD NOT be repeated. | 403            |
| HTTP_GATEWAY_TIMEOUT | public static final int HTTP_GATEWAY_TIMEOUT  
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. | 504            |
| HTTP_GONE        | public static final int HTTP_GONE  
410: The requested resource is no longer available at the server and no forwarding address is known. | 410            |
| HTTP_INTERNAL_ERROR | public static final int HTTP_INTERNAL_ERROR  
500: The server encountered an unexpected condition which prevented it from fulfilling the request. | 500            |
| HTTP_LENGTH_REQUIRED | public static final int HTTP_LENGTH_REQUIRED  
411: The server refuses to accept the request without a defined Content-Length. | 411            |
| HTTP_MOVED_PERM  | public static final int HTTP_MOVED_PERM  
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. | 301            |
| HTTP_MOVED_TEMP  | public static final int HTTP_MOVED_TEMP                                      |                |
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 temporality" to "found". The semantics were not changed. The Location header indicates where the application should resend the request.)*
Constant value: **302**

### HTTP_MULT_CHOICE

```java
public static final int HTTP_MULT_CHOICE
```

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.
Constant value: **300**

### HTTP_NO_CONTENT

```java
public static final int HTTP_NO_CONTENT
```

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

### HTTP_NOT_ACCEPTABLE

```java
public static final int HTTP_NOT_ACCEPTABLE
```

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.
Constant value: **406**

### HTTP_NOT_AUTHORITATIVE

```java
public static final int HTTP_NOT_AUTHORITATIVE
```

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

### HTTP_NOT_FOUND

```java
public static final int HTTP_NOT_FOUND
```

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

### HTTP_NOT_IMPLEMENTED

```java
public static final int HTTP_NOT_IMPLEMENTED
```

501: The server does not support the functionality required to fulfill the request.
Constant value: **501**

### HTTP_NOT_MODIFIED

```java
public static final int HTTP_NOT_MODIFIED
```

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.
Constant value: **304**
HTTP_OK

public static final int HTTP_OK

200: The request has succeeded.
   Constant value: 200

HTTP_PARTIAL

public static final int HTTP_PARTIAL

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

HTTP_PAYMENT_REQUIRED

public static final int HTTP_PAYMENT_REQUIRED

402: This code is reserved for future use.
   Constant value: 402

HTTP_PRECON_FAILED

public static final int HTTP_PRECON_FAILED

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

HTTP_PROXY_AUTH

public static final int HTTP_PROXY_AUTH

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

HTTP_REQ_TOO_LONG

public static final int HTTP_REQ_TOO_LONG

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

HTTP_RESET

public static final int HTTP_RESET

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

HTTP_SEE_OTHER

public static final int HTTP_SEE_OTHER

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

public static final int HTTP_TEMP_REDIRECT

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

HTTP_UNAUTHORIZED

public static final int HTTP_UNAUTHORIZED

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

HTTP_UNAVAILABLE

public static final int HTTP_UNAVAILABLE

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

HTTP_UNSUPPORTED_RANGE

public static final int HTTP_UNSUPPORTED_RANGE

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.
Constant value: 416

HTTP_UNSUPPORTED_TYPE

public static final int HTTP_UNSUPPORTED_TYPE

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.
Constant value: 415

HTTP_USE_PROXY

public static final int HTTP_USE_PROXY

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

HTTP_VERSION

public static final int HTTP_VERSION

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

POST

public static final java.lang.String POST

HTTP Post method.
### Methods

#### getDate

```java
class GET
public long getDate() throws java.io.IOException
```

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:**
IOException - if an error occurred connecting to the server.

#### getExpiration

```java
class GET
public long getExpiration() throws java.io.IOException
```

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:**
IOException - if an error occurred connecting to the server.

#### getFile

```java
class GET
public java.lang.String getFile()
```

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

```java
class GET
public java.lang.String getHeaderField(int n) throws java.io.IOException
```

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:**
IOException - if an error occurred connecting to the server.

#### getHeaderField

```java
class GET
public java.lang.String getHeaderField(String name) throws java.io.IOException
```

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:
   IOException - if an error occurred connecting to the server.

getHeaderFieldDate
public long getHeaderFieldDate(String name, long def)
throws java.io.IOException

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:
   IOException - if an error occurred connecting to the server.

getHeaderFieldInt
public int getHeaderFieldInt(String name, int def)
throws java.io.IOException

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:
   IOException - if an error occurred connecting to the server.

getHeaderFieldKey
public java.lang.String getHeaderFieldKey(int n)
throws java.io.IOException

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:
IOException - if an error occurred connecting to the server.

**getHost**

```java
public java.lang.String getHost()
```

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.

---

**getHost**

```java
public long getLastModified()
throws java.io.IOException
```

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:**

IOException - if an error occurred connecting to the server.

---

**getPort**

```java
public int getPort()
```

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**

```java
public java.lang.String getProtocol()
```

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**

```java
public java.lang.String getQuery()
```

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**

```java
public java.lang.String getRef()
```

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.
getRequestMethod

public java.lang.String getRequestMethod()

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

Returns:
the HTTP request method

See Also: HttpConnection.setRequestMethod(String)

ggetRequestProperty

public java.lang.String getRequestProperty(String key)

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: HttpConnection.setRequestProperty(String, String)

ggetResponseCode

public int getResponseCode() throws java.io.IOException

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:
IOException - if an error occurred connecting to the server.

ggetResponseMessage

public java.lang.String getResponseMessage() throws java.io.IOException

-91-
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:**
IOException - if an error occurred connecting to the server.

---

**getURL**

```
public java.lang.String getURL()
```

Return a string representation of the URL for this connection.

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

---

**setRequestMethod**

```
public void setRequestMethod(String method)
```

Throws java.io.IOException

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:**
IOException - if the method cannot be reset or if the requested method isn't valid for HTTP.

**See Also:** HttpConnection.getRequestMethod()

---

**setRequestProperty**

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

Throws java.io.IOException

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:**
IOException - is thrown if the connection is in the connected state.
See Also: `HttpConnection.getRequestProperty(String)`
javax.microedition.io

HttpsConnection

Declaration

public interface HttpsConnection extends HttpConnection

All Superinterfaces:

javax.microedition.io.HttpConnection

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 defines the scheme.

The SSL V3 protocol as specified in The SSL Protocol Version 3.0 MUST be supported as the secure protocol. An implementation that supports TLS v1.0 and supports section E of RFC 2246 (to provide backwards compatibility with SSL v3.0 servers) is compliant with this requirement. Other secure protocols, such as TLS Protocols Version 1.0 as specified in RFC 2246, WTLS as specified in WAP Forum Specifications June 2000 (WAP 1.2.1) conformance release - Wireless Transport Layer Security document WAP-T99 and WAP(TM) TLS Profile and Tunneling Specification as specified in WAP-219-TLS-20010411-a, MAY also be supported.

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.
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: CertificateException

Fields inherited from interface javax.microedition.io.HttpConnection

GET, HEAD, HTTP_ACCEPTED, HTTP_BAD_GATEWAY, HTTP_BAD_METHOD, HTTP_BAD_REQUEST,
HTTP_CLIENT_TIMEOUT, HTTP_CONFLICT, HTTP_CREATED, HTTP_ENTITY_TOO_LARGE,
HTTP_EXPECT_FAILED, HTTP_FORBIDDEN, HTTP_GATEWAY_TIMEOUT, HTTP_GONE, HTTP_INTERNAL_ERROR,
HTTP_LENGTH_REQUIRED, HTTP_MOVED_PERM, HTTP_MOVED_TEMP, HTTP_MULTI_CHOICE,
HTTP_NO_CONTENT, HTTP_NOT_ACCEPTABLE, HTTP_NOT_AUTHORITATIVE, HTTP_NOT_FOUND,
HTTP_NOT_IMPLEMENTED, HTTP_NOT_MODIFIED, HTTP_OK, HTTP_PARTIAL, HTTP_PAYMENT_REQUIRED,
HTTP_PRECON_FAILED, HTTP_PROXY_AUTH, HTTP_REQ_TOO_LONG, HTTP_RESET, HTTP_SEE_OTHER,
HTTP_TEMP_REDIRECT, HTTP_UNAUTHORIZED, HTTP_UNAVAILABLE, HTTP_UNSUPPORTED_RANGE,
HTTP_UNSUPPORTED_TYPE, HTTP_USE_PROXY, HTTP_VERSION, POST
### Method Summary

<table>
<thead>
<tr>
<th>Java Class</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td><strong>getPort()</strong></td>
<td>Returns the network port number of the URL for this <code>HttpsConnection</code>.</td>
</tr>
<tr>
<td><code>javax.microedition.io.SecurityInfo</code></td>
<td><strong>getSecurityInfo()</strong></td>
<td>Return the security information associated with this successfully opened connection.</td>
</tr>
</tbody>
</table>

### Methods inherited from interface `javax.microedition.io.HttpConnection`

- `getDate`, `getExpiration`, `getFile`, `getHeaderField`, `getHeaderField`, `getHeaderFieldDate`, `getHeaderFieldInt`, `getHeaderFieldKey`, `getHost`, `getLastModified`, `getPort`, `getProtocol`, `getQuery`, `getRef`, `getRequestMethod`, `getRequestProperty`, `getResponseCode`, `getResponseMessage`, `getURL`, `setRequestMethod`, `setRequestProperty`.

### Methods inherited from interface `ContentConnection`

- `getEncoding`, `getLength`, `getType`.

### Methods inherited from interface `InputConnection`

- `openDataInputStream`, `openInputStream`.

### Methods inherited from interface `Connection`

- `close`.

### Methods inherited from interface `OutputConnection`

- `openDataOutputStream`, `openOutputStream`.

### Methods inherited from interface `Connection`

- `close`.

### Methods

#### getPort

```java
public int getPort()
```

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

**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

```java
public javax.microedition.io.SecurityInfo getSecurityInfo()
```

Throws `java.io.IOException`

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:

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. In every protocol, the **AllowedSender** field MUST match with the appropriate address field of the incoming event. The address field to use and the exact syntax and semantics of the address depend on the protocol. However, the address and the filter MUST be compared by exact string matching where the strings are compared character by character and the characters need to match exactly except as allowed by the two wildcard characters: asterisk(\*) and question mark(?).

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]

| 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 UDPOutputConnection 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 SocketConnection 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 responsibility 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.

```plaintext
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() {
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.

```java
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) {
    ...
}
...
```

Sample Ping MIDlet Processing -

```ini
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
```
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

### Method Summary

<table>
<thead>
<tr>
<th>static java.lang.String</th>
<th>getFilter(String connection)</th>
<th>Retrieve the registered filter for a requested connection.</th>
</tr>
</thead>
<tbody>
<tr>
<td>static java.lang.String</td>
<td>getMIDlet(String connection)</td>
<td>Retrieve the registered MIDlet for a requested connection.</td>
</tr>
<tr>
<td>static java.lang.String[]</td>
<td>listConnections(boolean available)</td>
<td>Return a list of registered connections for the current MIDlet suite.</td>
</tr>
<tr>
<td>static long</td>
<td>registerAlarm(String midlet, long time)</td>
<td>Register a time to launch the specified application.</td>
</tr>
<tr>
<td>static void</td>
<td>registerConnection(String connection, String midlet, String filter)</td>
<td>Register a dynamic connection with the application management software.</td>
</tr>
<tr>
<td>static boolean</td>
<td>unregisterConnection(String connection)</td>
<td>Remove a dynamic connection registration.</td>
</tr>
</tbody>
</table>

**Methods inherited from class Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
Methods

**getFilter**

public static java.lang.String getFilter(String connection)

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: PushRegistry.registerConnection(String, String, String)

**getMIDlet**

public static java.lang.String getMIDlet(String connection)

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: PushRegistry.registerConnection(String, String, String)

**listConnections**

public static java.lang.String[] listConnections(boolean available)

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**

public static long registerAlarm(String midlet, long time)

throws java.lang.ClassNotFoundException, javax.microedition.io.ConnectionNotFoundException

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: Date.getTime(), Timer, TimerTask

registerConnection

public static void registerConnection(String connection, String midlet, String filter)
throws java.lang.ClassNotFoundException, java.io.IOException

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
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: PushRegistry.unregisterConnection(String)

unregisterConnection

public static boolean unregisterConnection(String connection)

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:** PushRegistry.registerConnection(String, String, String)
javax.microedition.io

SecureConnection

Declaration

public interface SecureConnection extends SocketConnection

All Superinterfaces:

d javax.microedition.io.SocketConnection

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.

The SSL V3 protocol as specified in The SSL Protocol Version 3.0 MUST be supported as the secure protocol. An
implementation that supports TLS v1.0 and supports section E of RFC 2246 (to provide backwards compatibility
with SSL v3.0 servers) is compliant with this requirement. Other secure protocols, such as TLS Protocols Version
1.0 as specified in RFC 2246 and WAP(TM) TLS Profile and Tunneling Specification as specified in WAP-219-
TLS-20010411-a, MAY also be supported.

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

### Fields inherited from interface javax.microedition.io.SocketConnection
- DELAY, KEEPALIVE, LINGER, RCVBUF, SNDBUF

### Method Summary

<table>
<thead>
<tr>
<th>javax.microedition.io.SecurityInfo</th>
<th>getSecurityInfo()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Return the security information associated with this connection when it was opened.</td>
</tr>
</tbody>
</table>

### Methods inherited from interface javax.microedition.io.SocketConnection
- getAddress, getLocalAddress, getLocalPort, getPort, getSocketOption, setSocketOption

### Methods inherited from interface InputConnection
- openDataInputStream, openInputStream

### Methods inherited from interface Connection
- close

### Methods inherited from interface OutputConnection
- openDataOutputStream, openOutputStream

### Methods inherited from interface Connection
- close
getSecurityInfo

public javax.microedition.io.SecurityInfo getSecurityInfo()
        throws java.io.IOException

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

    Returns:
    the security information associated with this open connection.

    Throws:
    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: CertificateException, SecureConnection, HttpsConnection

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String getCipherSuite()</td>
</tr>
<tr>
<td>Returns the name of the cipher suite in use for the connection.</td>
</tr>
<tr>
<td>java.lang.String getProtocolName()</td>
</tr>
<tr>
<td>Returns the secure protocol name.</td>
</tr>
<tr>
<td>java.lang.String getProtocolVersion()</td>
</tr>
<tr>
<td>Returns the protocol version.</td>
</tr>
<tr>
<td>javax.microedition.pki.Certificate getServerCertificate()</td>
</tr>
<tr>
<td>Returns the Certificate used to establish the secure connection with the server.</td>
</tr>
</tbody>
</table>

Methods

getCipherSuite

public java.lang.String getCipherSuite()

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 implementations 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

public java.lang.String getProtocolName()

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

public java.lang.String getProtocolVersion()

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

public javax.microedition.pki.Certificate getServerCertificate()

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

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.

<table>
<thead>
<tr>
<th>&lt;socket_connection_string&gt;</th>
<th>::= &quot;socket://&quot;</th>
<th>&quot;socket://&quot;&lt;hostport&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;hostport&gt;</td>
<td>::= host &quot;:&quot; port</td>
<td></td>
</tr>
<tr>
<td>&lt;host&gt;</td>
<td>::= omitted for inbound connections, See SocketConnection</td>
<td></td>
</tr>
<tr>
<td>&lt;port&gt;</td>
<td>::= numeric port number (omitted for system assigned port)</td>
<td></td>
</tr>
</tbody>
</table>

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

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td>getLocalAddress()</td>
</tr>
<tr>
<td></td>
<td>Gets the local address to which the socket is bound.</td>
</tr>
<tr>
<td>int</td>
<td>getLocalPort()</td>
</tr>
<tr>
<td></td>
<td>Returns the local port to which this socket is bound.</td>
</tr>
</tbody>
</table>

Methods inherited from interface StreamConnectionNotifier

acceptAndOpen

Methods inherited from interface Connection

close
Methods

getLocalAddress

define: java.lang.String getLocalAddress()
define: throws java.io.IOException

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 reassignment.

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

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

Throws:
IOException - if the connection was closed

See Also: SocketConnection

getLocalPort

define: int getLocalPort()
define: throws java.io.IOException

Returns the local port to which this socket is bound.

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

Throws:
IOException - if the connection was closed

See Also: SocketConnection
javax.microedition.io

SocketConnection

Declaration

public interface SocketConnection extends StreamConnection

All Subinterfaces:

javax.microedition.io.SecureConnection

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

**Field Summary**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELAY</td>
<td>Socket option for the small buffer writing delay (0). Value: 0</td>
</tr>
<tr>
<td>KEEPALIVE</td>
<td>Socket option for the keep alive feature (2). Value: 2</td>
</tr>
<tr>
<td>LINGER</td>
<td>Socket option for the linger time to wait in seconds before closing a connection with pending data output (1). Value: 1</td>
</tr>
<tr>
<td>RCVBUF</td>
<td>Socket option for the size of the receiving buffer (3). Value: 3</td>
</tr>
<tr>
<td>SNDBUF</td>
<td>Socket option for the size of the sending buffer (4). Value: 4</td>
</tr>
</tbody>
</table>

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String getAddress()</td>
<td>Gets the remote address to which the socket is bound.</td>
</tr>
<tr>
<td>java.lang.String getLocalAddress()</td>
<td>Gets the local address to which the socket is bound.</td>
</tr>
<tr>
<td>int getLocalPort()</td>
<td>Returns the local port to which this socket is bound.</td>
</tr>
</tbody>
</table>
### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>int</strong> <code>getPort()</code></td>
<td>Returns the remote port to which this socket is bound.</td>
</tr>
<tr>
<td><strong>int</strong> <code>getSocketOption(byte option)</code></td>
<td>Get a socket option for the connection.</td>
</tr>
<tr>
<td><strong>void</strong> <code>setSocketOption(byte option, int value)</code></td>
<td>Set a socket option for the connection.</td>
</tr>
</tbody>
</table>

#### Methods inherited from interface `InputConnection`

- `openDataInputStream`, `openInputStream`

#### Methods inherited from interface `Connection`

- `close`

#### Methods inherited from interface `OutputConnection`

- `openDataOutputStream`, `openOutputStream`

#### Methods inherited from interface `Connection`

- `close`

### Fields

#### DELAY

*public static final byte* **DELAY**

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.
Constant value: 0

#### KEEPALIVE

*public static final byte* **KEEPALIVE**

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.
Constant value: 2

#### LINGER

*public static final byte* **LINGER**

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.
Constant value: 1

#### RCVBUF

*public static final byte* **RCVBUF**

Socket option for the size of the *receiving buffer* (3).
Constant value: 3
SNDBUF

public static final byte SNDBUF

Socket option for the size of the sending buffer (4).
Constant value: 4

Methods

getAddress

public java.lang.String getAddress ()
  throws java.io.IOException

  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:
  IOException - if the connection was closed.

getLocalAddress

public java.lang.String getLocalAddress ()
  throws java.io.IOException

  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 reassignment.

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

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

  Throws:
  IOException - if the connection was closed.

See Also: ServerSocketConnection

getLocalPort

public int getLocalPort ()
  throws java.io.IOException

  Returns the local port to which this socket is bound.

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

  Throws:
  IOException - if the connection was closed.

See Also: ServerSocketConnection

getPort

public int getPort ()
  throws java.io.IOException

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

Throws:
IOException - if the connection was closed.

**getSocketOption**

```java
public int getSocketOption(byte option)
    throws java.lang.IllegalArgumentException,
            java.io.IOException
```

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
- IOException - if the connection was closed

**See Also:** `SocketConnection.setSocketOption(byte, int)`

**setSocketOption**

```java
public void setSocketOption(byte option, int value)
    throws java.lang.IllegalArgumentException,
            java.io.IOException
```

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
- IOException - if the connection was closed

**See Also:** `SocketConnection.getSocketOption(byte)`
javax.microedition.io

UDPDatagramConnection

Declaration

public interface UDPDatagramConnection extends 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.

<table>
<thead>
<tr>
<th>BNF Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;datagram_connection_string&gt;</code> ::= &quot;datagram://&quot;</td>
<td>&quot;datagram://&quot;&lt;hostport&gt;</td>
</tr>
<tr>
<td><code>&lt;hostport&gt;</code> ::= host&quot;:&quot;port</td>
<td></td>
</tr>
<tr>
<td><code>&lt;host&gt;</code> ::= host name or IP address (omitted for inbound connections)</td>
<td></td>
</tr>
<tr>
<td><code>&lt;port&gt;</code> ::= numeric port number (omitted for system assigned port)</td>
<td></td>
</tr>
</tbody>
</table>

Since: MIDP 2.0

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String getLocalAddress()</td>
<td>Gets the local address to which the datagram connection is bound.</td>
</tr>
<tr>
<td>int getLocalPort()</td>
<td>Returns the local port to which this datagram connection is bound.</td>
</tr>
</tbody>
</table>

Methods inherited from interface DatagramConnection

getMaximumLength, getNominalLength, newDatagram, newDatagram, newDatagram, newDatagram, receive, send

Methods inherited from interface Connection

close

Methods
getLocalAddress

class public java.lang.String getLocalAddress()
   throws java.io.IOException

   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:
   IOException - if the connection was closed.

   See Also: ServerSocketConnection

getLocalPort

class public int getLocalPort()
   throws java.io.IOException

   Returns the local port to which this datagram connection is bound.

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

   Throws:
   IOException - if the connection was closed.

   See Also: ServerSocketConnection
8.1 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.

8.1.1 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 `javax.microedition.lcdui.Screen`.

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 `javax.microedition.lcdui.Canvas` and `javax.microedition.lcdui.Graphics`.

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.

### 8.1.1.1 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.

8.1.1.2 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.

8.1.1.3 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.

8.1.2 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 an exception to the callback serialization rule, which occurs when the `Canvas.serviceRepaints` method is called. This method causes 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`
- `Canvas.keyPressed`
- `Canvas.keyRepeated`
- `Canvas.keyReleased`
- `Canvas.paint`
- `Canvas.pointerDragged`
- `Canvas.pointerPressed`
- `Canvas.pointerReleased`
- `Canvas.showNotify`
- `Canvas.sizeChanged`
- `CommandListener.commandAction`
- `CustomItem.getMinContentHeight`
- `CustomItem.getMinContentWidth`
- `CustomItem.getPrefContentHeight`
- `CustomItem.getPrefContentWidth`
- `CustomItem.hideNotify`
- `CustomItem.keyPressed`
- `CustomItem.keyRepeated`
- `CustomItem.keyReleased`
- `CustomItem.paint`
- `CustomItem.pointerDragged`
- `CustomItem.pointerPressed`
- `CustomItem.pointerReleased`
- `CustomItem.showNotify`
- `CustomItem.sizeChanged`
- `CustomItem.traverse`
- `CustomItem.traverseOut`
- `Displayable.sizeChanged`
- `ItemCommandListener.commandAction`
- `ItemStateListener.itemStateChanged`
- `Runnable.run` resulting from a call to `Display.callSerially`

Note that `Timer` events are not considered UI events. Timer callbacks may run concurrently with UI event callbacks, although `TimerTask` 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 so that work triggered by timer events can be serialized with the UI event callbacks.

8.1.2.1 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 implementation 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.

8.1.2.2 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. Availability of various text input modes (for example, predictive input and numbers-only input) SHOULD be consistent across Java and native
applications. This means, for example, that if predictive text input mode is available in native applications, it SHOULD also be available in Java applications.

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.

**Jog Dial Interaction**

A device can have a 3-way or 5-way jog dial as a control mechanism. A 3-way jog dial is usually a wheel that rotates in two directions (to indicate scrolling) and can also be pressed (to indicate a selection). A 5-way jog dial is typically similar to a 3-way jog dial with the added possibility to tilt the wheel sideways. A jog dial wheel might have the ability to be rolled several steps in each direction. Alternatively, a jog dial wheel might only have the ability to be rotated by a limited angle, returning to the base position when released. When MIDP is implemented on a device with a jog dial, the requirements are as follows:

- An implementation MUST generate key press events in response to jog dial movements. An implementation MAY reuse key codes assigned to device keys with functions similar to jog dial functions (for example, key codes assigned to cursor keys can also be used for jog dial events). However, an implementation MAY choose to use unique key codes for jog dial events.
- An implementation MUST map key press events generated by jog dial movements to game actions. An application can learn game actions via the Canvas.getGameAction() method. The mapping of game actions MUST always be natural in relation to the spatial orientation of the device (that is, mapping MUST be natural from the user point of view. For example, if the jog dial is rolled upwards, UP events are generated). This also means that an implementation MAY need to deliver different key events and game actions depending on the spatial orientation of the device. For example, an application running in a mode that requires a certain orientation of the device can get different key codes than an application running in another orientation. In the same vein, a clamshell device MAY generate different jog dial events when the clamshell is in OPEN or CLOSED mode.
  - A 3-way jog dial MUST generate key events with key codes mapping to game actions UP, DOWN and FIRE.
  - A 5-way jog dial MUST generate key events with key codes mapping to game actions UP, DOWN, LEFT, RIGHT and FIRE.
  - A jog dial MAY also be used to control high level UI objects.

Note: Depending on the mechanics of the jog wheel, the implementation is not necessarily able to generate key repeat events for some movements (for example, when the wheel is rotated).

**8.1.2.3 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.

8.1.2.4 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.

```java
class MovingBlocksCanvas extends Canvas {
    public void keyPressed(int keyCode) {
        int action = getGameAction(keyCode);
        switch (action) {
            case LEFT:
                moveBlockLeft();
                break;
            case RIGHT:
                ... // code for right action
        }
    }
}
```

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:

```java
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:

```java
public static boolean hasPointerEvents();
public static boolean hasPointerMotionEvents();
```

8.1.2.5 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.

8.1.3 Graphics and Text in Low-Level API

8.1.3.1 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:

```java
// 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:

```java
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.

8.1.3.2 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.

### 8.1.3.3 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`:

```java
public static final int getWidth();
public static final int getHeight();
```

### 8.1.3.4 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.

### 8.1.4 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 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 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:

```java
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:

```java
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>;
    }
}
```

8.1.5 Text Truncation in UI Components
Many MIDP LCDUI graphical components can contain text (that is, an alphanumeric string) that is shown to the user. Examples of such components are List, TextBox, Alert, StringItem, Form, and Item. An implementation often needs to truncate such visible text because it does not fit in the designated space of a given UI component. In this case, an implementation MUST use an appropriate visual indication (for example an ellipsis symbol) to signal the user that the text is truncated. The actual symbol or symbols used to represent the truncated text depends on the locale that is currently selected in the device. However, the visual indication SHOULD be consistent with the visual indication used in the device’s native UI.

8.1.6 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.

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
<table>
<thead>
<tr>
<th><strong>Class Summary</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interfaces</strong></td>
</tr>
<tr>
<td><strong>Choice</strong></td>
</tr>
<tr>
<td><strong>CommandListener</strong></td>
</tr>
<tr>
<td><strong>ItemCommandListener</strong></td>
</tr>
<tr>
<td><strong>ItemStateListener</strong></td>
</tr>
<tr>
<td><strong>Classes</strong></td>
</tr>
<tr>
<td><strong>Alert</strong></td>
</tr>
<tr>
<td><strong>AlertType</strong></td>
</tr>
<tr>
<td><strong>Canvas</strong></td>
</tr>
<tr>
<td><strong>ChoiceGroup</strong></td>
</tr>
<tr>
<td><strong>Command</strong></td>
</tr>
<tr>
<td><strong>CustomItem</strong></td>
</tr>
<tr>
<td><strong>DateField</strong></td>
</tr>
<tr>
<td><strong>Display</strong></td>
</tr>
<tr>
<td><strong>Displayable</strong></td>
</tr>
<tr>
<td><strong>Font</strong></td>
</tr>
<tr>
<td><strong>Form</strong></td>
</tr>
<tr>
<td><strong>Gauge</strong></td>
</tr>
<tr>
<td><strong>Graphics</strong></td>
</tr>
<tr>
<td><strong>Image</strong></td>
</tr>
<tr>
<td><strong>ImageItem</strong></td>
</tr>
<tr>
<td><strong>Item</strong></td>
</tr>
<tr>
<td><strong>List</strong></td>
</tr>
<tr>
<td><strong>Screen</strong></td>
</tr>
<tr>
<td><strong>Spacer</strong></td>
</tr>
<tr>
<td><strong>StringItem</strong></td>
</tr>
<tr>
<td><strong>TextBox</strong></td>
</tr>
<tr>
<td><strong>TextField</strong></td>
</tr>
<tr>
<td><strong>Ticker</strong></td>
</tr>
</tbody>
</table>
javax.microedition.lcdui

Alert

Declaration

public class Alert extends Screen

Object

| ~---javax.microedition.lcdui.Displayable
| ^---javax.microedition.lcdui.Screen
| +--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().

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 object that is used as an activity or progress indicator. By default, an Alert has no activity indicator; one may be set with the Alert.setIndicator(Gauge) 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 Alert.DISMISS_COMMAND 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 Alert.FOREVER. 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 Alert.setCommandListener(CommandListener) 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) method and the Display.setCurrent(Displayable) 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 with Displayable.setTicker on an Alert, however it may not be displayed due to implementation restrictions.

Since: MIDP 1.0
See Also: AlertType

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final DISMISS_COMMAND</td>
<td>A Command delivered to a listener to indicate that the Alert has been dismissed.</td>
</tr>
<tr>
<td>public static final FOREVER</td>
<td>FOREVER indicates that an Alert is kept visible until the user dismisses it. Value: -2</td>
</tr>
</tbody>
</table>

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Alert(String title)</td>
<td>Constructs a new, empty Alert object with the given title.</td>
</tr>
<tr>
<td>public Alert(String title, String alertText, Image alertImage, AlertType alertType)</td>
<td>Constructs a new Alert object with the given title, content string and image, and alert type.</td>
</tr>
</tbody>
</table>
### Method Summary

#### void `addCommand(Command cmd)`
Similar to `Displayable.addCommand(Command)`, however when the application first adds a command to an Alert, `Alert.DISMISS_COMMAND` is implicitly removed.

#### int `getDefaultTimeout()`
Gets the default time for showing an Alert.

#### `javax.microedition.lcdui.Image` `getImage()`
Gets the `Image` used in the Alert.

#### `javax.microedition.lcdui.Gauge` `getIndicator()`
Gets the activity indicator for this Alert.

#### `java.lang.String` `getString()`
Gets the text string used in the Alert.

#### int `getTimeout()`
Gets the time this Alert will be shown.

#### `javax.microedition.lcdui.AlertType` `getType()`
Gets the type of the Alert.

#### void `removeCommand(Command cmd)`
Similar to `Displayable.removeCommand(Command)`, however when the application removes the last command from an Alert, `Alert.DISMISS_COMMAND` is implicitly added.

#### void `setCommandListener(CommandListener l)`
The same as `Displayable.setCommandListener(CommandListener)` but with the following additional semantics.

#### void `setImage(Image img)`
Sets the `Image` used in the Alert.

#### void `setIndicator(Gauge indicator)`
Sets an activity indicator on this Alert.

#### void `setString(String str)`
Sets the text string used in the Alert.

#### void `setTimeout(int time)`
Set the time for which the Alert is to be shown.

#### void `setType(AlertType type)`
Sets the type of the Alert.

---

Methods inherited from class `javax.microedition.lcdui.Displayable`:
- `addCommand`, `addHeight`, `getTicker`, `getTitle`, `getWidth`, `isShown`, `removeCommand`, `setCommandListener`, `setTicker`, `setTitle`, `sizeChanged`

Methods inherited from class `Object`:
- `equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`
**Fields**

**DISMISS_COMMAND**

```java
public static final javax.microedition.lcdui.Command DISMISS_COMMAND
```

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**

```java
public static final int FOREVER
```

FOREVER indicates that an Alert is kept visible until the user dismisses it. It is used as a value for the parameter to `setTimeout()` 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.

Constant value: -2

**Constructors**

**Alert**

```java
public Alert(String title)
```

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

```java
Alert(title, null, null, null)
```

**Parameters:**
- `title` - the title string, or `null`

**See Also:** `javax.microedition.lcdui.Alert(String, String, Image, AlertType)`

---

**Alert**

```java
public Alert(String title,
    String alertText, 
    Image alertImage,
    AlertType alertType)
``
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. 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**

```java
public void addCommand(Command cmd)
```

Similar to Displayable.addCommand(Command), however when the application first adds a command to an Alert, Alert.DISMISS_COMMAND is implicitly removed. Calling this method with DISMISS_COMMAND as the parameter has no effect.

Parameters:
- cmd - the command to be added

Throws:
- NullPointerException - if cmd is null

**getDefaultTimeout**

```java
public int getDefaultTimeout()
```

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, 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**

```java
public javax.microedition.lcdui.Image getImage()
```

Gets the Image used in the Alert.

Returns:
- the Alert's image, or null if there is no image

See Also: Alert.setImage(Image)

**getIndicator**

```java
public javax.microedition.lcdui.Gauge getIndicator()
```

Gets the activity indicator for this Alert.

Returns:
- a reference to this Alert's activity indicator, or null if there is none

See Also: Alert.setIndicator(Gauge)

Since: MIDP 2.0
**getString**

```java
public java.lang.String getString()
```

Gets the text string used in the `Alert`.

**Returns:**
- the Alert's text string, or null if there is no text

**See Also:** `Alert.setString(String)`

---

**getTimeout**

```java
public int getTimeout()
```

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 `Alert.setTimeout(int)`. 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:** `Alert.setTimeout(int)`

---

**getType**

```java
public javax.microedition.lcdui.AlertType getType()
```

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:** `Alert.setType(AlertType)`

---

**removeCommand**

```java
public void removeCommand(Command cmd)
```

Similar to `Displayable.removeCommand(Command)`, however when the application removes the last command from an `Alert`, `Alert.DISMISS_COMMAND` is implicitly added. Calling this method with `DISMISS_COMMAND` as the parameter has no effect.

**Parameters:**
- cmd - the command to be removed

---

**setCommandListener**

```java
public void setCommandListener(CommandListener l)
```

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.

**Parameters:**
- l - the new listener, or null

---

**setImage**

```java
public void setImage(Image img)
```

...
**setImage**

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

```java
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:** Alert.getImage()

---

**setIndicator**

```java
public void setIndicator(Gauge indicator)
```

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

**See Also:** Alert.getIndicator()

**Since:** MIDP 2.0

---

**setString**

```java
public void setString(String str)
```

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:** Alert.getString()

---

**setTimeout**

```java
public void setTimeout(int time)
```

Sets 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: Alert.getTimeout()

**setType**

`public void setType(AlertType type)`

Sets the type of the Alert. The handling and behavior of specific AlertTypes is described in AlertType.

**Parameters:**

- `type` - an AlertType, or `null` if the Alert has no specific type

See Also: Alert.getType()
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

Field Summary

| public static final | ALARM | An ALARM AlertType is a hint to alert the user to an event for which the user has previously requested to be notified. |
| public static final | CONFIRMATION | A CONFIRMATION AlertType is a hint to confirm user actions. |
| public static final | ERROR | An ERROR AlertType is a hint to alert the user to an erroneous operation. |
| public static final | INFO | An INFO AlertType typically provides non-threatening information to the user. |
| public static final | WARNING | A WARNING AlertType is a hint to warn the user of a potentially dangerous operation. |

Constructor Summary

| protected | AlertType() |
| Protected constructor for subclasses. |

Method Summary

| boolean | playSound(Display display) |
| Alert the user by playing the sound for this AlertType. |

Methods inherited from class Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
Fields

**ALARM**

```java
public static final javax.microedition.lcdui.AlertType ALARM
```

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**

```java
public static final javax.microedition.lcdui.AlertType CONFIRMATION
```

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**

```java
public static final javax.microedition.lcdui.AlertType ERROR
```

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**

```java
public static final javax.microedition.lcdui.AlertType INFO
```

An INFO AlertType typically provides non-threatening information to the user. For example, a simple splash screen might be an INFO AlertType.

**WARNING**

```java
public static final javax.microedition.lcdui.AlertType WARNING
```

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**

```java
protected AlertType()
```

Protected constructor for subclasses.

Methods

**playSound**

```java
public boolean playSound(Display display)
```

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
Canvas

Declaration

public abstract class Canvas extends Displayable

Object

javax.microedition.lcdui.Canvas

Direct Known Subclasses:
javax.microedition.lcdui.game.GameCanvas

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 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 few 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, KEY_NUM1, KEY_NUM2, KEY_NUM3, KEY_NUM4, KEY_NUM5, KEY_NUM6, KEY_NUM7, KEY_NUM8, KEY_NUM9, KEY_STAR, and KEY_POUND. (These key codes correspond to keys on an ITU-T standard telephone keypad.) Other keys may be present on the keyboard, and they will generally have key codes distinct from those listed 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:
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 or TextField 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 getKeyNamemethod 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, DOWN, LEFT, RIGHT, FIRE, GAME_A, GAME_B, GAME_C, and GAME_D.

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) method, and it can translate a game action into a key code using the getKeyCode(int gameAction) 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:

\[
g == \text{getGameAction}((\text{get keyCode}(g))) \quad \text{(1)}
\]
\[
k == \text{getKeyCode}((\text{get GameAction}(k))) \quad \text{(2)}
\]

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() 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.

If the device keypad can be used in different modes, and some modes restrict user access to certain keys, or if the device has several different keypads that are not meant to used simultaneously, a mobile handset implementation is NOT REQUIRED to ensure that the mapping between key codes and game action does not change during the execution of the application. Consequently, applications SHOULD NOT use the getKeyCode() method to learn which key is assigned to a certain game action. Instead, applications SHOULD use the getGameAction() method to check which game action is associated with a given key code.

Commands

It is also possible for the user to issue commands 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() 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() method is called prior to the Canvas actually being made visible on the display, and the hideNotify() 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() 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 Canvas.setFullScreenMode(boolean) method.

Calling Canvas.setFullScreenMode(boolean) may result in sizeChanged() 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

### Field Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static</td>
<td>DOWN</td>
<td>Constant for the DOWN game action. Value: 6</td>
</tr>
<tr>
<td>final</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public static</td>
<td>FIRE</td>
<td>Constant for the FIRE game action. Value: 8</td>
</tr>
<tr>
<td>final</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public static</td>
<td>GAME_A</td>
<td>Constant for the general purpose &quot;A&quot; game action. Value: 9</td>
</tr>
<tr>
<td>final</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public static</td>
<td>GAME_B</td>
<td>Constant for the general purpose &quot;B&quot; game action. Value: 10</td>
</tr>
<tr>
<td>final</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public static</td>
<td>GAME_C</td>
<td>Constant for the general purpose &quot;C&quot; game action. Value: 11</td>
</tr>
<tr>
<td>final</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public static</td>
<td>GAME_D</td>
<td>Constant for the general purpose &quot;D&quot; game action. Value: 12</td>
</tr>
<tr>
<td>final</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public static</td>
<td>KEY_NUM0</td>
<td>keyCode for ITU-T key 0. Value: 48</td>
</tr>
<tr>
<td>final</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public static</td>
<td>KEY_NUM1</td>
<td>keyCode for ITU-T key 1. Value: 49</td>
</tr>
<tr>
<td>final</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public static</td>
<td>KEY_NUM2</td>
<td>keyCode for ITU-T key 2. Value: 50</td>
</tr>
<tr>
<td>final</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public static</td>
<td>KEY_NUM3</td>
<td>keyCode for ITU-T key 3. Value: 51</td>
</tr>
<tr>
<td>final</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public static</td>
<td>KEY_NUM4</td>
<td>keyCode for ITU-T key 4. Value: 52</td>
</tr>
<tr>
<td>final</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public static</td>
<td>KEY_NUM5</td>
<td>keyCode for ITU-T key 5. Value: 53</td>
</tr>
<tr>
<td>final</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public static</td>
<td>KEY_NUM6</td>
<td>keyCode for ITU-T key 6. Value: 54</td>
</tr>
<tr>
<td>final</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
public static final KEY_NUM7 keyCode for ITU-T key 7.
Value: 55

public static final KEY_NUM8 keyCode for ITU-T key 8.
Value: 56

public static final KEY_NUM9 keyCode for ITU-T key 9.
Value: 57

public static final KEY_POUND keyCode for ITU-T key "pound" (#).
Value: 35

public static final KEY_STAR keyCode for ITU-T key "star" (*).
Value: 42

public static final LEFT Constant for the LEFT game action.
Value: 2

public static final RIGHT Constant for the RIGHT game action.
Value: 5

public static final UP Constant for the UP game action.
Value: 1

Constructor Summary
protected Canvas()
Constructs a new Canvas object.

Method Summary
int getGameAction(int keyCode)
Gets the game action associated with the given key code of the device.

int getHeight()
Gets the height in pixels of the displayable area of the Canvas.

int getKeyCode(int gameAction)
Gets a key code that corresponds to the specified game action on the device.

java.lang.String getKeyName(int keyCode)
Gets an informative key string for a key.

int getWidth()
Gets the width in pixels of the displayable area of the Canvas.

boolean hasPointerEvents()
Checks if the platform supports pointer press and release events.
<table>
<thead>
<tr>
<th>boolean</th>
<th>hasPointerMotionEvents()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Checks if the platform supports pointer motion events (pointer dragged).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>boolean</th>
<th>hasRepeatEvents()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Checks if the platform can generate repeat events when key is kept down.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>hideNotify()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The implementation calls hideNotify() shortly after the Canvas has been removed from the display.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>boolean</th>
<th>isDoubleBuffered()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Checks if the Canvas is double buffered by the implementation.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>keyPressed(int keyCode)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Called when a key is pressed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>keyReleased(int keyCode)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Called when a key is released.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>keyRepeated(int keyCode)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Called when a key is repeated (held down).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>abstract void</th>
<th>paint(Graphics g)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Renders the Canvas.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>pointerDragged(int x, int y)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Called when the pointer is dragged.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>pointerPressed(int x, int y)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Called when the pointer is pressed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>pointerReleased(int x, int y)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Called when the pointer is released.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>repaint()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Requests a repaint for the entire Canvas.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>repaint(int x, int y, int width, int height)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Requests a repaint for the specified region of the Canvas.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>serviceRepaints()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Forces any pending repaint requests to be serviced immediately.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>setFullScreenMode(boolean mode)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Controls whether the Canvas is in full-screen mode or in normal mode.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>showNotify()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The implementation calls showNotify() immediately prior to this Canvas being made visible on the display.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>sizeChanged(int w, int h)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Called when the drawable area of the Canvas has been changed.</td>
</tr>
</tbody>
</table>

**Methods inherited from class** `javax.microedition.lcdui.Displayable`

- addCommand, getHeight, getTicker, getTitle, getWidth, isShown, removeCommand, setCommandListener, setTicker, setTitle, sizeChanged

**Methods inherited from class** `java.lang.Object`

- equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
Fields

**DOWN**

```java
public static final int DOWN

    Constant for the DOWN game action.
    Constant value 6 is set to DOWN.
    Constant value: 6
```

**FIRE**

```java
public static final int FIRE

    Constant for the FIRE game action.
    Constant value 8 is set to FIRE.
    Constant value: 8
```

**GAME_A**

```java
public static final int GAME_A

    Constant for the general purpose "A" game action.
    Constant value 9 is set to GAME_A.
    Constant value: 9
```

**GAME_B**

```java
public static final int GAME_B

    Constant for the general purpose "B" game action.
    Constant value 10 is set to GAME_B.
    Constant value: 10
```

**GAME_C**

```java
public static final int GAME_C

    Constant for the general purpose "C" game action.
    Constant value 11 is set to GAME_C.
    Constant value: 11
```

**GAME_D**

```java
public static final int GAME_D

    Constant for the general purpose "D" game action.
    Constant value 12 is set to GAME_D.
    Constant value: 12
```
KEY_NUM0
public static final int KEY_NUM0

    keyCode for ITU-T key 0.
    Constant value 48 is set to KEY_NUM0.
    Constant value: 48

KEY_NUM1
public static final int KEY_NUM1

    keyCode for ITU-T key 1.
    Constant value 49 is set to KEY_NUM1.
    Constant value: 49

KEY_NUM2
public static final int KEY_NUM2

    keyCode for ITU-T key 2.
    Constant value 50 is set to KEY_NUM2.
    Constant value: 50

KEY_NUM3
public static final int KEY_NUM3

    keyCode for ITU-T key 3.
    Constant value 51 is set to KEY_NUM3.
    Constant value: 51

KEY_NUM4
public static final int KEY_NUM4

    keyCode for ITU-T key 4.
    Constant value 52 is set to KEY_NUM4.
    Constant value: 52

KEY_NUM5
public static final int KEY_NUM5

    keyCode for ITU-T key 5.
    Constant value 53 is set to KEY_NUM5.
    Constant value: 53

KEY_NUM6
public static final int KEY_NUM6

    keyCode for ITU-T key 6.
    Constant value 54 is set to KEY_NUM6.
    Constant value: 54
KEY_NUM7
public static final int KEY_NUM7
    keyCode for ITU-T key 7.
    Constant value 55 is set to KEY_NUM7.
    Constant value: 55

KEY_NUM8
public static final int KEY_NUM8
    keyCode for ITU-T key 8.
    Constant value 56 is set to KEY_NUM8.
    Constant value: 56

KEY_NUM9
public static final int KEY_NUM9
    keyCode for ITU-T key 9.
    Constant value 57 is set to KEY_NUM9.
    Constant value: 57

KEY_POUND
public static final int KEY_POUND
    keyCode for ITU-T key "pound" (#).
    Constant value 35 is set to KEY_POUND.
    Constant value: 35

KEY_STAR
public static final int KEY_STAR
    keyCode for ITU-T key "star" (*).
    Constant value 42 is set to KEY_STAR.
    Constant value: 42

LEFT
public static final int LEFT
    Constant for the LEFT game action.
    Constant value 2 is set to LEFT.
    Constant value: 2

RIGHT
public static final int RIGHT
    Constant for the RIGHT game action.
    Constant value 5 is set to RIGHT.
Constant value: 5

UP

public static final int UP

Constant for the UP game action.

Constant value 1 is set to UP.
Constant value: 1

Constructors

Canvas

protected Canvas()

Constructs a new Canvas object.

Methods

getGameAction

public int getGameAction(int keyCode)

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.

The mapping between key codes and game actions will not change during the execution of the application. (Note: Follow link to the last paragraph in Game Actions section for a description of an exception case)

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

public int getHeight()

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 Canvas.sizeChanged(int, int) method.

Returns:
height of the displayable area

getKeyCode

public int getKeyCode(int gameAction)

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. 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 Canvas.getGameAction(int) 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. (Note: Follow link to the last paragraph in Game Actions section for a description of an exception case)
**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**

public java.lang.String getKeyName(int keyCode)

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**

public int getWidth()

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 Canvas.sizeChanged(int, int) method.

**Returns:**

width of the displayable area

---

**hasPointerEvents**

public boolean hasPointerEvents()

Checks if the platform supports pointer press and release events. Pointer press and release events MUST be supported if the underlying hardware supports this feature.

**Returns:**

true if the device supports pointer events If the underlying hardware supports pointer events then the return value shall always be true.

---

**hasPointerMotionEvents**

public boolean hasPointerMotionEvents()
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. Pointer motion events MUST be supported
if the underlying hardware supports this feature.

**Returns:**
true if the device supports pointer motion events. If the underlying hardware supports pointer motion
events then the return value shall always be true.

---

**hasRepeatEvents**

public boolean hasRepeatEvents()

Checks if the platform can generate repeat events when key is kept down. Repeat events MUST be supported
by all implementations.

**Returns:**
true always

---

**hideNotify**

protected void hideNotify()

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**

public boolean isDoubleBuffered()

Checks if the Canvas is double buffered by the implementation. All implementations MUST support double
buffered graphics.

**Returns:**
true always

---

**keyPressed**

protected void keyPressed(int keyCode)

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**

protected void keyReleased(int keyCode)

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

protected void keyRepeated(int keyCode)

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: Canvas.hasRepeatEvents()

paint

protected abstract void paint(Graphics g)

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();
- the stroke style is SOLID;
- 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

protected void pointerDragged(int x, int y)
Called when the pointer is dragged.

The `hasPointerMotionEvents()` 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**

```java
protected void pointerPressed(int x, int y)
```

Called when the pointer is pressed.

The `hasPointerEvents()` 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**

```java
protected void pointerReleased(int x, int y)
```

Called when the pointer is released.

The `hasPointerEvents()` 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**

```java
public final void repaint()
```

Requests a repaint for the entire `Canvas`. The effect is identical to

```java
repaint(0, 0, getWidth(), getHeight());
```

**repaint**

```java
public final void repaint(int x, int y, int width, int height)
```
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() or serviceRepaints(), 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), Canvas.serviceRepaints()

---

serviceRepaints

public final void serviceRepaints()

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() 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)

---

setFullScreenMode

public void setFullScreenMode(boolean mode)

Controls whether the Canvas is in full-screen mode or in normal mode. If Canvas is in full-screen mode, the following requirements apply:

- If CommandListener is set for a Canvas, an implementation MUST display the commands in the User Interface when a key normally reserved for the delivery of commands is pressed and deliver the selected command.
- If no CommandListener is set for a Canvas, an implementation MUST deliver key code events for key presses that are otherwise reserved for the delivery of commands.

Parameters:
- mode - true if the Canvas is to be in full screen mode, false otherwise

Since: MIDP 2.0

---

showNotify

protected void showNotify()

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

protected void sizeChanged(int w, int h)

Called when the drawable area of the Canvas has been changed. This method has augmented semantics compared to Displayable.sizeChanged.

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.

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:
  javax.microedition.lcdui.ChoiceGroup, javax.microedition.lcdui.List

Description

Choice defines an API for a user interface components implementing selection from predefined number of choices. Such UI components are list and ChoiceGroup. 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 part, and a font attribute that are all treated as a unit. The text 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.

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, insert, or set 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) and Display.getBestImageHeight(int).

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 Choice.setFitPolicy(int) and Choice.getFitPolicy() methods control this preference. The valid settings are Choice.TEXT_WRAP_DEFAULT, Choice.TEXT_WRAP_ON, and Choice.TEXT_WRAP_OFF. 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), exclusive-choice, multiple-choice, and pop-up (valid only for ChoiceGroup).

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 and setSelectedIndex methods. This state is available to the application through the getSelectedFlags and getSelectedIndex 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 or a ItemStateListener 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.
### Field Summary

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXCLUSIVE</td>
<td>1</td>
</tr>
<tr>
<td>IMPLICIT</td>
<td>3</td>
</tr>
<tr>
<td>MULTIPLE</td>
<td>2</td>
</tr>
<tr>
<td>POPUP</td>
<td>4</td>
</tr>
<tr>
<td>TEXT_WRAP_DEFAULT</td>
<td>0</td>
</tr>
<tr>
<td>TEXT_WRAP_OFF</td>
<td>2</td>
</tr>
<tr>
<td>TEXT_WRAP_ON</td>
<td>1</td>
</tr>
</tbody>
</table>

**EXCLUSIVE** is a choice having exactly one element selected at time.

**IMPLICIT** is a choice in which the currently focused element is selected when a Command is initiated.

**MULTIPLE** is a choice that can have arbitrary number of elements selected at a time.

**POPUP** is a choice having exactly one element selected at a time.

**TEXT_WRAP_DEFAULT** 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.

**TEXT_WRAP_OFF** Constant for hinting that text element contents should be limited to a single line.

**TEXT_WRAP_ON** Constant for hinting that text element contents should be wrapped to to multiple lines if necessary to fit available content space.

### Method Summary

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int append</td>
<td>Appends an element to the Choice.</td>
</tr>
<tr>
<td>void delete</td>
<td>Deletes the element referenced by elementNum.</td>
</tr>
<tr>
<td>void deleteAll</td>
<td>Deletes all elements from this Choice, leaving it with zero elements.</td>
</tr>
<tr>
<td>int getFitPolicy</td>
<td>Gets the application's preferred policy for fitting Choice element contents to the available screen space.</td>
</tr>
<tr>
<td>Font getFont</td>
<td>Gets the application's preferred font for rendering the specified element of this Choice.</td>
</tr>
<tr>
<td>Image getImage</td>
<td>Gets the Image part of the element referenced by elementNum.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>int getSelectedFlags(boolean[] selectedArray_return)</td>
<td>Queries the state of a Choice and returns the state of all elements in the boolean array selectedArray_return.</td>
</tr>
<tr>
<td>int getSelectedIndex()</td>
<td>Returns the index number of an element in the Choice that is selected.</td>
</tr>
<tr>
<td>java.lang.String getString(int elementNum)</td>
<td>Gets the String part of the element referenced by elementNum.</td>
</tr>
<tr>
<td>void insert(int elementNum, String stringPart, Image imagePart)</td>
<td>Inserts an element into the Choice just prior to the element specified.</td>
</tr>
<tr>
<td>boolean isSelected(int elementNum)</td>
<td>Gets a boolean value indicating whether this element is selected.</td>
</tr>
<tr>
<td>void set(int elementNum, String stringPart, Image imagePart)</td>
<td>Sets the String and Image parts of the element referenced by elementNum, replacing the previous contents of the element.</td>
</tr>
<tr>
<td>void setFitPolicy(int fitPolicy)</td>
<td>Sets the application's preferred policy for fitting Choice element contents to the available screen space.</td>
</tr>
<tr>
<td>void setFont(int elementNum, Font font)</td>
<td>Sets the application's preferred font for rendering the specified element of this Choice.</td>
</tr>
<tr>
<td>void setSelectedFlags(boolean[] selectedArray)</td>
<td>Attempts to set the selected state of every element in the Choice.</td>
</tr>
<tr>
<td>void setSelectedIndex(int elementNum, boolean selected)</td>
<td>For MULTIPLE, this simply sets an individual element's selected state.</td>
</tr>
<tr>
<td>int size()</td>
<td>Gets the number of elements present.</td>
</tr>
</tbody>
</table>

### Fields

**EXCLUSIVE**

public static final int EXCLUSIVE

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.
Constant value: 1

**IMPLICIT**

public static final int IMPLICIT

IMPLICIT is a choice in which the currently focused element is selected when a Command is initiated.

The IMPLICIT type is not valid for ChoiceGroup objects.

Value 3 is assigned to IMPLICIT.
Constant value: 3
MULTIPLE

public static final int MULTIPLE

MULTIPLE is a choice that can have arbitrary number of elements selected at a time.

Value 2 is assigned to MULTIPLE.
Constant value: 2

POPUP

public static final int POPUP

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 objects.

Value 4 is assigned to POPUP.
Constant value: 4

Since: MIDP 2.0

TEXT_WRAP_DEFAULT

public static final int TEXT_WRAP_DEFAULT

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.
Constant value: 0

See Also: Choice.getFitPolicy(), Choice.setFitPolicy(int)

Since: MIDP 2.0

TEXT_WRAP_OFF

public static final int TEXT_WRAP_OFF

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.
Constant value: 2

See Also: Choice.getFitPolicy(), Choice.setFitPolicy(int)

Since: MIDP 2.0

TEXT_WRAP_ON

public static final int TEXT_WRAP_ON
Constant for hinting that text element contents should be wrapped 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.
Constant value: 1

**See Also:** Choice.getFitPolicy(), Choice.setFitPolicy(int)

**Since:** MIDP 2.0

### Methods

#### append

```java
public int append(String stringPart, Image imagePart)
```

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

```java
deprecated public void delete(int elementNum)
```

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

```java
deprecated public void deleteAll()
```

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

```java
public int getFitPolicy()
```

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 Choice.TEXT_WRAP_DEFAULT, Choice.TEXT_WRAP_ON, or Choice.TEXT_WRAP_OFF

**See Also:** Choice.setFitPolicy(int)
Since: MIDP 2.0

getFont

public javax.microedition.lcdui.Font getFont(int elementNum)

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

See Also: Choice.setFont(int, Font)
Since: MIDP 2.0

getImage

public javax.microedition.lcdui.Image getImage(int elementNum)

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: Choice.getString(int)

getSelectedFlags

public int getSelectedFlags(boolean[] selectedArray_return)

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: Choice.setSelectedFlags(boolean[])
getSelectedIndex

public int getSelectedIndex()

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.

Returns:
index of selected element, or -1 if none

See Also: Choice.setSelectedIndex(int, boolean)

getString

public java.lang.String getString(int elementNum)

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: Choice.getImage(int)

insert

public void insert(int elementNum,
                   String stringPart,
                   Image imagePart)

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().

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

public boolean isSelected(int elementNum)

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:
**set**

```java
public void set(int elementNum,
                String stringPart,
                Image imagePart)
```

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**

```java
public void setFitPolicy(int fitPolicy)
```

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`, `Choice.TEXT_WRAP_ON`, and `Choice.TEXT_WRAP_OFF`. 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

See Also: `Choice.getFitPolicy()`
Since: MIDP 2.0

---

**setFont**

```java
public void setFont(int elementNum,
                    Font font)
```

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

See Also: `Choice.getFont(int)`
Since: MIDP 2.0
**setSelectedFlags**

public void **setSelectedFlags**(boolean[] selectedArray)

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 collects the selection status

**Throws:**

- IllegalArgumentException - if selectedArray is shorter than the size of the Choice
- NullPointerException - if selectedArray is null

**See Also:** Choice.getSelectedFlags(boolean[])

**setSelectedIndex**

public void **setSelectedIndex**(int elementNum, boolean selected)

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:** Choice.getSelectedIndex()

**size**

public int **size**()

Gets the number of elements present.

**Returns:**

the number of elements in the Choice
javax.microedition.lcdui

ChoiceGroup

Declaration

public class ChoiceGroup extends Item implements Choice

Object

| javax.microedition.lcdui.Item
| javax.microedition.lcdui.ChoiceGroup

All Implemented Interfaces:
javax.microedition.lcdui.Choice

Description

A ChoiceGroup is a group of selectable elements intended to be placed within a Form. 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 interface.

Since: MIDP 1.0

Fields inherited from class javax.microedition.lcdui.Item

BUTTON, HYPERLINK, LAYOUT_2, LAYOUT_BOTTOM, LAYOUT_CENTER, LAYOUT_DEFAULT, LAYOUT_EXPAND, LAYOUT_LEFT, LAYOUT_NEWLINE_AFTER, LAYOUT_NEWLINE_BEFORE, LAYOUT_RIGHT, LAYOUT_SHRINK, LAYOUT_TOP, LAYOUT_VCENTER, LAYOUT_VEXPAND, LAYOUT_VSHRINK, PLAIN

Fields inherited from interface javax.microedition.lcdui.Choice

EXCLUSIVE, IMPLICIT, MULTIPLE, POPUP, TEXT_WRAP_DEFAULT, TEXT_WRAP_OFF, TEXT_WRAP_ON

Constructor Summary

| public ChoiceGroup(String label, int choiceType) |
| Creates a new, empty ChoiceGroup, specifying its title and its type. |

| public ChoiceGroup(String label, int choiceType, String[] stringElements, Image[] imageElements) |
| 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. |

Method Summary

| int append(String stringPart, Image imagePart) |
| Appends an element to the ChoiceGroup. |

| void delete(int elementNum) |
| Deletes the element referenced by elementNum. |

| void deleteAll() |
| Deletes all elements from this ChoiceGroup. |
int getFitPolicy()
Gets the application's preferred policy for fitting Choice element contents to the available screen space.

javax.microedition.lcdui.Font getFont(int elementNum)
Gets the application's preferred font for rendering the specified element of this Choice.

javax.microedition.lcdui.Image getImage(int elementNum)
Gets the Image part of the element referenced by elementNum.

int getSelectedFlags(boolean[] selectedArray_return)
Queries the state of a ChoiceGroup and returns the state of all elements in the boolean array selectedArray_return.

int getSelectedIndex()
Returns the index number of an element in the ChoiceGroup that is selected.

java.lang.String getString(int elementNum)
Gets the String part of the element referenced by elementNum.

void insert(int elementNum, String stringPart, Image imagePart)
Inserts an element into the ChoiceGroup just prior to the element specified.

boolean isSelected(int elementNum)
Gets a boolean value indicating whether this element is selected.

void set(int elementNum, String stringPart, Image imagePart)
Sets the String and Image parts of the element referenced by elementNum, replacing the previous contents of the element.

void setFitPolicy(int fitPolicy)
Sets the application's preferred policy for fitting Choice element contents to the available screen space.

void setFont(int elementNum, Font font)
Sets the application's preferred font for rendering the specified element of this Choice.

void setSelectedFlags(boolean[] selectedArray)
Attempts to set the selected state of every element in the ChoiceGroup.

void setSelectedIndex(int elementNum, boolean selected)
For ChoiceGroup objects of type MULTIPLE, this simply sets an individual element's selected state.

int size()
Returns the number of elements in the ChoiceGroup.

Methods inherited from class javax.microedition.lcdui.Item
addCommand, getLabel, getLayout, getMinimumHeight, getMinimumWidth, getPreferredHeight, getPreferredWidth, notifyStateChanged, removeCommand, setDefaultCommand, setItemCommandListener, setLabel, setLayout, setPreferredSize

Methods inherited from class Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from interface javax.microedition.lcdui.Choice
append, delete, deleteAll, getFitPolicy, getFont, getImage, getSelectedFlags, getSelectedIndex, getString, insert, isSelected, set, setFitPolicy, setFont, setSelectedFlags, setSelectedIndex, size

Constructors

ChoiceGroup

public ChoiceGroup(String label,  
int choiceType)

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)  
choiceType - EXCLUSIVE, MULTIPLE, or POPUP

Throws:
  IllegalArgumentException - if choiceType is not one of EXCLUSIVE, MULTIPLE, or POPUP

See Also: Choice.EXCLUSIVE, Choice.MULTIPLE, Choice.IMPLICIT, Choice.POPUP

ChoiceGroup

public ChoiceGroup(String label,  
int choiceType,  
String[] stringElements,  
Image[] imageElements)

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)  
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, Choice.MULTIPLE, Choice.IMPLICIT, Choice.POPUP

Methods
append

public int append(String stringPart, Image imagePart)

Appends an element to the ChoiceGroup.

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

public void delete(int elementNum)

Deletes the element referenced by elementNum.

Parameters:
- elementNum - the index of the element to be deleted

Throws:
- IndexOutOfBoundsException - if elementNum is invalid

deleteAll

public void deleteAll()

Deletes all elements from this ChoiceGroup.

getFitPolicy

public int getFitPolicy()

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 Choice.TEXT_WRAP_DEFAULT, Choice.TEXT_WRAP_ON, or Choice.TEXT_WRAP_OFF

See Also: ChoiceGroup.setFitPolicy(int)

Since: MIDP 2.0

g.getFont

public javax.microedition.lcdui.Font getFont(int elementNum)

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

**See Also:** ChoiceGroup.setFont(int, Font)

**Since:** MIDP 2.0

---

**getImage**

```java
public javax.microedition.lcdui.Image getImage(int elementNum)
```

Gets the Image part of the element referenced by elementNum.

**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:** ChoiceGroup.getString(int)

---

**getSelectedFlags**

```java
public int getSelectedFlags(boolean[] selectedArray_return)
```

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.

**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:** ChoiceGroup.setSelectedFlags(boolean[])

---

**getSelectedIndex**

```java
public int getSelectedIndex()
```

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.

**Returns:**
- index of selected element, or -1 if none

**See Also:** ChoiceGroup.setSelectedIndex(int, boolean)
**getString**

```java
public java.lang.String getString(int elementNum)
```

Gets the String part of the element referenced by `elementNum`.

**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:** `ChoiceGroup.getImage(int)`

---

**insert**

```java
public void insert(int elementNum, 
                   String stringPart, 
                   Image imagePart)
```

Inserts an element into the `ChoiceGroup` just prior to the element specified.

**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**

```java
public boolean isSelected(int elementNum)
```

Gets a boolean value indicating whether this element is selected.

**Parameters:**
- `elementNum` - the index of the element to be queried

**Returns:**
- selection state of the element

**Throws:**
- `IndexOutOfBoundsException` - if `elementNum` is invalid

---

**set**

```java
public void set(int elementNum, 
                 String stringPart, 
                 Image imagePart)
```

Sets the String and Image parts of the element referenced by `elementNum`, replacing the previous contents of the element.

**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

public void setFitPolicy(int fitPolicy)

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, Choice.TEXT_WRAP_ON, and Choice.TEXT_WRAP_OFF. 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

See Also: ChoiceGroup.getFitPolicy()

Since: MIDP 2.0

setFont

public void setFont(int elementNum, Font font)

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

See Also: ChoiceGroup.getFont(int)

Since: MIDP 2.0

setSelectedFlags

public void setSelectedFlags(boolean[] selectedArray)

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.

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: ChoiceGroup.getSelectedFlags(boolean[])
**setSelectedIndex**

```java
public void setSelectedIndex(int elementNum, boolean selected)
```

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.

**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:** `ChoiceGroup.getSelectedIndex()`

---

**size**

```java
public int size()
```

Returns the number of elements in the `ChoiceGroup`.

**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 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.

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, CANCEL, EXIT, HELP, ITEM, OK, SCREEN, and STOP.

**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:

```java
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

### Field Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACK</td>
<td>A navigation command that returns the user to the logically previous screen.</td>
<td>2</td>
</tr>
<tr>
<td>CANCEL</td>
<td>A command that is a standard negative answer to a dialog implemented by current screen.</td>
<td>3</td>
</tr>
<tr>
<td>EXIT</td>
<td>A command used for exiting from the application.</td>
<td>7</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>HELP</td>
<td>This command specifies a request for on-line help.</td>
<td>5</td>
</tr>
<tr>
<td>ITEM</td>
<td>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.</td>
<td>8</td>
</tr>
<tr>
<td>OK</td>
<td>A command that is a standard positive answer to a dialog implemented by current screen.</td>
<td>4</td>
</tr>
<tr>
<td>SCREEN</td>
<td>Specifies an application-defined command that pertains to the current screen.</td>
<td>1</td>
</tr>
<tr>
<td>STOP</td>
<td>A command that will stop some currently running process, operation, etc.</td>
<td>6</td>
</tr>
</tbody>
</table>

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public Command(String label, int commandType, int priority)</code></td>
<td>Creates a new command object with the given short <code>label</code>, type, and priority.</td>
<td>String label, int commandType, int priority</td>
</tr>
<tr>
<td><code>public Command(String shortLabel, String longLabel, int commandType, int priority)</code></td>
<td>Creates a new command object with the given labels, type, and priority.</td>
<td>String shortLabel, String longLabel, int commandType, int priority</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int getCommandType()</code></td>
<td>Gets the type of the command.</td>
<td></td>
</tr>
<tr>
<td><code>java.lang.String getLabel()</code></td>
<td>Gets the short label of the command.</td>
<td></td>
</tr>
<tr>
<td><code>java.lang.String getLongLabel()</code></td>
<td>Gets the long label of the command.</td>
<td></td>
</tr>
<tr>
<td><code>int getPriority()</code></td>
<td>Gets the priority of the command.</td>
<td></td>
</tr>
</tbody>
</table>

Methods inherited from class `Object`:
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### Fields

**BACK**

public static final int BACK
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` 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`.
Constant value: 2

See Also: `Command.CANCEL`, `Command.STOP`

### CANCEL

```java
public static final int CANCEL
```

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` 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`.
Constant value: 3

See Also: `Command.BACK`, `Command.STOP`

### EXIT

```java
public static final int EXIT
```

A command used for exiting from the application. When the user invokes this command, the implementation does not exit automatically. The application's `commandAction` will be called, and it should exit the application if it is appropriate to do so.

Value 7 is assigned to `EXIT`.
Constant value: 7

### HELP

```java
public static final int HELP
```

This command specifies a request for on-line help. No help information is shown automatically by the implementation. The `commandAction` provided by the application is responsible for showing the help information.

Value 5 is assigned to `HELP`.
Constant value: 5

### ITEM

```java
public static final int ITEM
```

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`.
Constant value: 8

### OK

```java
public static final int OK
```
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` 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`.
Constant value: 4

**See Also:** `Command.CANCEL`

---

**SCREEN**

```java
public static final int SCREEN
```

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 contrast 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`.
Constant value: 1

**See Also:** `Command.BACK, Command.CANCEL`

---

**STOP**

```java
public static final int STOP
```

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` 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`.
Constant value: 6

**See Also:** `Command.BACK, Command.CANCEL`

---

### Constructors

**Command**

```java
public Command(String label,
                   int commandType,
                   int priority)
```

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:** `javax.microedition.lcdui.Command(String, String, int, int)`
Command

public Command(String shortLabel,
               String longLabel,
               int commandType,
               int priority)

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

gGetCommandType

public int getCommandType()

Gets the type of the command.

Returns:
- the type of the Command

getLabel

public java.lang.String getLabel()

Gets the short label of the command.

Returns:
- the Command's short label

getLongLabel

public java.lang.String getLongLabel()

Gets the long label of the command.

Returns:
- the Command's long label, or null if the Command has no long label

Since: MIDP 2.0

ggetPriority

public int getPriority()

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)

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandAction</td>
<td>void commandAction(Command c, Displayable d)</td>
<td>Indicates that a command event has occurred on Displayable d.</td>
</tr>
</tbody>
</table>

Methods

commandAction

public void commandAction(Command c, Displayable d)

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) or is the implicit SELECT_COMMAND of List.
d - the Displayable on which this event has occurred
javax.microedition.lcdui

CustomItem

Declaration

public abstract class CustomItem extends Item

Object
+-javax.microedition.lcdui.Item
++-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() 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, getMinContentWidth, getPrefContentHeight, getPrefContentWidth, and sizeChanged 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, Item.getMinimumWidth, Item.getPreferredHeight, and Item.getPreferredWidth.

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 and the paint 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, getMinContentWidth, getPrefContentHeight, and getPrefContentWidth, 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 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 a 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.setCurrentItem(Displayable) 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() 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 \textit{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 \textit{internal} to a CustomItem by setting one or both of the \texttt{TRAVERSE_HORIZONTAL} or \texttt{TRAVERSE_VERTICAL} bits in the value returned by \texttt{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 \textit{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 \texttt{false} to the initial call to the \texttt{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 \texttt{false} to the initial \texttt{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 \texttt{traverse} 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 \texttt{paint} method is called.

The colors for foreground, background, highlighted foreground, highlighted background, border, and highlighted border should be retrieved from \texttt{Display.getColor(int)}. 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 \texttt{Font.getFont(int)}. This will allow them to match the fonts used by other items on the device for a consistent visual appearance.

Since: MIDP 2.0

| Field Summary | |
protected static final KEY_PRESS
Interaction mode bit indicating support for key pressed events.
Value: 4

protected static final KEY_RELEASE
Interaction mode bit indicating support for key released events.
Value: 8

protected static final KEY_REPEAT
Interaction mode bit indicating support for key repeated events.
Value: 16

protected static final NONE
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.
Value: 0

protected static final POINTER_DRAG
Interaction mode bit indicating support for point dragged events.
Value: 128

protected static final POINTER_PRESS
Interaction mode bit indicating support for point pressed events.
Value: 32

protected static final POINTER_RELEASE
Interaction mode bit indicating support for point released events.
Value: 64

protected static final TRAVERSE_HORIZONTAL
Interaction mode bit indicating support of horizontal traversal internal to the CustomItem.
Value: 1

protected static final TRAVERSE_VERTICAL
Interaction mode bit indicating support for vertical traversal internal to the CustomItem.
Value: 2

Fields inherited from class javax.microedition.lcdui.Item
BUTTON, HYPERLINK, LAYOUT_2, LAYOUT_BOTTOM, LAYOUT_CENTER, LAYOUT_DEFAULT, LAYOUT_EXPAND, LAYOUT_LEFT, LAYOUT_NEWLINE_AFTER, LAYOUT_NEWLINE_BEFORE, LAYOUT_RIGHT, LAYOUT_SHRINK, LAYOUT_TOP, LAYOUT_VCENTER, LAYOUT_VEXPAND, LAYOUT_VSHRINK, PLAIN

Constructor Summary

protected CustomItem(String label)
Superclass constructor, provided so that the CustomItem subclass can specify its label.

Method Summary

int getGameAction(int keyCode)
Gets the game action associated with the given key code of the device.
<table>
<thead>
<tr>
<th>Method Type</th>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>getInteractionModes()</td>
<td>Gets the available interaction modes.</td>
</tr>
<tr>
<td>abstract int</td>
<td>getMinContentHeight()</td>
<td>Implemented by the subclass to return the minimum height of the content area, in pixels.</td>
</tr>
<tr>
<td>abstract int</td>
<td>getMinContentWidth()</td>
<td>Implemented by the subclass to return the minimum width of the content area, in pixels.</td>
</tr>
<tr>
<td>abstract int</td>
<td>getPrefContentHeight(int width)</td>
<td>Implemented by the subclass to return the preferred height of the content area, in pixels.</td>
</tr>
<tr>
<td>abstract int</td>
<td>getPrefContentWidth(int height)</td>
<td>Implemented by the subclass to return the preferred width of the content area, in pixels.</td>
</tr>
<tr>
<td>void</td>
<td>hideNotify()</td>
<td>Called by the system to notify the item that it is now completely invisible, when it previously had been at least partially visible.</td>
</tr>
<tr>
<td>void</td>
<td>invalidate()</td>
<td>Signals that the CustomItem’s size and traversal location need to be updated.</td>
</tr>
<tr>
<td>void</td>
<td>keyPressed(int keyCode)</td>
<td>Called by the system when a key is pressed.</td>
</tr>
<tr>
<td>void</td>
<td>keyReleased(int keyCode)</td>
<td>Called by the system when a key is released.</td>
</tr>
<tr>
<td>void</td>
<td>keyRepeated(int keyCode)</td>
<td>Called by the system when a key is repeated.</td>
</tr>
<tr>
<td>abstract void</td>
<td>paint(Graphics g, int w, int h)</td>
<td>Implemented by the subclass to render the item within its container.</td>
</tr>
<tr>
<td>void</td>
<td>pointerDragged(int x, int y)</td>
<td>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.</td>
</tr>
<tr>
<td>void</td>
<td>pointerPressed(int x, int y)</td>
<td>Called by the system when a pointer down action (for example, a pen tap) has occurred within the item.</td>
</tr>
<tr>
<td>void</td>
<td>pointerReleased(int x, int y)</td>
<td>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.</td>
</tr>
<tr>
<td>void</td>
<td>repaint()</td>
<td>Called by subclass code to request that the item be repainted.</td>
</tr>
<tr>
<td>void</td>
<td>repaint(int x, int y, int w, int h)</td>
<td>Called by subclass code to request that the specified rectangular area of the item be repainted.</td>
</tr>
<tr>
<td>void</td>
<td>showNotify()</td>
<td>Called by the system to notify the item that it is now at least partially visible, when it previously had been completely invisible.</td>
</tr>
</tbody>
</table>
### Methods

**Methods inherited from class** [javax.microedition.lcdui.Item](https://developer.java.net/javadoc/j2me/javax/microedition/lcdui/Item.html)

- `addCommand, getLabel, getLayout, getMinimumHeight, getMinimumWidth, getPreferredSize, getPreferredHeight, getPreferredWidth, notifyStateChanged, removeCommand, setDefaultCommand, setItemCommandListener, setLabel, setLayout, setPreferredSize`

**Methods inherited from class** [Object](https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html)

- `equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

---

### Fields

**KEYPRESS**

*protected static final int* `KEYPRESS`

Interaction mode bit indicating support for key pressed events.

`KEYPRESS` has the value 4.

Constant value: 4

*See Also:* CustomItem.getInteractionModes(), CustomItem.keyPressed(int)

**KEYRELEASE**

*protected static final int* `KEYRELEASE`

Interaction mode bit indicating support for key released events.

`KEYRELEASE` has the value 8.

Constant value: 8

*See Also:* CustomItem.getInteractionModes(), CustomItem.keyReleased(int)

**KEYREPEAT**

*protected static final int* `KEYREPEAT`

Interaction mode bit indicating support for key repeated events.

`KEYREPEAT` has the value 0x10.

Constant value: 16

*See Also:* CustomItem.getInteractionModes(), CustomItem.keyRepeated(int)

**NONE**

*protected static final int* `NONE`
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.

Constant value: 0

See Also: CustomItem.traverse(int, int, int, int[])

---

**POINTER_DRAG**

protected static final int POINTER_DRAG

Interaction mode bit indicating support for point dragged events.

POINTER_DRAG has the value 0x80.

Constant value: 128

See Also: CustomItem.getInteractionModes(), CustomItem.pointerDragged(int, int)

---

**POINTER_PRESS**

protected static final int POINTER_PRESS

Interaction mode bit indicating support for point pressed events.

POINTER_PRESS has the value 0x20.

Constant value: 32

See Also: CustomItem.getInteractionModes(), CustomItem.pointerPressed(int, int)

---

**POINTER_RELEASE**

protected static final int POINTER_RELEASE

Interaction mode bit indicating support for point released events.

POINTER_RELEASE has the value 0x40.

Constant value: 64

See Also: CustomItem.getInteractionModes(), CustomItem.pointerReleased(int, int)

---

**TRAVERSE_HORIZONTAL**

protected static final int TRAVERSE_HORIZONTAL

Interaction mode bit indicating support of horizontal traversal internal to the CustomItem.

TRAVERSE_HORIZONTAL has the value 1.

Constant value: 1

See Also: CustomItem.getInteractionModes(), CustomItem.traverse(int, int, int, int[])

---

**TRAVERSE_VERTICAL**

protected static final int TRAVERSE_VERTICAL

Interaction mode bit indicating support for vertical traversal internal to the CustomItem.

TRAVERSE_VERTICAL has the value 2.

Constant value: 2

See Also: CustomItem.getInteractionModes(), CustomItem.traverse(int, int, int, int[])
CustomItem

protected CustomItem(String label)

Superclass constructor, provided so that the CustomItem subclass can specify its label.

Parameters:
  label - the CustomItem's label

Methods

g GameAction

public int getGameAction(int keyCode)

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

protected final int getInteractionModes()

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

protected abstract int getMinContentHeight()

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 and paint methods.

Returns:
  the minimum content height in pixels

getMinContentWidth

protected abstract int getMinContentWidth()

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 and paint methods.

Returns:
  the minimum content width in pixels
getPrefContentHeight

protected abstract int getPrefContentHeight(int width)

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 and paint 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

protected abstract int getPrefContentWidth(int height)

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 and paint 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

protected void hideNotify()

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

protected final void invalidate()
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`, `getMinContentWidth`, `getPrefContentHeight`, `getPrefContentWidth`, `sizeChanged`, or `traverse`. The `paint` 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**

```java
protected void keyPressed(int keyCode)
```

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:** `CustomItem.getInteractionModes()`

**keyReleased**

```java
protected void keyReleased(int keyCode)
```

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:** `CustomItem.getInteractionModes()`

**keyRepeated**

```java
protected void keyRepeated(int keyCode)
```

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:** `CustomItem.getInteractionModes()`

**paint**

```java
protected abstract void paint(Graphics g, int w, int h)
```

-200-
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();
- the stroke style is SOLID;

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

protected void pointerDragged(int \(x\),
                                  int \(y\))

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 \text{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: CustomItem.getInteractionModes()

pointerPressed

protected void pointerPressed(int \(x\),
                                  int \(y\))

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 \text{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: CustomItem.getInteractionModes()

pointerReleased

protected void pointerReleased(int \(x\),
                                  int \(y\)
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 y coordinate of the pointer up

**See Also:** `CustomItem.getInteractionModes()`

---

**repaint**

```java
protected final void repaint()
```

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**

```java
protected final void repaint(int x,
    int y,
    int w,
    int h)
```

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 a 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**

```java
protected void showNotify()
```

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**

```java
protected void sizeChanged(int w,
    int h)
```

---

-202-
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**

```java
protected boolean traverse(int dir,
        int viewportWidth,
        int viewportHeight,
        int[] visRect_inout)
```
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`, `Canvas.DOWN`, `Canvas.LEFT`, `Canvas.RIGHT`, or `NONE`.
- `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:** `CustomItem.getInteractionModes()`, `CustomItem.traverseOut()`, `CustomItem.TRAVERSE_HORIZONTAL`, `CustomItem.TRAVERSE_VERTICAL`

---

### `traverseOut`

```java
protected void traverseOut()
```

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.)

**See Also:** `CustomItem.getInteractionModes()`, `CustomItem.traverse(int, int, int, int[])`, `CustomItem.TRAVERSE_HORIZONTAL`, `CustomItem.TRAVERSE_VERTICAL`
java.microedition.lcdui

DateField

Declaration

public class DateField extends Item

Object

+--javax.microedition.lcdui.Item
   +--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

Field Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| DATE | Input mode for date information (day, month, year).
| Value: 1 |
| DATE_TIME | Input mode for date (day, month, year) and time (minutes, hours) information.
| Value: 3 |
| TIME | Input mode for time information (hours and minutes).
| Value: 2 |

Fields inherited from class java.microedition.lcdui.Item

BUTTON, HYPERLINK, LAYOUT_2, LAYOUT_BOTTOM, LAYOUT_CENTER, LAYOUT_DEFAULT, LAYOUT_EXPAND, LAYOUT_LEFT, LAYOUT_NEWLINE_AFTER, LAYOUT_NEWLINE_BEFORE, LAYOUT_RIGHT, LAYOUT_SHRINK, LAYOUT_TOP, LAYOUT_VCENTER, LAYOUT_VEXPAND, LAYOUT_VSHRINK, PLAIN

Constructor Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DateField(String label, int mode)</td>
<td>Creates a DateField object with the specified label and mode.</td>
</tr>
</tbody>
</table>
public DateField(String label, int mode, TimeZone timeZone)

Creates a date field in which calendar calculations are based on specific
TimeZone object and the default calendaring system for the current locale.

Method Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.util.Date</td>
<td>getDate()</td>
<td>Returns date value of this field.</td>
</tr>
<tr>
<td>int</td>
<td>getInputMode()</td>
<td>Gets input mode for this date field.</td>
</tr>
<tr>
<td>void</td>
<td>setDate(Date date)</td>
<td>Sets a new value for this field.</td>
</tr>
<tr>
<td>void</td>
<td>setInputMode(int mode)</td>
<td>Set input mode for this date field.</td>
</tr>
</tbody>
</table>

Methods inherited from class javax.microedition.lcdui.Item

addCommand, getLabel, getLayout, getMinimumHeight, getMinimumWidth,
genPreferredHeight, getPreferredWidth, notifyStateChanged, removeCommand,
setDefaultCommand, setItemCommandListener, setLabel, setLayout, setPreferredSize

Methods inherited from class Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Fields

**DATE**

public static final int DATE

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.
Constant value: 1

**DATE_TIME**

public static final int DATE_TIME

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.
Constant value: 3

**TIME**

public static final int TIME

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

public DateField(String label, int mode)

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

public DateField(String label, int mode, TimeZone timeZone)

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

public java.util.Date getDate()

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.

Returns:
- date object representing time or date depending on input mode

See Also: DateField.setDateTime(Date)

getInputMode

public int getInputMode()

Gets input mode for this date field. Valid input modes are DATE, TIME and DATE_TIME.

Returns:
- input mode of this field

See Also: DateField.setInputMode(int)
setDate

public void setDate(Date date)

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: DateField.getDate()

setInputMode

public void setInputMode(int mode)

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: DateField.getInputMode()
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 objects and are thus defined there instead of here.

There is exactly one instance of Display per MIDlet and the application can get a reference to that instance by calling the getDisplay() 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() 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 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() method for setting the current Displayable and a getCurrent() 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 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(). In the case of Canvas, the showNotify() and hideNotify() 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()` 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

### Field Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Summary</th>
</tr>
</thead>
</table>
| public static final | ALERT
  Image type for Alert image. Value: 3 |
| public static final | CHOICE_GROUP_ELEMENT
  Image type for ChoiceGroup element image. Value: 2 |
| public static final | COLOR_BACKGROUND
  A color specifier for use with `getColor`. Value: 0 |
| public static final | COLOR_BORDER
  A color specifier for use with `getColor`. Value: 4 |
| public static final | COLOR_FOREGROUND
  A color specifier for use with `getColor`. Value: 1 |
public static final COLOR_HIGHLIGHTED_BACKGROUND
   A color specifier for use with getColor.
   Value: 2

public static final COLOR_HIGHLIGHTED_BORDER
   A color specifier for use with getColor.
   Value: 5

public static final COLOR_HIGHLIGHTED_FOREGROUND
   A color specifier for use with getColor.
   Value: 3

public static final LIST_ELEMENT
   Image type for List element image.
   Value: 1

### Method Summary

**void** callSerially(Runnable r)
Causes the Runnable object r to have its run() method called later, serialized with the event stream, soon after completion of the repaint cycle.

**boolean** flashBacklight(int duration)
Requests a flashing effect for the device's backlight.

**int** getBestImageHeight(int imageType)
Returns the best image height for a given image type.

**int** getBestImageWidth(int imageType)
Returns the best image width for a given image type.

**int** getBorderStyle(boolean highlighted)
Returns the stroke style used for border drawing depending on the state of the component (highlighted/non-highlighted).

**int** getColor(int colorSpecifier)
Returns one of the colors from the high level user interface color scheme, in the form 0x00RRGGBB based on the colorSpecifier passed in.

**javax.microedition.lcdui.Displayable** getCurrent()
Gets the current Displayable object for this MIDlet.

**static javax.microedition.lcdui.Display** getDisplay(MIDlet m)
Gets the Display object that is unique to this MIDlet.

**boolean** isColor()
Gets information about color support of the device.

**int** numAlphaLevels()
Gets the number of alpha transparency levels supported by this implementation.

**int** numColors()
Gets the number of colors (if isColor() is true) or graylevels (if isColor() is false) that can be represented on the device.

**void** setCurrent(Alert alert, Displayable nextDisplayable)
Requests that this Alert be made current, and that nextDisplayable be made current after the Alert is dismissed.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| ```java
void setCurrent(Displayable nextDisplayable)
``` | Requests that a different Displayable object be made visible on the display. |
| ```java
void setCurrentItem(Item item)
``` | 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. |
| ```java
boolean vibrate(int duration)
``` | Requests operation of the device's vibrator. |

**Methods inherited from class `Object`**

`equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

**Fields**

**ALERT**

```java
public static final int ALERT
```

Image type for Alert image.

The value of `ALERT` is 3.
Constant value: **3**

See Also: `Display.getBestImageWidth(int)`, `Display.getBestImageHeight(int)`
Since: MIDP 2.0

**CHOICE_GROUP_ELEMENT**

```java
public static final int CHOICE_GROUP_ELEMENT
```

Image type for ChoiceGroup element image.

The value of `CHOICE_GROUP_ELEMENT` is 2.
Constant value: **2**

See Also: `Display.getBestImageWidth(int)`, `Display.getBestImageHeight(int)`
Since: MIDP 2.0

**COLOR_BACKGROUND**

```java
public static final int COLOR_BACKGROUND
```

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**.
Constant value: **0**

See Also: `Display.getColor(int)`
Since: MIDP 2.0

**COLOR_BORDER**

```java
public static final int COLOR_BORDER
```
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.
Constant value: 4

See Also: `Display.getColor(int)`
Since: MIDP 2.0

**COLOR_FOREGROUND**

```java
public static final int COLOR_FOREGROUND
```
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 contrast with background color.

`COLOR_FOREGROUND` has the value 1.
Constant value: 1

See Also: `Display.getColor(int)`
Since: MIDP 2.0

**COLOR_HIGHLIGHTED_BACKGROUND**

```java
public static final int COLOR_HIGHLIGHTED_BACKGROUND
```
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 contrast with the highlighted foreground.

`COLOR_HIGHLIGHTED_BACKGROUND` has the value 2.
Constant value: 2

See Also: `Display.getColor(int)`
Since: MIDP 2.0

**COLOR_HIGHLIGHTED_BORDER**

```java
public static final int COLOR_HIGHLIGHTED_BORDER
```
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.
Constant value: 5

See Also: `Display.getColor(int)`
Since: MIDP 2.0

**COLOR_HIGHLIGHTED_FOREGROUND**

```java
public static final int COLOR_HIGHLIGHTED_FOREGROUND
```
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 contrast with the highlighted background.

`COLOR_HIGHLIGHTED_FOREGROUND` has the value 3.
Constant value: 3
LIST_ELEMENT

public static final int LIST_ELEMENT

    Image type for List element image.
    The value of LIST_ELEMENT is 1.
    Constant value: 1

See Also: Display.getColor(int), Display.getBestImageWidth(int), Display.getBestImageHeight(int)
Since: MIDP 2.0

Methods

callSerially

public void callSerially(Runnable r)
Causes the Runnable object \( r \) to have its \( \text{run()} \) 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 \( \text{r.run()} \) will be serialized along with the event calls into the application. The \( \text{run()} \) method will be called exactly once for each call to \( \text{callSerially()} \). Calls to \( \text{run()} \) will occur in the order in which they were requested by calls to \( \text{callSerially()} \).

If the current Displayable is a Canvas that has a repaint pending at the time of a call to \( \text{callSerially()} \), the \( \text{paint()} \) method of the Canvas will be called and will return, and a buffer switch will occur (if double buffering is in effect), before the \( \text{run()} \) 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 \( \text{callSerially()} \), the \( \text{paint()} \) methods of the CustomItems will be called and will return before the \( \text{run()} \) method of the Runnable is called. Calls to the \( \text{run()} \) 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 \( \text{callSerially()} \) method may be called from any thread. The call to the \( \text{run()} \) method will occur independently of the call to \( \text{callSerially()} \). In particular, \( \text{callSerially()} \) will never block waiting for \( \text{r.run()} \) to return.

As with other callbacks, the call to \( \text{r.run()} \) must return quickly. If it is necessary to perform a long-running operation, it may be initiated from within the \( \text{run()} \) method. The operation itself should be performed within another thread, allowing \( \text{run()} \) to return.

The \( \text{callSerially()} \) 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 \( \text{repaint()} \). The application must then wait until the frame is actually displayed, after which the setup for the next frame may occur. The call to \( \text{run()} \) notifies the application that the previous frame has finished painting. The example below shows \( \text{callSerially()} \) being used for this purpose.

```java
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);
        }
    }
}
```

Parameters:
Since: MIDP 2.0

### flashBacklight

```java
public boolean flashBacklight(int duration)
```

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 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

```java
public int getBestImageHeight(int imageType)
```

Returns the best image height for a given image type. The image type must be one of `Display.LIST_ELEMENT`, `Display.CHOICE_GROUP_ELEMENT`, or `Display.ALERT`.

**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

```java
public int getBestImageWidth(int imageType)
```

Returns the best image width for a given image type. The image type must be one of `Display.LIST_ELEMENT`, `Display.CHOICE_GROUP_ELEMENT`, or `Display.ALERT`.

**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

public int getBorderStyle(boolean highlighted)

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 or Graphics.SOLID

Since: MIDP 2.0

getColor

public int getColor(int colorSpecifier)

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 Display.COLOR_BACKGROUND, Display.COLOR_FOREGROUND, Display.COLOR_HIGHLIGHTED_BACKGROUND, Display.COLOR_HIGHLIGHTED_FOREGROUND, Display.COLOR_BORDER, or Display.COLOR_HIGHLIGHTED_BORDER

Returns:
color in the form of 0x00RRGGBB

Throws:
IllegalArgumentException - if colorSpecifier is not a valid color specifier

Since: MIDP 2.0

gGetCurrent

public javax.microedition.lcdui.Displayable getCurrent()

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() 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: Display.setCurrent(Displayable)

gDisplay

public static javax.microedition.lcdui.Display getDisplay(MIDlet m)

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
isColor

public boolean isColor()

Gets information about color support of the device.

Returns:
    true if the display supports color, false otherwise

numAlphaLevels

public int numAlphaLevels()

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

public int numColors()

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

public void setCurrent(Alert alert, Displayable nextDisplayable)

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).

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, Display.getCurrent()
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, 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, Displayable) method. If the application calls setCurrent() while an Alert is current, the Alert is removed from the display and any timer it may have set is cancelled.

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:** Display.getCurrent()
**setCurrentItem**

```java
public void setCurrentItem(Item item)
```

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)` 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` - if the item is not owned by a container
- `IllegalStateException` - if the item is owned by an Alert
- `NullPointerException` - if `item` is null

**Since:** MIDP 2.0

**vibrate**

```java
public boolean vibrate(int duration)
```

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:
javax.microedition.lcdui.Canvas, javax.microedition.lcdui.Screen

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

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>addCommand(Command cmd) Adds a command to the Displayable.</td>
</tr>
<tr>
<td>int</td>
<td>getHeight() Gets the height in pixels of the displayable area available to the application.</td>
</tr>
<tr>
<td>javax.microedition.lcdui.Ticker</td>
<td>getTicker() Gets the ticker used by this Displayable.</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getTitle() Gets the title of the Displayable.</td>
</tr>
<tr>
<td>int</td>
<td>getWidth() Gets the width in pixels of the displayable area available to the application.</td>
</tr>
<tr>
<td>boolean</td>
<td>isShown() Checks if the Displayable is actually visible on the display.</td>
</tr>
<tr>
<td>void</td>
<td>removeCommand(Command cmd) Removes a command from the Displayable.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>void setCommandListener(T)</td>
<td>Sets a listener for Commands to this Displayable, replacing any previous CommandListener.</td>
</tr>
<tr>
<td>void setTicker(T)</td>
<td>Sets a ticker for use with this Displayable, replacing any previous ticker.</td>
</tr>
<tr>
<td>void setTitle(String)</td>
<td>Sets the title of the Displayable.</td>
</tr>
<tr>
<td>void sizeChanged(int, int)</td>
<td>The implementation calls this method when the available area of the Displayable has been changed.</td>
</tr>
</tbody>
</table>

**Methods inherited from class Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

## Methods

### addCommand

**public void addCommand(Command cmd)**

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

**public int getHeight()**

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

**public javax.microedition.lcdui.Ticker getTicker()**

Gets the ticker used by this Displayable.

**Returns:**
- ticker object used, or null if no ticker is present

**See Also:** Displayable.setTicker(Ticker)

**Since:** MIDP 2.0
getTitle

public java.lang.String getTitle()

Gets the title of the Displayable. Returns null if there is no title.

Returns:
the title of the instance, or null if no title

See Also: Displayable.setTitle(String)
Since: MIDP 2.0

getWidth

public int getWidth()

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

public boolean isShown()

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
Since: MIDP 2.0

removeCommand

public void removeCommand(Command cmd)

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

public void setCommandListener(CommandListener l)

Sets a listener for Commands 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

public void setTicker(Ticker ticker)
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().

Parameters:

ticker - the ticker object used on this screen

See Also: Displayable.getTicker()
Since: MIDP 2.0

setTitle

public void setTitle(String s)

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().

Parameters:

s - the new title, or null for no title

See Also: Displayable.getTitle()
Since: MIDP 2.0

sizeChanged

protected void sizeChanged(int w, int h)

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

Field Summary

<table>
<thead>
<tr>
<th>public static final</th>
<th>FACE_MONOSPACED</th>
<th>The &quot;monospace&quot; font face.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Value: 32</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>FACE_PROPORTIONAL</th>
<th>The &quot;proportional&quot; font face.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Value: 64</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>FACE_SYSTEM</th>
<th>The &quot;system&quot; font face.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Value: 0</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>public static final</td>
<td>FONT_INPUT_TEXT&lt;br&gt;Font specifier used by the implementation to draw text input by a user.&lt;br&gt;Value: 1</td>
<td></td>
</tr>
<tr>
<td>public static final</td>
<td>FONT_STATIC_TEXT&lt;br&gt;Default font specifier used to draw Item and Screen contents.&lt;br&gt;Value: 0</td>
<td></td>
</tr>
<tr>
<td>public static final</td>
<td>SIZE_LARGE&lt;br&gt;The &quot;large&quot; system-dependent font size.&lt;br&gt;Value: 16</td>
<td></td>
</tr>
<tr>
<td>public static final</td>
<td>SIZE_MEDIUM&lt;br&gt;The &quot;medium&quot; system-dependent font size.&lt;br&gt;Value: 0</td>
<td></td>
</tr>
<tr>
<td>public static final</td>
<td>SIZE_SMALL&lt;br&gt;The &quot;small&quot; system-dependent font size.&lt;br&gt;Value: 8</td>
<td></td>
</tr>
<tr>
<td>public static final</td>
<td>STYLE_BOLD&lt;br&gt;The bold style constant.&lt;br&gt;Value: 1</td>
<td></td>
</tr>
<tr>
<td>public static final</td>
<td>STYLE_ITALIC&lt;br&gt;The italicized style constant.&lt;br&gt;Value: 2</td>
<td></td>
</tr>
<tr>
<td>public static final</td>
<td>STYLE_PLAIN&lt;br&gt;The plain style constant.&lt;br&gt;Value: 0</td>
<td></td>
</tr>
<tr>
<td>public static final</td>
<td>STYLE_UNDERLINED&lt;br&gt;The underlined style constant.&lt;br&gt;Value: 4</td>
<td></td>
</tr>
</tbody>
</table>

**Method Summary**

- `public int charsWidth(char[] ch, int offset, int length)`<br>Returns the advance width of the characters in `ch`, starting at the specified offset and for the specified number of characters (length).

- `public int charWidth(char ch)`<br>Gets the advance width of the specified character in this Font.

- `public int getBaselinePosition()`<br>Gets the distance in pixels from the top of the text to the text's baseline.

- `public static javax.microedition.lcdui.Font getDefaultFont()`<br>Gets the default font of the system.

- `public int getFace()`<br>Gets the face of the font.

- `public static javax.microedition.lcdui.Font getFont(int fontSpecifier)`<br>Gets the Font used by the high level user interface for the `fontSpecifier` passed in.
**getFont(int face, int style, int size)**

Obtains an object representing a font having the specified face, style, and size.

**getHeigh(t)**

Gets the standard height of a line of text in this font.

**getSize()**

Gets the size of the font.

**getStyle()**

Gets the style of the font.

**isBold()**

Returns `true` if the font is bold.

**isItalic()**

Returns `true` if the font is italic.

**isPlain()**

Returns `true` if the font is plain.

**isUnderlined()**

Returns `true` if the font is underlined.

**stringWidth(String str)**

Gets the total advance width for showing the specified `String` in this `Font`.

**substringWidth(String str, int offset, int len)**

Gets the total advance width for showing the specified substring in this `Font`.

**Methods inherited from class Object**

`equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`.

**Fields**

**FACE_MONOSPACE**

`public static final int FACE_MONOSPACE`

The "monospace" font face.

Value 32 is assigned to `FACE_MONOSPACE`.

Constant value: 32

**FACE_PROPORTIONAL**

`public static final int FACE_PROPORTIONAL`

The "proportional" font face.

Value 64 is assigned to `FACE_PROPORTIONAL`.

Constant value: 64

**FACE_SYSTEM**

`public static final int FACE_SYSTEM`
The "system" font face.

Value 0 is assigned to FACE_SYSTEM.
Constant value: 0

**FONT_INPUT_TEXT**

```java
public static final int FONT_INPUT_TEXT = ...
```

Font specifier used by the implementation to draw text input by a user. FONT_INPUT_TEXT has the value 1.
Constant value: 1

See Also: Font.getFont(int)
Since: MIDP 2.0

**FONT_STATIC_TEXT**

```java
public static final int FONT_STATIC_TEXT = ...
```

Default font specifier used to draw Item and Screen contents. FONT_STATIC_TEXT has the value 0.
Constant value: 0

See Also: Font.getFont(int)
Since: MIDP 2.0

**SIZE_LARGE**

```java
public static final int SIZE_LARGE = ...
```

The "large" system-dependent font size.
Value 16 is assigned to SIZE_LARGE.
Constant value: 16

**SIZE_MEDIUM**

```java
public static final int SIZE_MEDIUM = ...
```

The "medium" system-dependent font size.
Value 0 is assigned to STYLE_MEDIUM.
Constant value: 0

**SIZE_SMALL**

```java
public static final int SIZE_SMALL = ...
```

The "small" system-dependent font size.
Value 8 is assigned to STYLE_SMALL.
Constant value: 8

**STYLE_BOLD**

```java
public static final int STYLE_BOLD = ...
```

The bold style constant. This may be combined with the other style constants for mixed styles.
Value 1 is assigned to STYLE_BOLD.
Constant value: 1
**STYLE_ITALIC**

public static final int **STYLE_ITALIC**

The italicized style constant. This may be combined with the other style constants for mixed styles.

Value 2 is assigned to **STYLE_ITALIC**.
Constant value: 2

**STYLE_PLAIN**

public static final int **STYLE_PLAIN**

The plain style constant. This may be combined with the other style constants for mixed styles.

Value 0 is assigned to **STYLE_PLAIN**.
Constant value: 0

**STYLE_UNDERLINED**

public static final int **STYLE_UNDERLINED**

The underlined style constant. This may be combined with the other style constants for mixed styles.

Value 4 is assigned to **STYLE_UNDERLINED**.
Constant value: 4

## Methods

**charsWidth**

public int **charsWidth**(char[] ch,  
    int offset,  
    int length)

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 \((offset + length) \leq ch.length\).

**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**

public int **charWidth**(char ch)

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:**
getBaselinePosition

public int getBaselinePosition()

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

getDefaultValue

public static javax.microedition.lcdui.Font getDefaultFont()

Gets the default font of the system.

Returns:
the default font

getFont

public static javax.microedition.lcdui.Font getFont(int fontSpecifier)

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

public static javax.microedition.lcdui.Font getFont(int face, int style, int size)

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_MONOSPACESPACE, 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:
java.lang.IllegalArgumentException - if face, style, or size are not legal values

getHeight
public int getHeight()

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
public int getSize()

Gets the size of the font.

Returns:
one of SIZE_SMALL, SIZE_MEDIUM, SIZE_LARGE

getStyle
public int getStyle()

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: Font.isPlain(), Font.isBold(), Font.isItalic()

isBold
public boolean isBold()

Returns true if the font is bold.

Returns:
true if font is bold
See Also: Font.getStyle()

isItalic
public boolean isItalic()

Returns true if the font is italic.

Returns:
true if font is italic
See Also: Font.getStyle()

isPlain
public boolean isPlain()

Returns true if the font is plain.
See Also: `Font.getStyle()`

### isUnderlined

**public boolean isUnderlined()**

Returns true if the font is underlined.

**Returns:**
true if font is underlined

See Also: `Font.getStyle()`

### stringWidth

**public int stringWidth(String str)**

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

**public int substringWidth(String str, int offset, int len)**

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 Screen

Object

----------javax.microedition.lcdui.Displayable
----------javax.microedition.lcdui.Screen

+---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 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 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 that 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.getPreferredSize 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

-237-
For all cases where text is wrapped, line breaks must occur at each newline character (\n). 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() method of the listener declared to the Form with the setItemStateListener() method.

As with other Displayable objects, a Form can declare commands and declare a command listener with the commandListener() method. CommandListener objects are distinct from ItemStateListener 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

### Constructor Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| public | Form(String title)  
|        | Creates a new, empty Form. |
| public | Form(String title, Item[] items)  
|        | Creates a new Form with the specified contents. |

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| int    | append(Image img)  
|        | Adds an item consisting of one Image to the Form. |
| int    | append(Item item)  
|        | Adds an Item into the Form. |
| int    | append(String str)  
|        | Adds an item consisting of one String to the Form. |
| void   | delete(int itemNum)  
|        | Deletes the Item referenced by itemNum. |
| void   | deleteAll()  
|        | Deletes all the items from this Form, leaving it with zero items. |
| javax.microedition.lcdui.Item | get(int itemNum)  
|        | Gets the item at given position. |
int getHeight()
Returns the height in pixels of the displayable area available for items.

int getWidth()
Returns the width in pixels of the displayable area available for items.

void insert(int itemNum, Item item)
Inserts an item into the Form just prior to the item specified.

void set(int itemNum, Item item)
Sets the item referenced by itemNum to the specified item, replacing the previous item.

void setItemStateListener(ItemStateListener iListener)
Sets the ItemStateListener for the Form, replacing any previous ItemStateListener.

int size()
Gets the number of items in the Form.

Methods inherited from class javax.microedition.lcdui.Displayable

addCommand, getHeight, getTicker, getTitle, getWidth, isShown, removeCommand, setCommandListener, setTicker, setTitle, sizeChanged

Methods inherited from class Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

Form

public Form(String title)

Creates a new, empty Form.

Parameters:
title - the Form's title, or null for no title

Form

public Form(String title, Item[] items)

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 - if one of the items is already owned by another container
NullPointerException - if an element of the items array is null

Methods
**append**

public int append(Image img)

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**

public int append(Item item)

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 to be added.

**Returns:**

the assigned index of the Item

**Throws:**

IllegalStateException - if the item is already owned by a container

NullPointerException - if item is null

---

**append**

public int append(String str)

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**

public void delete(int itemNum)

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**

```java
public void deleteAll()
```

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**

```java
public javax.microedition.lcdui.Item get(int itemNum)
```

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**

```java
public int getHeight()
```

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.

**Returns:**
the height of the displayable area of the `Form` in pixels

**Since:** MIDP 2.0

---

**getWidth**

```java
public int getWidth()
```

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.

**Returns:**
the width of the `Form` in pixels

**Since:** MIDP 2.0

---

**insert**

```java
public void insert(int itemNum,
                   Item item)
```

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()-1]`, 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)`.

The semantics are otherwise identical to `append(Item)`.

**Parameters:**
- `itemNum` - the index where insertion is to occur
- `item` - the item to be inserted
throws:
   IndexOutOfBoundsException - if itemNum is invalid
   IllegalStateException - if the item is already owned by a container
   NullPointerException - if item is null

set

public void set(int itemNum,
               Item item)

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 - if the item is already owned by a container
   NullPointerException - if item is null

setItemStateListener

public void setItemStateListener(ItemStateListener iListener)

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

public int size()

Gets the number of items in the Form.

Returns:
   the number of items
**Gauge**

**Declaration**

```java
class Gauge extends Item
```

**Description**

Gauge 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 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` 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

### Field Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final CONTINUOUS_IDLE</td>
<td>The value representing the continuous-idle state of a non-interactive <code>Gauge</code> with indefinite range. Value: 0</td>
</tr>
<tr>
<td>public static final CONTINUOUS_RUNNING</td>
<td>The value representing the continuous-running state of a non-interactive <code>Gauge</code> with indefinite range. Value: 2</td>
</tr>
<tr>
<td>public static final INCREMENTAL_IDLE</td>
<td>The value representing the incremental-idle state of a non-interactive <code>Gauge</code> with indefinite range. Value: 1</td>
</tr>
<tr>
<td>public static final INCREMENTAL_UPDATING</td>
<td>The value representing the incremental-updating state of a non-interactive <code>Gauge</code> with indefinite range. Value: 3</td>
</tr>
<tr>
<td>public static final INDEFINITE</td>
<td>A special value used for the maximum value in order to indicate that the <code>Gauge</code> has indefinite range. Value: -1</td>
</tr>
</tbody>
</table>

Fields inherited from class [javax.microedition.lcdui.Item](https://docs.oracle.com/javase/7/docs/api/javax/microedition/lcdui/Item.html)
**Constructor Summary**

<table>
<thead>
<tr>
<th>public Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gauge(String label, boolean interactive, int maxValue, int initialValue)</td>
<td>Creates a new Gauge object with the given label, in interactive or non-interactive mode, with the given maximum and initial values.</td>
</tr>
</tbody>
</table>

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int getMaxValue()</td>
<td>Gets the maximum value of this Gauge object.</td>
</tr>
<tr>
<td>int getValue()</td>
<td>Gets the current value of this Gauge object.</td>
</tr>
<tr>
<td>boolean isInteractive()</td>
<td>Tells whether the user is allowed to change the value of the Gauge.</td>
</tr>
<tr>
<td>void setMaxValue(int maxValue)</td>
<td>Sets the maximum value of this Gauge object.</td>
</tr>
<tr>
<td>void setValue(int value)</td>
<td>Sets the current value of this Gauge object.</td>
</tr>
</tbody>
</table>

Methods inherited from class `javax.microedition.lcdui.Item`:

- addCommand, getLabel, getLayout, getMinimumHeight, getMinimumWidth, getPreferredSize, getPreferredHeight, getPreferredWidth, notifyStateChanged, removeCommand, setDefaultCommand, setItemCommandListener, setLabel, setLayout, setPreferredSize

Methods inherited from class `Object`:

- equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Fields**

**CONTINUOUS_IDLE**

public static final int CONTINUOUS_IDLE

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.
Constant value: 0
Since: MIDP 2.0

**CONTINUOUS_RUNNING**

public static final int CONTINUOUS_RUNNING
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.
Constant value: 2

Since: MIDP 2.0

---

**INCREMENTAL_IDLE**

```java
public static final int INCREMENTAL_IDLE
```

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.
Constant value: 1

Since: MIDP 2.0

---

**INCREMENTAL_UPDATING**

```java
public static final int INCREMENTAL_UPDATING
```

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.
Constant value: 3

Since: MIDP 2.0

---

**INDEFINITE**

```java
public static final int INDEFINITE
```

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.
Constant value: -1

Since: MIDP 2.0

---

### Constructors

```
Gauge
```

```java
public Gauge(String label,
             boolean interactive,
             int maxValue,
             int initialValue)
```
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:** `Gauge.INDEFINITE`, `Gauge.CONTINUOUS_IDLE`, `Gauge.INCREMENTAL_IDLE`, `Gauge.CONTINUOUS_RUNNING`, `Gauge.INCREMENTAL_UPDATING`

### Methods

#### getMaxValue

```java
public int getMaxValue()
```

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:** `Gauge.INDEFINITE`, `Gauge.setMaxValue(int)`

#### getValue

```java
public int getValue()
```

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:** `Gauge.CONTINUOUS_IDLE`, `Gauge.INCREMENTAL_IDLE`, `Gauge.CONTINUOUS_RUNNING`, `Gauge.INCREMENTAL_UPDATING`, `Gauge.setValue(int)}`
isInteractive

public boolean isInteractive()

Tells whether the user is allowed to change the value of the Gauge.

Returns:
a boolean indicating whether the Gauge is interactive

setMaxValue

public void setMaxValue(int maxValue)

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: Gauge.INDEFINITE,Gauge.getMaxValue()

setValue

public void setValue(int value)

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: Gauge.CONTINUOUS_IDLE,Gauge.INCREMENTAL_IDLE,Gauge.CONTINUOUS_RUNNING,Gauge.INCREMENTAL_UPDATING,Gauge.getValue()
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 class for obtaining device characteristics, such as whether color is available and how many distinct gray levels are available. Applications may also use getDisplayColor() 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() and drawRegion() 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 is that compositing any pixel with a fully opaque destination pixel always results in a fully opaque 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() 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() 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()`, `drawChars()`, `drawString()`, and `drawSubString()` 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()` 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()`, do not modify the clip. The methods `getClipX()`, `getClipY()`, `getClipWidth()` and `getClipHeight()` 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()`. 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,

```java
    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 `getStringWidth()` or `getCharWidth()` 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:
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 \((\text{LEFT}, \text{HCENTER}, \text{RIGHT})\) combined with one of the vertical constants \((\text{TOP}, \text{BASELINE}, \text{BOTTOM})\) 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 \(\text{TOP} \mid \text{LEFT}\).

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 \(\text{VCENTER}\) 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:

\[
\begin{align*}
g\text{.drawString}(\text{str}, & \ x, \ y, \ \text{TOP}\mid\text{LEFT}); \\
g\text{.drawString}(\text{str}, & \ x + f\text{.stringWidth}(\text{str})/2, \ y, \ \text{TOP}\mid\text{HCENTER}); \\
g\text{.drawString}(\text{str}, & \ x + f\text{.stringWidth}(\text{str}), \ y, \ \text{TOP}\mid\text{RIGHT}); \\
g\text{.drawString}(\text{str}, & \ \ y + f\text{.getBaselinePosition}(), \ \text{BASELINE}\mid\text{LEFT}); \\
g\text{.drawString}(\text{str}, & \ x + f\text{.stringWidth}(\text{str})/2, \ y + f\text{.getBaselinePosition}(), \ \text{BASELINE}\mid\text{HCENTER}); \\
g\text{.drawString}(\text{str}, & \ x + f\text{.stringWidth}(\text{str}), \ y + f\text{.getBaselinePosition}(), \ \text{BASELINE}\mid\text{RIGHT}); \\
drawString(\text{str}, & \ x, \\
& \ y + f\text{.getHeight}(), \ \text{BOTTOM}\mid\text{LEFT}); \\
drawString(\text{str}, & \ x + f\text{.stringWidth}(\text{str})/2, \\
& \ y + f\text{.getHeight}(), \ \text{BOTTOM}\mid\text{HCENTER}); \\
drawString(\text{str}, & \ x + f\text{.stringWidth}(\text{str}), \\
& \ y + f\text{.getHeight}(), \ \text{BOTTOM}\mid\text{RIGHT}); \\
\end{align*}
\]

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 \textit{stringWidth()} and \textit{getHeight()} calls of class \texttt{Font}. For example, given the following code:

\[
\begin{align*}
// (5) \\
g\text{.drawString}(\text{string1+string2}, & \ x, \ y, \ \text{TOP}\mid\text{LEFT}); \\
// (6) \\
g\text{.drawString}(\text{string1}, & \ x, \ y, \ \text{TOP}\mid\text{LEFT}); \\
g\text{.drawString}(\text{string2}, & \ x + f\text{.stringWidth}(\text{string1}), \ y, \ \text{TOP}\mid\text{LEFT}); \\
\end{align*}
\]

Code fragments (5) and (6) behave similarly if not identically. This occurs because \textit{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 \textit{Y-position} of subsequent lines. For example:
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

Since: MIDP 1.0

### Field Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Constant</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final</td>
<td>BASELINE</td>
<td>Constant for positioning the anchor point at the baseline of text.</td>
<td>64</td>
</tr>
<tr>
<td>public static final</td>
<td>BOTTOM</td>
<td>Constant for positioning the anchor point of text and images below the text or image.</td>
<td>32</td>
</tr>
<tr>
<td>public static final</td>
<td>DOTTED</td>
<td>Constant for the DOTTED stroke style.</td>
<td>1</td>
</tr>
<tr>
<td>public static final</td>
<td>HCENTER</td>
<td>Constant for centering text and images horizontally around the anchor point</td>
<td>1</td>
</tr>
<tr>
<td>public static final</td>
<td>LEFT</td>
<td>Constant for positioning the anchor point of text and images to the left of the text or image.</td>
<td>4</td>
</tr>
<tr>
<td>public static final</td>
<td>RIGHT</td>
<td>Constant for positioning the anchor point of text and images to the right of the text or image.</td>
<td>8</td>
</tr>
</tbody>
</table>
public static final SOLID Constant for the SOLID stroke style.
Value: 0

public static final TOP Constant for positioning the anchor point of text and images above the text or image.
Value: 16

public static final VCENTER Constant for centering images vertically around the anchor point.
Value: 2

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>void clipRect(int x, int y, int width, int height)</td>
</tr>
</tbody>
</table>
Intersects the current clip with the specified rectangle.

| void copyArea(int x_src, int y_src, int width, int height, int x_dest, int y_dest, int anchor) |
Copies the contents of a rectangular area \((x_{\text{src}}, y_{\text{src}}, \text{width}, \text{height})\) to a destination area, whose anchor point identified by anchor is located at \((x_{\text{dest}}, y_{\text{dest}})\).

| void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle) |
Draws the outline of a circular or elliptical arc covering the specified rectangle, using the current color and stroke style.

| void drawChar(char character, int x, int y, int anchor) |
Draws the specified character using the current font and color.

| void drawChars(char[] data, int offset, int length, int x, int y, int anchor) |
Draws the specified characters using the current font and color.

| void drawImage(Image img, int x, int y, int anchor) |
Draws the specified image by using the anchor point.

| void drawLine(int x1, int y1, int x2, int y2) |
Draws a line between the coordinates \((x_{1}, y_{1})\) and \((x_{2}, y_{2})\) using the current color and stroke style.

| void drawRect(int x, int y, int width, int height) |
Draws the outline of the specified rectangle using the current color and stroke style.

| void drawRegion(Image src, int x_src, int y_src, int width, int height, int transform, int x_dest, int y_dest, int anchor) |
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.

| void drawRGB(int[] rgbData, int offset, int scanlength, int x, int y, int width, int height, boolean processAlpha) |
Renders a series of device-independent RGB+transparency values in a specified region.

| void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight) |
Draws the outline of the specified rounded corner rectangle using the current color and stroke style.
<p>| void | <code>drawString(String str, int x, int y, int anchor)</code> |
|      | Draws the specified String using the current font and color. |
| void | <code>drawSubstring(String str, int offset, int len, int x, int y, int anchor)</code> |
|      | Draws the specified String using the current font and color. |
| void | <code>fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)</code> |
|      | Fills a circular or elliptical arc covering the specified rectangle. |
| void | <code>fillRect(int x, int y, int width, int height)</code> |
|      | Fills the specified rectangle with the current color. |
| void | <code>fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)</code> |
|      | Fills the specified rounded corner rectangle with the current color. |
| void | <code>fillTriangle(int x1, int y1, int x2, int y2, int x3, int y3)</code> |
|      | Fills the specified triangle with the current color. |
| int  | <code>getBlueComponent()</code> |
|      | Gets the blue component of the current color. |
| int  | <code>getClipHeight()</code> |
|      | Gets the height of the current clipping area. |
| int  | <code>getClipWidth()</code> |
|      | Gets the width of the current clipping area. |
| int  | <code>getClipX()</code> |
|      | Gets the X offset of the current clipping area, relative to the coordinate system origin of this graphics context. |
| int  | <code>getClipY()</code> |
|      | Gets the Y offset of the current clipping area, relative to the coordinate system origin of this graphics context. |
| int  | <code>getColor()</code> |
|      | Gets the current color. |
| int  | <code>getDisplayColor(int color)</code> |
|      | Gets the color that will be displayed if the specified color is requested. |
| <code>javax.microedition.lcdui.Font</code> | <code>getFont()</code> |
|      | Gets the current font. |
| int  | <code>getGrayScale()</code> |
|      | Gets the current grayscale value of the color being used for rendering operations. |
| int  | <code>getGreenComponent()</code> |
|      | Gets the green component of the current color. |
| int  | <code>getRedComponent()</code> |
|      | Gets the red component of the current color. |
| int  | <code>getStrokeStyle()</code> |
|      | Gets the stroke style used for drawing operations. |
| int  | <code>getTranslateX()</code> |
|      | Gets the X coordinate of the translated origin of this graphics context. |</p>
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int getTranslateY()</code></td>
<td>Gets the Y coordinate of the translated origin of this graphics context.</td>
</tr>
<tr>
<td><code>void setClip(int x, int y, int width, int height)</code></td>
<td>Sets the current clip to the rectangle specified by the given coordinates.</td>
</tr>
<tr>
<td><code>void setColor(int RGB)</code></td>
<td>Sets the current color to the specified RGB values.</td>
</tr>
<tr>
<td><code>void setColor(int red, int green, int blue)</code></td>
<td>Sets the current color to the specified RGB values.</td>
</tr>
<tr>
<td><code>void setFont(Font font)</code></td>
<td>Sets the font for all subsequent text rendering operations.</td>
</tr>
<tr>
<td><code>void setGrayScale(int value)</code></td>
<td>Sets the current grayscale to be used for all subsequent rendering operations.</td>
</tr>
<tr>
<td><code>void setStrokeStyle(int style)</code></td>
<td>Sets the stroke style used for drawing lines, arcs, rectangles, and rounded rectangles.</td>
</tr>
<tr>
<td><code>void translate(int x, int y)</code></td>
<td>Translates the origin of the graphics context to the point ((x, y)) in the current coordinate system.</td>
</tr>
</tbody>
</table>

Methods inherited from class Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Fields

**BASELINE**

```java
public static final int BASELINE
```

Constant for positioning the anchor point at the baseline of text.

Value 64 is assigned to BASELINE.
Constant value: 64

**BOTTOM**

```java
public static final int BOTTOM
```

Constant for positioning the anchor point of text and images below the text or image.

Value 32 is assigned to BOTTOM.
Constant value: 32

**DOTTED**

```java
public static final int DOTTED
```

Constant for the DOTTED stroke style.

Value 1 is assigned to DOTTED.
Constant value: 1
HCENTER

public static final int HCENTER

Constant for centering text and images horizontally around the anchor point.

Value 1 is assigned to HCENTER.
Constant value: 1

LEFT

public static final int LEFT

Constant for positioning the anchor point of text and images to the left of the text or image.

Value 4 is assigned to LEFT.
Constant value: 4

RIGHT

public static final int RIGHT

Constant for positioning the anchor point of text and images to the right of the text or image.

Value 8 is assigned to RIGHT.
Constant value: 8

SOLID

public static final int SOLID

Constant for the SOLID stroke style.

Value 0 is assigned to SOLID.
Constant value: 0

TOP

public static final int TOP

Constant for positioning the anchor point of text and images above the text or image.

Value 16 is assigned to TOP.
Constant value: 16

VCENTER

public static final int VCENTER

Constant for centering images vertically around the anchor point.

Value 2 is assigned to VCENTER.
Constant value: 2

Methods
clipRect

public void clipRect (int x,  
   int y,  
   int width,  
   int height)

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: Graphics.setClip(int, int, int, int)

copyArea

public void copyArea (int x_src,  
   int y_src,  
   int width,  
   int height,  
   int x_dest,  
   int y_dest,  
   int anchor)

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:

\[
\begin{align*}
   x_{\text{src}} + tx & \geq 0 \\
   y_{\text{src}} + ty & \geq 0 \\
   x_{\text{src}} + tx + \text{width} & \leq \text{width of Graphics object's destination} \\
   y_{\text{src}} + ty + \text{height} & \leq \text{height of Graphics object's destination}
\end{align*}
\]

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
drawArc

public void drawArc(int x,
int y,
int width,
int height,
int startAngle,
int arcAngle)

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: Graphics.fillArc(int, int, int, int, int, int)

drawChar

public void drawChar(char character,
int x,
int y,
int anchor)

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: Graphics.drawString(String, int, int, int), Graphics.drawChars(char[], int, int, int, int)
**drawChars**

```java
public void drawChars(char[] data,
                      int offset,
                      int length,
                      int x,
                      int y,
                      int anchor)
```

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:** `Graphics.drawString(String, int, int, int)`

---

**drawImage**

```java
public void drawImage(Image img,
                      int x,
                      int y,
                      int anchor)
```

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` 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`

---

**drawLine**

```java
public void drawLine(int x1,
                     int y1,
                     int x2,
                     int y2)
```

Draws a line between the coordinates `(x1,y1)` and `(x2,y2)` using the current color and stroke style.
Parameters:
- \texttt{x1} - the x coordinate of the start of the line
- \texttt{y1} - the y coordinate of the start of the line
- \texttt{x2} - the x coordinate of the end of the line
- \texttt{y2} - the y coordinate of the end of the line

\textbf{drawRect}

\begin{verbatim}
public void \texttt{drawRect} (int x, 
    int y, 
    int width, 
    int height)
\end{verbatim}

Draws the outline of the specified rectangle using the current color and stroke style. The resulting rectangle will cover an area \((\text{width} + 1)\) pixels wide by \((\text{height} + 1)\) pixels tall. If either width or height is less than zero, nothing is drawn.

Parameters:
- \texttt{x} - the x coordinate of the rectangle to be drawn
- \texttt{y} - the y coordinate of the rectangle to be drawn
- \texttt{width} - the width of the rectangle to be drawn
- \texttt{height} - the height of the rectangle to be drawn

\textbf{See Also:} Graphics.fillRect(int, int, int, int)

\textbf{drawRegion}

\begin{verbatim}
public void \texttt{drawRegion} (Image src, 
    int x_src, 
    int y_src, 
    int width, 
    int height, 
    int transform, 
    int x_dest, 
    int y_dest, 
    int anchor)
\end{verbatim}
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 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
- IllegalArgumentOutOfRangeException - if transform is invalid
- IllegalArgumentOutOfRangeException - if anchor is invalid
- IllegalArgumentOutOfRangeException - if the region to be copied exceeds the bounds of the source image

Since: MIDP 2.0
drawRGB

public void drawRGB(int[] rgbData,
    int offset,
    int scanlength,
    int x,
    int y,
    int width,
    int height,
    boolean processAlpha)

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:

\[
P(a, b) = rgbData[offset + (a - x) + (b - y) \times scanlength]
\]

for

\[
x \leq a < x + width
\]
\[
y \leq b < y + height
\]

This capability is provided in the `Graphics` class so that it can be used to render both to the screen and to off-screen `Image` objects. The ability to retrieve ARGB values is provided by the `Image.getRGB(int[], int, int, int, int, int)` 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

public void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)

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: Graphics.fillRoundRect(int, int, int, int, int, int)

drawString

public void drawString(String str, int x, int y, int anchor)

Draws the specified \(\text{String}\) using the current font and color. The \(x, y\) position is the position of the anchor point. See anchor points.

Parameters:
- \(\text{str}\) - the \(\text{String}\) 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 \(\text{str}\) is null
- IllegalArgumentException - if anchor is not a legal value

See Also: Graphics.drawString(char[], int, int, int, int, int)

drawSubstring

public void drawSubstring(String str, int offset, int len, int x, int y, int anchor)

Draws the specified \(\text{String}\) 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 \(\text{str}\). The offset parameter must be within the range \([0..(\text{str}.length())]\), inclusive. The len parameter must be a non-negative integer such that \((\text{offset} + \text{len}) \leq \text{str}.length()\).

Parameters:
- \(\text{str}\) - the \(\text{String}\) 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: Graphics.drawString(String, int, int, int)

fillArc

public void fillArc(int x,
                     int y,
                     int width,
                     int height,
                     int startAngle,
                     int arcAngle)

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: Graphics.drawArc(int, int, int, int, int, int)

fillRect

public void fillRect(int x,
                      int y,
                      int width,
                      int height)

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: Graphics.drawRect(int, int, int, int)
fillRoundRect

public void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)

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: Graphics.drawRoundRect(int, int, int, int, int, int)

fillTriangle

public void fillTriangle(int x1, int y1, int x2, int y2, int x3, int y3)

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

public int getBlueComponent()

Gets the blue component of the current color.

Returns:
- integer value in range 0-255

See Also: Graphics.setColor(int, int, int)

getClipHeight

public int getClipHeight()

Gets the height of the current clipping area.

Returns:
- height of the current clipping area.

See Also: Graphics.clipRect(int, int, int, int), Graphics.setClip(int, int, int, int)
getClipWidth

public int getClipWidth()

    Gets the width of the current clipping area.

    Returns:
    width of the current clipping area.

See Also: Graphics.clipRect(int, int, int, int), Graphics.setClip(int, int, int, int)

getClipX

public int getClipX()

    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: Graphics.clipRect(int, int, int, int), Graphics.setClip(int, int, int, int)

getClipY

public int getClipY()

    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: Graphics.clipRect(int, int, int, int), Graphics.setClip(int, int, int, int)

ggetColor

public int getColor()

    Gets the current color.

    Returns:
    an integer in form 0x00RRGGBB

See Also: Graphics.setColor(int, int, int)

getDisplayColor

public int getDisplayColor(int color)

    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**

```java
public javax.microedition.lcdui.Font getFont()
```

Gets the current font.

**Returns:**
- current font

**See Also:** Font, Graphics.setFont(Font)

---

**getGrayScale**

```java
public int getGrayScale()
```

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:** Graphics.setGrayScale(int)

---

**getGreenComponent**

```java
public int getGreenComponent()
```

Gets the green component of the current color.

**Returns:**
- integer value in range 0-255

**See Also:** Graphics.setColor(int, int, int)

---

**getRedComponent**

```java
public int getRedComponent()
```

Gets the red component of the current color.

**Returns:**
- integer value in range 0-255

**See Also:** Graphics.setColor(int, int, int)

---

**getStrokeStyle**

```java
public int getStrokeStyle()
```

Gets the stroke style used for drawing operations.

**Returns:**
- stroke style, SOLID or DOTTED

**See Also:** Graphics.setStrokeStyle(int)

---

**getTranslateX**

```java
public int getTranslateX()
```

Gets the X coordinate of the translated origin of this graphics context.

**Returns:**
-
getTranslateY

public int getTranslateY()

Gets the Y coordinate of the translated origin of this graphics context.

Returns:
Y of current origin

setClip

public void setClip(int x,
                 int y,
                 int width,
                 int height)

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: Graphics.clipRect(int, int, int, int)

setColor

public void setColor(int RGB)

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: Graphics.getColor()

setColor

public void setColor(int red,
                 int green,
                 int blue)

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: Graphics.getColor()

setFont

public void setFont(Font font)
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, Graphics.getFont(), Graphics.drawString(String, int, int, int), Graphics.drawChars(char[], int, int, int, int, int)

---

**setGrayScale**

public void setGrayScale(int value)

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: Graphics.getGrayScale()

---

**setStrokeStyle**

public void setStrokeStyle(int style)

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: Graphics.getStrokeStyle()

---

**translate**

public void translate(int x, int y)

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: Graphics.getTranslateX(), Graphics.getTranslateY()
javax.microedition.lcdui

Image

Declaration

public class 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 Image.getGraphics() 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 method. It is possible to create a mutable copy of an immutable image using a technique similar to the following:

```java
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 = 2^bitdepth - 1), fully transparent pixels (alpha = 0), and semitransparent pixels (0 < alpha < 2^bitdepth - 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() 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() and getHeight() 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
```

JPEG Image Format

ISO/IEC JPEG together with JFIF must be supported by a compliant device. The support for ISO/IEC JPEG only applies to baseline DCT, non-differential, Huffman coding, as defined in table B.1, symbol 'SOF0' in [1]. This support extends to the class javax.microedition.lcdui.Image, including the following methods:

- public static Image createImage(byte[] imageData, int imageOffset, int imageLength)
- public static Image createImage(Image source)
- public static Image createImage(Image image, int x, int y, int width, int height, int transform)
- public static Image createImage(java.io.InputStream stream)
- public static Image createImage(String name)

Reference


ISO/IEC JPEG


JFIF


Since: MIDP 1.0

Method Summary

```
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static javax.microedition.lcdui.Image</td>
<td>createImage(byte[] imageData, int imageOffset, int imageLength)</td>
</tr>
<tr>
<td></td>
<td>Creates an immutable image which is decoded from the data stored in the specified byte array at the specified offset and length.</td>
</tr>
</tbody>
</table>
```
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>javax.microedition.lcdui.Image static createImage(Image source)</code></td>
<td>Creates an immutable image from a source image.</td>
</tr>
<tr>
<td><code>javax.microedition.lcdui.Image static createImage(Image image, int x, int y, int width, int height, int transform)</code></td>
<td>Creates an immutable image using pixel data from the specified region of a source image, transformed as specified.</td>
</tr>
<tr>
<td><code>javax.microedition.lcdui.Image static createImage(InputStream stream)</code></td>
<td>Creates an immutable image from decoded image data obtained from an InputStream.</td>
</tr>
<tr>
<td><code>javax.microedition.lcdui.Image static createImage(int width, int height)</code></td>
<td>Creates a new, mutable image for off-screen drawing.</td>
</tr>
<tr>
<td><code>javax.microedition.lcdui.Image static createImage(String name)</code></td>
<td>Creates an immutable image from decoded image data obtained from the named resource.</td>
</tr>
<tr>
<td><code>javax.microedition.lcdui.Image static createRGBImage(int[] rgb, int width, int height, boolean processAlpha)</code></td>
<td>Creates an immutable image from a sequence of ARGB values, specified as 0xAARRGGBB.</td>
</tr>
<tr>
<td><code>javax.microedition.lcdui.Graphics static getGraphics()</code></td>
<td>Creates a new Graphics object that renders to this image.</td>
</tr>
<tr>
<td><code>int getHeight()</code></td>
<td>Gets the height of the image in pixels.</td>
</tr>
<tr>
<td><code>void getRGB(int[] rgbData, int offset, int scanlength, int x, int y, int width, int height)</code></td>
<td>Obtains ARGB pixel data from the specified region of this image and stores it in the provided array of integers.</td>
</tr>
<tr>
<td><code>int getWidth()</code></td>
<td>Gets the width of the image in pixels.</td>
</tr>
<tr>
<td><code>boolean isMutable()</code></td>
<td>Check if this image is mutable.</td>
</tr>
</tbody>
</table>

Methods inherited from class **Object**

`equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

### Methods

**createImage**

```java
public static javax.microedition.lcdui.Image createImage(byte[] imageData,
int imageOffset,
int imageLength)
```
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**

```java
public static javax.microedition.lcdui.Image createImage(Image source)
```

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)` method, draw into it using a `Graphics` object obtained with the `getGraphics()` 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**

```java
public static javax.microedition.lcdui.Image createImage(Image image, int x, int y, int width, int height, int transform)
```
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` 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**

```java
public static javax.microedition.lcdui.Image createImage(InputStream stream) throws java.io.IOException
```

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
**createImage**

```java
public static javax.microedition.lcdui.Image createImage(int width,
                                                          int height)
```

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

**Since:** MIDP 2.0

---

**createRGBImage**

```java
public static javax.microedition.lcdui.Image createRGBImage(int[] rgb,
                                                             int width,
                                                             int height,
                                                             boolean processAlpha)
```

---

**createImage**

```java
public static javax.microedition.lcdui.Image createImage(String name)
throws java.io.IOException
```

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

---
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:

\[
P(a, b) = rgb[a + b \times width];
\]

for

\[
0 \leq a < width \\
0 \leq b < height
\]

**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 \( width \times height \).

**Since:** MIDP 2.0

### getGraphics

```java
public javax.microedition.lcdui.Graphics getGraphics()
```

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()`;
- the stroke style is `SOLID`; 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` - if the image is immutable

---

**getHeight**

```java
public int getHeight()
```

 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**

```java
public void getRGB(int[] rgbData, int offset, int scanlength, int x, int y, int width, int height)
```
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

### get getWidth

```java
class Image {
    public int getWidth() {
        // Gets the width of the image in pixels. The value returned must reflect the actual width of the image when rendered.
        return width;  // width of the image
    }
}
```

### isMutable

```java
class Image {
    public boolean isMutable() {
        // 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.
        return true;  // true if the image is mutable, false otherwise
    }
}
```
javax.microedition.lcdui

ImageItem

Declaration

public class ImageItem extends Item

Object

   +--javax.microedition.lcdui.Item
   +--javax.microedition.lcdui.ImageItem

Description

An item that can contain an image.

Each ImageItem object contains a reference to an Image 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 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

<table>
<thead>
<tr>
<th>Field Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
</tr>
<tr>
<td>public static</td>
</tr>
<tr>
<td>final</td>
</tr>
<tr>
<td>public static</td>
</tr>
<tr>
<td>final</td>
</tr>
<tr>
<td>public static</td>
</tr>
<tr>
<td>final</td>
</tr>
<tr>
<td>public static</td>
</tr>
<tr>
<td>final</td>
</tr>
<tr>
<td>public static</td>
</tr>
<tr>
<td>final</td>
</tr>
</tbody>
</table>
public static final LAYOUT_RIGHT
See Item.LAYOUT_RIGHT.
Value: 2

Fields inherited from class javax.microedition.lcdui.Item

BUTTON, HYPERLINK, LAYOUT_2, LAYOUT_BOTTOM, LAYOUT_CENTER, LAYOUT_DEFAULT, LAYOUT_EXPAND, LAYOUT_LEFT, LAYOUT_NEWLINE_AFTER, LAYOUT_NEWLINE_BEFORE, LAYOUT_RIGHT, LAYOUT_SHRINK, LAYOUT_TOP, LAYOUT_VCENTER, LAYOUT_VEXPAND, LAYOUT_VSHRINK, PLAIN

Constructor Summary

public ImageItem(String label, Image img, int layout, String altText)
Creates a new ImageItem with the given label, image, layout directive, and alternate text string.

public ImageItem(String label, Image image, int layout, String altText, int appearanceMode)
Creates a new ImageItem object with the given label, image, layout directive, alternate text string, and appearance mode.

Method Summary

java.lang.String getAltText() Gets the text string to be used if the image exceeds the device's capacity to display it.

int getAppearanceMode() Returns the appearance mode of the ImageItem.

javax.microedition.lcdui.Image getImage() Gets the image contained within the ImageItem, or null if there is no contained image.

int getLayout() Gets the layout directives used for placing the image.

void setAltText(String text) Sets the alternate text of the ImageItem, or null if no alternate text is provided.

void setImage(Image img) Sets the Image object contained within the ImageItem.

void setLayout(int layout) Sets the layout directives.

Methods inherited from class javax.microedition.lcdui.Item

addCommand, getLabel, getLayout, getMinimumHeight, getMinimumWidth, getPreferredSizeHeight, getPreferredSizeWidth, notifyStateChanged, removeCommand, setDefaultCommand, setItemCommandListener, setLabel, setLayout, setPreferredSizeSize

Methods inherited from class Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Fields
**LAYOUT_CENTER**

public static final int LAYOUT_CENTER

See Item.LAYOUT_CENTER.

Value 3 is assigned to LAYOUT_CENTER.
Constant value: 3

---

**LAYOUT_DEFAULT**

public static final int LAYOUT_DEFAULT

See Item.LAYOUT_DEFAULT.

Value 0 is assigned to LAYOUT_DEFAULT.
Constant value: 0

---

**LAYOUT_LEFT**

public static final int LAYOUT_LEFT

See Item.LAYOUT_LEFT.

Value 1 is assigned to LAYOUT_LEFT.
Constant value: 1

---

**LAYOUT_NEWLINE_AFTER**

public static final int LAYOUT_NEWLINE_AFTER

See Item.LAYOUT_NEWLINE_AFTER.

Value 0x200 is assigned to LAYOUT_NEWLINE_AFTER.
Constant value: 512

---

**LAYOUT_NEWLINE_BEFORE**

public static final int LAYOUT_NEWLINE_BEFORE

See Item.LAYOUT_NEWLINE_BEFORE.

Value 0x100 is assigned to LAYOUT_NEWLINE_BEFORE.
Constant value: 256

---

**LAYOUT_RIGHT**

public static final int LAYOUT_RIGHT

See Item.LAYOUT_RIGHT.

Value 2 is assigned to LAYOUT_RIGHT.
Constant value: 2

---

Constructors
**ImageItem**

public ImageItem(String label,  
Image img,  
int layout,  
String altText)

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);
```

**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:** javax.microedition.lcdui.ImageItem(String, Image, int, String, int)

---

**ImageItem**

public ImageItem(String label,  
Image image,  
int layout,  
String altText,  
int appearanceMode)

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, Item.HYPERLINK, or Item.BUTTON

Throws:
IllegalArgumentException - if the layout value is not a legal combination of directives
IllegalArgumentException - if appearanceMode invalid

Since: MIDP 2.0

Methods

getAltText
public java.lang.String getAltText()

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: ImageItem.setAltText(String)

getAppearanceMode
public int getAppearanceMode()

Returns the appearance mode of the ImageItem. See Appearance Modes.

Returns:
the appearance mode value, one of Item.PLAIN, Item.HYPERLINK, or Item.BUTTON

Since: MIDP 2.0

getImage
public javax.microedition.lcdui.Image getImage()

Gets the image contained within the ImageItem, or null if there is no contained image.

Returns:
image used by the ImageItem

See Also: ImageItem.setImage(Image)

getLayout
public int getLayout()

Gets the layout directives used for placing the image.

Returns:
a combination of layout directive values

See Also: ImageItem.setLayout(int)

setAltText
public void setAltText(String text)

Sets the alternate text of the ImageItem, or null if no alternate text is provided.

Parameters:
text - the new alternate text

See Also: ImageItem.getAltText()
setImage

public void setImage(Image img)

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

eItem.setItem(eItem.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: ImageItem.getImage()

setLayout

public void setLayout(int layout)

Sets the layout directives.

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: ImageItem.getLayout()
javax.microedition.lcdui

Item

Declaration

public abstract class Item

Object

down--javax.microedition.lcdui.Item

Direct Known Subclasses:

javax.microedition.lcdui.ChoiceGroup,
javax.microedition.lcdui.CustomButton,
javax.microedition.lcdui.DateField, javax.microedition.lcdui.Gauge,
javax.microedition.lcdui.ImageItem, javax.microedition.lcdui.Spacer,
javax.microedition.lcdui.StringItem,
javax.microedition.lcdui.TextField

Description

A superclass for components that can be added to a Form. 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 `Item.setLayout(int)` method and is the return value from the `Item.getLayout()` 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` 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 `getPreferredSize` and `getPreferredHeight` 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 Item.addCommand(Command) or Item.setDefaultCommand(Command) and if the Command has not been removed with a subsequent call to Item.removeCommand(Command). 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 Item.setDefaultCommand(Command) 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, HYPERLINK, or BUTTON. 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. A StringItem that has item commands and the appearance mode Item.PLAIN MUST always be presented as a StringItem with added command(s) and appearance mode HYPERLINK.

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

<table>
<thead>
<tr>
<th>Field Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>An appearance mode value indicating that the Item MUST appear as a button.</td>
</tr>
<tr>
<td>Value: 2</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>An appearance mode value indicating that the Item MUST appear as a hyperlink.</td>
</tr>
<tr>
<td>Value: 1</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>A layout directive indicating that new MIDP 2.0 layout rules are in effect for this Item.</td>
</tr>
<tr>
<td>Value: 16384</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>A layout directive indicating that this Item MUST have a bottom-aligned layout.</td>
</tr>
<tr>
<td>Value: 32</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>A layout directive indicating that this Item MUST have a horizontally centered layout.</td>
</tr>
<tr>
<td>Value: 3</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>A layout directive indicating that this Item MUST follow the default layout policy of its container.</td>
</tr>
<tr>
<td>Value: 0</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>An implementation MUST interpret this layout directive as an indication that this Item's width can be increased to fill available space.</td>
</tr>
<tr>
<td>Value: 2048</td>
</tr>
<tr>
<td>Method</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>addCommand</td>
</tr>
<tr>
<td>getLabel</td>
</tr>
<tr>
<td>getLayout</td>
</tr>
<tr>
<td>Method</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td><code>int getMinimumHeight()</code></td>
</tr>
<tr>
<td><code>int getMinimumWidth()</code></td>
</tr>
<tr>
<td><code>int getPreferredSizeHeight()</code></td>
</tr>
<tr>
<td><code>int getPreferredSizeWidth()</code></td>
</tr>
<tr>
<td><code>void notifyStateChanged()</code></td>
</tr>
<tr>
<td><code>void removeCommand(Command cmd)</code></td>
</tr>
<tr>
<td><code>void setDefaultCommand(Command cmd)</code></td>
</tr>
<tr>
<td><code>void setItemCommandListener(ItemCommandListener l)</code></td>
</tr>
<tr>
<td><code>void setLabel(String label)</code></td>
</tr>
<tr>
<td><code>void setLayout(int layout)</code></td>
</tr>
<tr>
<td><code>void setPreferredSize(int width, int height)</code></td>
</tr>
</tbody>
</table>

Methods inherited from class Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Fields**

**BUTTON**

public static final int BUTTON

An appearance mode value indicating that the Item MUST appear as a button.

Value 2 is assigned to BUTTON.
Constant value: 2
Since: MIDP 2.0

**HYPERLINK**

public static final int HYPERLINK

An appearance mode value indicating that the Item MUST appear as a hyperlink.

Value 1 is assigned to HYPERLINK.
Constant value: 1
**LAYOUT_2**

```java
public static final int LAYOUT_2
```

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.
Constant value: 16384

**Since:** MIDP 2.0

---

**LAYOUT_BOTTOM**

```java
public static final int LAYOUT_BOTTOM
```

A layout directive indicating that this Item MUST have a bottom-aligned layout.

Value 0x20 is assigned to LAYOUT_BOTTOM.
Constant value: 32

**Since:** MIDP 2.0

---

**LAYOUT_CENTER**

```java
public static final int LAYOUT_CENTER
```

A layout directive indicating that this Item MUST have a horizontally centered layout.

Value 3 is assigned to LAYOUT_CENTER.
Constant value: 3

**Since:** MIDP 2.0

---

**LAYOUT_DEFAULT**

```java
public static final int LAYOUT_DEFAULT
```

A layout directive indicating that this Item MUST follow the default layout policy of its container.

Value 0 is assigned to LAYOUT_DEFAULT.
Constant value: 0

**Since:** MIDP 2.0

---

**LAYOUT_EXPAND**

```java
public static final int LAYOUT_EXPAND
```

An implementation MUST interpret this layout directive as an indication that this Item's width can be increased to fill available space.

Value 0x800 is assigned to LAYOUT_EXPAND.
Constant value: 2048

**Since:** MIDP 2.0

---

**LAYOUT_LEFT**

```java
public static final int LAYOUT_LEFT
```
A layout directive indicating that this Item MUST have a left-aligned layout.

Value 1 is assigned to LAYOUT_LEFT.
Constant value: 1

Since: MIDP 2.0

LAYOUT_NEWLINE_AFTER

public static final int LAYOUT_NEWLINE_AFTER

A layout directive indicating that this Item MUST be 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.
Constant value: 512

Since: MIDP 2.0

LAYOUT_NEWLINE_BEFORE

public static final int LAYOUT_NEWLINE_BEFORE

A layout directive indicating that this Item MUST be placed at the beginning of a new line or row.

Value 0x100 is assigned to LAYOUT_NEWLINE_BEFORE.
Constant value: 256

Since: MIDP 2.0

LAYOUT_RIGHT

public static final int LAYOUT_RIGHT

A layout directive indicating that this Item MUST have a right-aligned layout.

Value 2 is assigned to LAYOUT_RIGHT.
Constant value: 2

Since: MIDP 2.0

LAYOUT_SHRINK

public static final int LAYOUT_SHRINK

An implementation MUST interpret this layout directive as an indication that this Item's width can be reduced to its minimum width.

Value 0x400 is assigned to LAYOUT_SHRINK.
Constant value: 1024

Since: MIDP 2.0

LAYOUT_TOP

public static final int LAYOUT_TOP

A layout directive indicating that this Item MUST have a top-aligned layout.

Value 0x10 is assigned to LAYOUT_TOP.
Constant value: 16

Since: MIDP 2.0
LAYOUT_VCENTER

```java
public static final int LAYOUT_VCENTER
```

A layout directive indicating that this Item MUST have a vertically centered layout.

Value 0x30 is assigned to LAYOUT_VCENTER.
Constant value: 48

Since: MIDP 2.0

LAYOUT_VEXPAND

```java
public static final int LAYOUT_VEXPAND
```

An implementation MUST interpret this layout directive as an indication that this Item’s height can be increased to fill available space.

Value 0x2000 is assigned to LAYOUT_VEXPAND.
Constant value: 8192

Since: MIDP 2.0

LAYOUT_VSHRINK

```java
public static final int LAYOUT_VSHRINK
```

An implementation MUST interpret this layout directive as an indication that this Item’s height can be reduced to its minimum height.

Value 0x1000 is assigned to LAYOUT_VSHRINK.
Constant value: 4096

Since: MIDP 2.0

PLAIN

```java
public static final int PLAIN
```

An appearance mode value indicating that the Item Must have a normal appearance.

Value 0 is assigned to PLAIN.
Constant value: 0

Since: MIDP 2.0

Methods

**addCommand**

```java
public void addCommand(Command cmd)
```

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

public java.lang.String getLabel()

Gets the label of this Item object.

Returns: the label string

See Also: Item.setLabel(String)

getLayout

public int getLayout()

Gets the layout directives used for placing the item.

Returns: a combination of layout directive values

See Also: Item.setLayout(int)
Since: MIDP 2.0

getMinimumHeight

public int getMinimumHeight()

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

public int getMinimumWidth()

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

public int getPreferredHeight()

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
See Also: Item.getPreferredWidth(), Item.setPreferredSize(int, int)
Since: MIDP 2.0
getPreferredWidth

public int getPreferredWidth()

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

See Also: Item.getPreferredHeight(), Item.setPreferredSize(int, int)
Since: MIDP 2.0

notifyStateChanged

public void notifyStateChanged()

Causes this Item's containing Form to notify the Item's ItemStateListener. 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 ItemStateListerner.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 - if the Item is not owned by a Form

Since: MIDP 2.0

removeCommand

public void removeCommand(Command cmd)
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:

```java
// Command c is the default command on Item item
item.removeCommand(c);
```

is equivalent to the following code:

```java
// 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

```
public void setDefaultCommand(Command cmd)
```

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 `Item.addCommand(Command)` 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` - if this Item is contained within an Alert

**Since:** MIDP 2.0

---

### setItemCommandListener

```
public void setItemCommandListener(ItemCommandListener l)
```

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 - if this Item is contained within an Alert

setLabel

public void setLabel(String label)

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 - if this Item is contained within an Alert

See Also: Item.getLabel()
javax.microedition.lcdui
ItemCommandListener

**Declaration**

```java
public interface ItemCommandListener
```

**Description**

A listener type for receiving notification of commands that have been invoked on `Item` objects. An `Item` can have commands associated with it. When such a command is invoked, the application is notified by having the `commandAction()` method called on the `ItemCommandListener` that had been set on the `Item` with a call to `setItemCommandListener()`.

**Since:** MIDP 2.0

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void commandAction(Command c, Item item)</code></td>
<td>Called by the system to indicate that a command has been invoked on a particular item.</td>
</tr>
</tbody>
</table>

**Methods**

**commandAction**

```java
public void commandAction(Command c, Item item)
```

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

ItemStateListerner

Declaration

public interface ItemStateListerner

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 screen.

Since: MIDP 1.0

See Also: Form.setItemStateListener(ItemStateListerner)

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void itemStateChanged(Item item)</td>
<td>Called when internal state of an Item has been changed by the user.</td>
</tr>
</tbody>
</table>

Methods

itemStateChanged

public void itemStateChanged(Item item)

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() 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.

Parameters:

item - the item that was changed
javax.microedition.lcdui

List

Declaration

public class List extends Screen implements Choice

Object

| javax.microedition.lcdui.Displayable
| javax.microedition.lcdui.Screen
| javax.microedition.lcdui.List

All Implemented Interfaces:
javax.microedition.lcdui.Choice

Description

A Screen containing list of choices. Most of its behavior is common with class ChoiceGroup, and their common API. The different List types in particular, are defined in interface Choice. 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 is invoked by notifying its CommandListener. 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, Choice.MULTIPLE, and Choice.IMPLICIT. The Choice type Choice.POPUP 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. 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 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:

```java
String[] elements = { ... }; //Menu items as List elements
List menuList = new List("Menu", List.IMPLICIT, elements, null);
Command selectCommand = new Command("Open", Command.ITEM, 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.

```java
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

### Field Summary

<table>
<thead>
<tr>
<th>public static final</th>
<th>SELECT_COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The default select command for IMPLICIT Lists.</td>
</tr>
</tbody>
</table>

Fields inherited from interface `javax.microedition.lcdui.Choice`

- EXCLUSIVE, IMPLICIT, MULTIPLE, POPUP, TEXT_WRAP_DEFAULT, TEXT_WRAP_OFF, TEXT_WRAP_ON

### Constructor Summary

<table>
<thead>
<tr>
<th>public</th>
<th>List(String title, int listType)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Creates a new, empty List, specifying its title and the type of the list.</td>
</tr>
</tbody>
</table>
public List(String title, int listType, String[] stringElements,
            Image[] imageElements)
    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.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int append(String stringPart, Image imagePart)</td>
<td>Appends an element to the List.</td>
</tr>
<tr>
<td>void delete(int elementNum)</td>
<td>Deletes the element referenced by elementNum.</td>
</tr>
<tr>
<td>void deleteAll()</td>
<td>Deletes all elements from this List.</td>
</tr>
<tr>
<td>int getFitPolicy()</td>
<td>Gets the application's preferred policy for fitting Choice element contents to the available screen space.</td>
</tr>
<tr>
<td>javax.microedition.lcdui.Font getFont(int elementNum)</td>
<td>Gets the application's preferred font for rendering the specified element of this Choice.</td>
</tr>
<tr>
<td>javax.microedition.lcdui.Image getImage(int elementNum)</td>
<td>Gets the Image part of the element referenced by elementNum.</td>
</tr>
<tr>
<td>int getSelectedFlags(boolean[] selectedArray_return)</td>
<td>Queries the state of a List and returns the state of all elements in the boolean array selectedArray_return.</td>
</tr>
<tr>
<td>int getSelectedIndex()</td>
<td>Returns the index number of an element in the List that is selected.</td>
</tr>
<tr>
<td>java.lang.String getString(int elementNum)</td>
<td>Gets the String part of the element referenced by elementNum.</td>
</tr>
<tr>
<td>void insert(int elementNum, String stringPart, Image imagePart)</td>
<td>Inserts an element into the List just prior to the element specified.</td>
</tr>
<tr>
<td>boolean isSelected(int elementNum)</td>
<td>Gets a boolean value indicating whether this element is selected.</td>
</tr>
<tr>
<td>void removeCommand(Command cmd)</td>
<td>The same as Displayable.removeCommand but with the following additional semantics.</td>
</tr>
<tr>
<td>void set(int elementNum, String stringPart, Image imagePart)</td>
<td>Sets the String and Image parts of the element referenced by elementNum, replacing the previous contents of the element.</td>
</tr>
<tr>
<td>void setFitPolicy(int fitPolicy)</td>
<td>Sets the application’s preferred policy for fitting Choice element contents to the available screen space.</td>
</tr>
<tr>
<td>void setFont(int elementNum, Font font)</td>
<td>Sets the application’s preferred font for rendering the specified element of this Choice.</td>
</tr>
<tr>
<td>void setSelectCommand(Command command)</td>
<td>Sets the Command to be used for an IMPLICIT List selection action.</td>
</tr>
</tbody>
</table>
void setSelectedFlags(boolean[] selectedArray)
Sets the selected state of all elements of the List.

void setSelectedIndex(int elementNum, boolean selected)
Sets the selected state of an element.

int size()
Gets the number of elements in the List.

Methods inherited from class javax.microedition.lcdui.Displayable

addCommand, getHeight, getTicker, getTitle, getWidth, isShown, removeCommand,
setCommandListener, setTicker, setTitle, sizeChanged

Methods inherited from class Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from interface javax.microedition.lcdui.Choice

append, delete, deleteAll, getFitPolicy, getFont, getImage, getSelectedFlags,
getSelectedIndex, getString, insert, isSelected, set, setFitPolicy, setFont,
setSelectedFlags, setSelectedIndex, size

Fields

SELECT_COMMAND

public static final javax.microedition.lcdui.Command SELECT_COMMAND

The default select command for IMPLICIT Lists. Applications using an IMPLICIT List should set their own select command using setSelectCommand.

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

public List(String title,
         int listType)

Creates a new, empty List, specifying its title and the type of the list.

Parameters:
    title - the screen's title (see Displayable)
    listType - one of IMPLICIT, EXCLUSIVE, or MULTIPLE

Throws:
See Also: Choice

List

public List(String title,
            int listType,
            String[] stringElements,
            Image[] imageElements)

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)
  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, Choice.MULTIPLE, Choice.IMPLICIT

Methods

append

public int append(String stringPart,
                   Image imagePart)

Appends an element to the List.

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

public void delete(int elementNum)

Deletes the element referenced by elementNum.

Parameters:
  elementNum - the index of the element to be deleted

Throws:
  IndexOutOfBoundsException - if elementNum is invalid
**deleteAll**

```java
public void deleteAll()
```

Deletes all elements from this List.

**getFitPolicy**

```java
public int getFitPolicy()
```

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 `Choice.TEXT_WRAP_DEFAULT, Choice.TEXT_WRAP_ON, or Choice.TEXT_WRAP_OFF`

**See Also:** `List.setFitPolicy(int)`

**Since:** MIDP 2.0

**getFont**

```java
public javax.microedition.lcdui.Font getFont(int elementNum)
```

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

**See Also:** `List.setFont(int, Font)`

**Since:** MIDP 2.0

**getImage**

```java
public javax.microedition.lcdui.Image getImage(int elementNum)
```

Gets the `Image` part of the element referenced by `elementNum`.

**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:** `List.getString(int)`

**getSelectedFlags**

```java
public int getSelectedFlags(boolean[] selectedArray_return)
```

Queries the state of a List and returns the state of all elements in the boolean array `selectedArray_return`. 
**getSelectedIndex**

```java
public int getSelectedIndex()
```

Returns the index number of an element in the List that is selected.

**Parameters:**
- `elementNum` - the index of the element to be queried

**Returns:**
- index of selected element, or -1 if none

**See Also:** List.setSelectedIndex(int, boolean)

**getString**

```java
public java.lang.String getString(int elementNum)
```

 Gets the String part of the element referenced by `elementNum`.

**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:** List.getImage(int)

**insert**

```java
public void insert(int elementNum, String stringPart, Image imagePart)
```

Inserts an element into the List just prior to the element specified.

**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**

```java
public boolean isSelected(int elementNum)
```

Gets a boolean value indicating whether this element is selected.

**Parameters:**
- `elementNum` - index to element to be queried

**Returns:**
- selection state of the element
Throws:
    IndexOutOfBoundsException - if elementNum is invalid

removeCommand

public void removeCommand(Command cmd)

The same as Displayable.removeCommand 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:

```java
// Command c is the select command on List list
list.removeCommand(c);
```

is equivalent to the following code:

```java
// Command c is the select command on List list
list.setSelectCommand(null);
list.removeCommand(c);
```

Parameters:
    cmd - the command to be removed

Since: MIDP 2.0

set

public void set(int elementNum,
        String stringPart,
        Image imagePart)

Sets the String and Image parts of the element referenced by elementNum, replacing the previous contents of the element.

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

public void setFitPolicy(int fitPolicy)

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, Choice.TEXT_WRAP_ON, and Choice.TEXT_WRAP_OFF. Fit policy is a hint, and the implementation may disregard the application's preferred policy.
setFont

```java
public void setFont(int elementNum, Font font)
```

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

**See Also:** `List.getFont(int)`

**Since:** MIDP 2.0

setSelectCommand

```java
public void setSelectCommand(Command command)
```

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

```java
public void setSelectedFlags(boolean[] selectedArray)
```

**See Also:** `List.getFitPolicy()`

**Since:** MIDP 2.0
Sets the selected state of all elements of the List.

**Parameters:**
- `selectedArray` - an array in which the method collects the selection status

**Throws:**
- `IllegalArgumentException` - if `selectedArray` is shorter than the size of the List
- `NullPointerException` - if `selectedArray` is null

**See Also:** `List.getSelectedFlags(boolean[])`

---

**setSelectedIndex**

```java
public void setSelectedIndex(int elementNum, boolean selected)
```

Sets the selected state of an element.

**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:** `List.getSelectedIndex()`

---

**size**

```java
public int size()
```

Gets the number of elements in the List.

**Returns:**
- the number of elements in the List
javax.microedition.lcdui

Screen

Declaration

public abstract class Screen extends Displayable

Object

|--javax.microedition.lcdui.Displayable
|   |--javax.microedition.lcdui.Screen

Direct Known Subclasses:

javax.microedition.lcdui.Alert, javax.microedition.lcdui.Form,
javax.microedition.lcdui.List, javax.microedition.lcdui.TextBox

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

| Methods inherited from class javax.microedition.lcdui.Displayable |
|---|---|
| addCommand, getHeight, getTicker, getTitle, getWidth, isShown, removeCommand, |
| setCommandListener, setTicker, setTitle, sizeChanged |

| Methods inherited from class Object |
|---|---|
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |
javax.microedition.lcdui

Spacer

Declaration

public class Spacer extends Item

Object

|-- javax.microedition.lcdui.Item
|-- 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

Fields inherited from class javax.microedition.lcdui.Item

BUTTON, HYPERLINK, LAYOUT_2, LAYOUT_BOTTOM, LAYOUT_CENTER, LAYOUT_DEFAULT, LAYOUT_EXPAND, LAYOUT_LEFT, LAYOUT_NEWLINE_AFTER, LAYOUT_NEWLINE_BEFORE, LAYOUT_RIGHT, LAYOUT_SHRINK, LAYOUT_TOP, LAYOUT_VCENTER, LAYOUT_VEXPAND, LAYOUT_VSHRINK, PLAIN

Constructor Summary

public Spacer(int minWidth, int minHeight)

Creates a new Spacer with the given minimum size.

Method Summary

void addCommand(Command cmd)

Spacers are restricted from having Commands, so this method will always throw IllegalStateException whenever it is called.

void setDefaultCommand(Command cmd)

Spacers are restricted from having Commands, so this method will always throw IllegalStateException whenever it is called.

void setLabel(String label)

Spacers are restricted to having null labels, so this method will always throw IllegalStateException whenever it is called.

void setMinimumSize(int minWidth, int minHeight)

Sets the minimum size for this spacer.
<table>
<thead>
<tr>
<th>Methods inherited from class</th>
<th>javax.microedition.lcdui.Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>addCommand, getLabel, getLayout, getMinimumHeight, getMinimumWidth, getPreferredSize, getHeight, getPreferredSize, notifyStateChanged, removeCommand, setDefaultCommand, setItemCommandListener,setLabel, setLayout, getPreferredSize</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods inherited from class</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait</td>
<td></td>
</tr>
</tbody>
</table>

### Constructors

**Spacer**

```java
public Spacer(int minWidth, int minHeight)
```

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

```java
public void addCommand(Command cmd)
```

Spacers are restricted from having Commands, so this method will always throw `IllegalStateException` whenever it is called.

**Parameters:**

- `cmd` - the Command

**Throws:**

- `IllegalStateException` - always

#### setDefaultCommand

```java
public void setDefaultCommand(Command cmd)
```

Spacers are restricted from having Commands, so this method will always throw `IllegalStateException` whenever it is called.

**Parameters:**

- `cmd` - the Command

**Throws:**

- `IllegalStateException` - always

#### setLabel

```java
public void setLabel(String label)
```
Spacers are restricted to having null labels, so this method will always throw `IllegalStateException` whenever it is called.

**Parameters:**
- `label` - the label string

**Throws:**
- `IllegalStateException` - always

---

### setMinimumSize

```java
public void setMinimumSize(int minWidth, int minHeight)
```

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 Item

Object

|-- javax.microedition.lcdui.Item

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>------</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
</tbody>
</table>

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.

Fields inherited from class javax.microedition.lcdui.Item

BUTTON, HYPERLINK, LAYOUT_2, LAYOUT_BOTTOM, LAYOUT_CENTER, LAYOUT_DEFAULT, LAYOUT_EXPAND, LAYOUT_LEFT, LAYOUT_NEWLINE_AFTER, LAYOUT_NEWLINE_BEFORE, LAYOUT_RIGHT, LAYOUT_SHRINK, LAYOUT_TOP, LAYOUT_VCENTER, LAYOUT_VEXPAND, LAYOUT_VSHRINK, PLAIN

Constructor Summary

| public StringItem(String label, String text) |
| Creates a new StringItem object. |

| public StringItem(String label, String text, int appearanceMode) |
| Creates a new StringItem object with the given label, textual content, and appearance mode. |

Method Summary

| int | getAppearanceMode() |
| Returns the appearance mode of the StringItem. |

| javax.microedition.lcdui.Font | getFont() |
| Gets the application's preferred font for rendering this StringItem. |

| java.lang.String | getText() |
| Gets the text contents of the StringItem, or null if the StringItem is empty. |

| void | setFont(Font font) |
| Sets the application's preferred font for rendering this StringItem. |

| void | setText(String text) |
| Sets the text contents of the StringItem. |

Methods inherited from class javax.microedition.lcdui.Item

addCommand, getLabel, getLayout, getMinimumHeight, getMinimumWidth, getPreferredSizeHeight, getPreferredSizeWidth, notifyStateChanged, removeCommand, setDefaultCommand, setItemCommandListener, setLabel, setLayout, setPreferredSize
Methods inherited from class Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

StringItem

public StringItem(String label, String text)

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: javax.microedition.lcdui.StringItem(String, String, int)

StringItem

public StringItem(String label, String text, int appearanceMode)

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:

```java
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, Item.HYPERLINK, or Item.BUTTON. A StringItem that has item commands and the appearance mode Item.PLAIN MUST always be presented as a StringItem with added command(s) and appearance mode HYPERLINK.
getAppearanceMode

```java
public int getAppearanceMode()
```

Returns the appearance mode of the StringItem. See Appearance Modes.

Returns: the appearance mode value, one of Item.PLAIN, Item.HYPERLINK, or Item.BUTTON

Since: MIDP 2.0

getFont

```java
public javax.microedition.lcdui.Font getFont()
```

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

See Also: StringItem.setFont(Font)

Since: MIDP 2.0

getText

```java
public java.lang.String getText()
```

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: StringItem.setText(String)

setFont

```java
public void setFont(Font font)
```

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

See Also: StringItem.getFont()

Since: MIDP 2.0

setText

```java
public void setText(String text)
```

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: `StringItem.getText()`
javax.microedition.lcdui

TextBox

Declaration

public class TextBox extends Screen

Object

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(). 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 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 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

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public TextBox(String title, String text, int maxSize, int constraints)</td>
<td>Creates a new TextBox object with the given title string, initial contents, maximum size in characters, and constraints.</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void delete(int offset, int length)</td>
<td>Deletes characters from the TextBox.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>int getCaretPosition()</code></td>
<td>Gets the current input position.</td>
</tr>
<tr>
<td><code>int getChars(char[] data)</code></td>
<td>Copies the contents of the TextBox into a character array starting at index zero.</td>
</tr>
<tr>
<td><code>int getConstraints()</code></td>
<td>Gets the current input constraints of the TextBox.</td>
</tr>
<tr>
<td><code>int getMaxSize()</code></td>
<td>Returns the maximum size (number of characters) that can be stored in this TextBox.</td>
</tr>
<tr>
<td><code>java.lang.String getString()</code></td>
<td>Gets the contents of the TextBox as a string value.</td>
</tr>
<tr>
<td><code>void insert(char[] data, int offset, int length, int position)</code></td>
<td>Inserts a subrange of an array of characters into the contents of the TextBox.</td>
</tr>
<tr>
<td><code>void insert(String src, int position)</code></td>
<td>Inserts a string into the contents of the TextBox.</td>
</tr>
<tr>
<td><code>void setChars(char[] data, int offset, int length)</code></td>
<td>Sets the contents of the TextBox from a character array, replacing the previous contents.</td>
</tr>
<tr>
<td><code>void setConstraints(int constraints)</code></td>
<td>Sets the input constraints of the TextBox.</td>
</tr>
<tr>
<td><code>void setInitialInputMode(String characterSubset)</code></td>
<td>Sets a hint to the implementation as to the input mode that should be used when the user initiates editing of this TextBox.</td>
</tr>
<tr>
<td><code>int setMaxSize(int maxSize)</code></td>
<td>Sets the maximum size (number of characters) that can be contained in this TextBox.</td>
</tr>
<tr>
<td><code>void setString(String text)</code></td>
<td>Sets the contents of the TextBox as a string value, replacing the previous contents.</td>
</tr>
<tr>
<td><code>int size()</code></td>
<td>Gets the number of characters that are currently stored in this TextBox.</td>
</tr>
</tbody>
</table>

**Methods inherited from class** [javax.microedition.lcdui.Displayable](https://developer.mozilla.org/en-US/docs/Web/API/javax/microedition/lcdui/Displayable)

- `addCommand`, `getHeight`, `getTicker`, `getTitle`, `getWidth`, `isShown`, `removeCommand`, `setCommandListener`, `setTicker`, `setTitle`, `sizeChanged`

**Methods inherited from class** [Object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object)

- `equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

**Constructors**
public TextBox(String title,
        String text,
        int maxSize,
        int constraints)

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 TextBox.getMaxSize().
- 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

dele

public void delete(int offset,
        int length)

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

gCaretP

public int getCaretPosition()

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

gChar

public int getChars(char[] data)
Covers 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:** `TextBox.setChars(char[], int, int)`

---

**getConstraints**

```
public int getConstraints()
```

Gets the current input constraints of the `TextBox`.

**Returns:**
- the current constraints value (see input constraints)

**See Also:** `TextBox.setConstraints(int)`

---

**getMaxSize**

```
public int getMaxSize()
```

Returns the maximum size (number of characters) that can be stored in this `TextBox`.

**Returns:**
- the maximum size in characters

**See Also:** `TextBox.setMaxSize(int)`

---

**getString**

```
public java.lang.String getString()
```

Gets the contents of the `TextBox` as a string value.

**Returns:**
- the current contents

**See Also:** `TextBox.setString(String)`

---

**insert**

```
public void insert(char[] data,
                    int offset,
                    int length,
                    int position)
```

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)`.

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
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() 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

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: TextBox.getChars(char[])
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:** `TextBox.getConstraints()`

### setInitialInputMode

```java
public void setInitialInputMode(String characterSubset)
```

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

```java
public int setMaxSize(int maxSize)
```

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:** `TextBox.getMaxSize()`

### setString

```java
public void setString(String text)
```

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:** `TextBox.getString()`

### size

```java
public int size()
```

Gets the number of characters that are currently stored in this TextBox.
**Returns:**
the number of characters
TextField

Declaration

```java
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()` 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

`TextBox` and `TextField` with input constraint `TextField.ANY` must support input of the following set of characters.

**Characters Supported by `TextBox` and `TextField`**

<table>
<thead>
<tr>
<th>Glyph</th>
<th>Unicode Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>U+00A</td>
<td>line feed</td>
</tr>
<tr>
<td></td>
<td>U+0020</td>
<td>space</td>
</tr>
<tr>
<td>!</td>
<td>U+0021</td>
<td>exclamation mark</td>
</tr>
<tr>
<td>&quot;</td>
<td>U+0022</td>
<td>quotation mark</td>
</tr>
<tr>
<td>#</td>
<td>U+0023</td>
<td>number sign</td>
</tr>
<tr>
<td>$</td>
<td>U+0024</td>
<td>dollar sign</td>
</tr>
<tr>
<td>%</td>
<td>U+0025</td>
<td>percent</td>
</tr>
<tr>
<td>&amp;</td>
<td>U+0026</td>
<td>ampersand</td>
</tr>
<tr>
<td>'</td>
<td>U+0027</td>
<td>apostrophe</td>
</tr>
<tr>
<td>(</td>
<td>U+0028</td>
<td>left parenthesis</td>
</tr>
<tr>
<td>)</td>
<td>U+0029</td>
<td>right parenthesis</td>
</tr>
<tr>
<td>*</td>
<td>U+002A</td>
<td>asterisk</td>
</tr>
<tr>
<td>+</td>
<td>U+002B</td>
<td>plus</td>
</tr>
<tr>
<td>,</td>
<td>U+002C</td>
<td>comma</td>
</tr>
<tr>
<td>-</td>
<td>U+002D</td>
<td>hyphen-minus</td>
</tr>
<tr>
<td>.</td>
<td>U+002E</td>
<td>period</td>
</tr>
<tr>
<td>/</td>
<td>U+002F</td>
<td>slash</td>
</tr>
<tr>
<td>0-9</td>
<td>U+0030 - U+0039</td>
<td>digit 0 through 9</td>
</tr>
<tr>
<td>:</td>
<td>U+003A</td>
<td>colon</td>
</tr>
<tr>
<td>;</td>
<td>U+003B</td>
<td>semicolon</td>
</tr>
<tr>
<td>&lt;</td>
<td>U+003C</td>
<td>less-than sign</td>
</tr>
<tr>
<td>=</td>
<td>U+003D</td>
<td>equals sign</td>
</tr>
<tr>
<td>&gt;</td>
<td>U+003E</td>
<td>greater-than sign</td>
</tr>
<tr>
<td>?</td>
<td>U+003F</td>
<td>question mark</td>
</tr>
<tr>
<td>@</td>
<td>U+0040</td>
<td>commercial at</td>
</tr>
<tr>
<td>A-Z</td>
<td>U+0041 - U+005A</td>
<td>Latin capital letter A through Z</td>
</tr>
<tr>
<td>[</td>
<td>U+005B</td>
<td>left square bracket</td>
</tr>
<tr>
<td>\</td>
<td>U+005C</td>
<td>backslash</td>
</tr>
<tr>
<td>]</td>
<td>U+005D</td>
<td>right square bracket</td>
</tr>
<tr>
<td>^</td>
<td>U+005E</td>
<td>circumflex accent</td>
</tr>
</tbody>
</table>
Instances of TextField and TextBox with either of the constraints TextField.EMAILADDR and TextField.URL must allow the same characters to be input as are allowed for input constraint TextField.ANY.

### Input Modes

The TextField shares the concept of input modes with the TextBox 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() 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

```java
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()` 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_ARMEANIAN
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:

```java
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

<table>
<thead>
<tr>
<th>Field Summary</th>
</tr>
</thead>
</table>
| **public static final** ANY | The user is allowed to enter any text.  
Value: 0 |
| **public static final** CONSTRAINT_MASK | The mask value for determining the constraint mode.  
Value: 65535 |
| **public static final** DECIMAL | The user is allowed to enter numeric values with optional decimal fractions, for example ".123", "0.123", or ".5".  
Value: 5 |
| **public static final** EMAILADDR | The user is allowed to enter an e-mail address.  
Value: 1 |
| **public static final** INITIAL_CAPS_SENTENCE | This flag is a hint to the implementation that during text editing, the initial letter of each sentence should be capitalized.  
Value: 2097152 |
| **public static final** INITIAL_CAPS_WORD | This flag is a hint to the implementation that during text editing, the initial letter of each word should be capitalized.  
Value: 1048576 |
| **public static final** NON_PREDICTIVE | Indicates that the text entered does not consist of words that are likely to be found in dictionaries typically used by predictive input schemes.  
Value: 524288 |
| **public static final** NUMERIC | The user is allowed to enter only an integer value.  
Value: 2 |
| **public static final** PASSWORD | Indicates that the text entered is confidential data that should be obscured whenever possible.  
Value: 65536 |
| **public static final** PHONENUMBER | The user is allowed to enter a phone number.  
Value: 3 |
| **public static final** SENSITIVE | 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.  
Value: 262144 |
<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNEDITABLE</td>
<td>131072</td>
<td>Indicates that editing is currently disallowed.</td>
</tr>
<tr>
<td>URL</td>
<td>4</td>
<td>The user is allowed to enter a URL.</td>
</tr>
</tbody>
</table>

Fields inherited from class `javax.microedition.lcdui.Item`:
- `BUTTON`, `HYPERLINK`, `LAYOUT_2`, `LAYOUT_BOTTOM`, `LAYOUT_CENTER`, `LAYOUT_DEFAULT`, `LAYOUT_EXPAND`, `LAYOUT_LEFT`, `LAYOUT_NEWLINE_AFTER`, `LAYOUT_NEWLINE_BEFORE`, `LAYOUT_RIGHT`, `LAYOUT_SHRINK`, `LAYOUT_TOP`, `LAYOUT_VCENTER`, `LAYOUT_VEXPAND`, `LAYOUT_VSHRINK`, `PLAIN`

Constructor Summary:

```java
public TextField(String label, String text, int maxSize, int constraints)
Creates a new TextField object with the given label, initial contents, maximum size in characters, and constraints.
```

Method Summary:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete(int offset, int length)</td>
<td>Deletes characters from the TextField.</td>
</tr>
<tr>
<td>getCaretPosition()</td>
<td>Gets the current input position.</td>
</tr>
<tr>
<td>getChars(char[] data)</td>
<td>Copies the contents of the TextField into a character array starting at index zero.</td>
</tr>
<tr>
<td>getConstraints()</td>
<td>Gets the current input constraints of the TextField.</td>
</tr>
<tr>
<td>getMaxSize()</td>
<td>Returns the maximum size (number of characters) that can be stored in this TextField.</td>
</tr>
<tr>
<td>getString()</td>
<td>Gets the contents of the TextField as a string value.</td>
</tr>
<tr>
<td>insert(char[] data, int offset, int length, int position)</td>
<td>Inserts a subrange of an array of characters into the contents of the TextField.</td>
</tr>
<tr>
<td>insert(String src, int position)</td>
<td>Inserts a string into the contents of the TextField.</td>
</tr>
<tr>
<td>setChars(char[] data, int offset, int length)</td>
<td>Sets the contents of the TextField from a character array, replacing the previous contents.</td>
</tr>
<tr>
<td>setConstraints(int constraints)</td>
<td>Sets the input constraints of the TextField.</td>
</tr>
<tr>
<td>setInitialInputMode(String characterSubset)</td>
<td>Sets a hint to the implementation as to the input mode that should be used when the user initiates editing of this TextField.</td>
</tr>
</tbody>
</table>
### User Interface Methods

#### TextField

- **setMaxSize(int maxSize)**
  
  Sets the maximum size (number of characters) that can be contained in this TextField.

- **setString(String text)**
  
  Sets the contents of the TextField as a string value, replacing the previous contents.

- **size()**
  
  Gets the number of characters that are currently stored in this TextField.

#### Methods inherited from class `javax.microedition.lcdui.Item`

- `addCommand`, `getLabel`, `getLayout`, `getMinimumHeight`, `getMinimumWidth`, `getPreferredSizeHeight`, `getPreferredSizeWidth`, `notifyStateChanged`, `removeCommand`, `setDefaultCommand`, `setItemCommandListener`, `setLabel`, `setLayout`, `setPreferredSize`

#### Methods inherited from class `Object`

- `equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

### Fields

#### ANY

- **ANY**
  
  The user is allowed to enter any text. Line breaks may be entered.

  Constant 0 is assigned to ANY.

  Constant value: 0

#### CONSTRAINT_MASK

- **CONSTRAINT_MASK**
  
  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 0xFF is assigned to CONSTRAINT_MASK.

  Constant value: 65535

#### DECIMAL

- **DECIMAL**
  
  public static final int DECIMAL
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.
Constant value: 5

Since: MIDP 2.0

---

EMAILADDR

public static final int EMAILADDR

The user is allowed to enter an e-mail address.

Constant 1 is assigned to EMAILADDR.
Constant value: 1

---

INITIAL_CAPS_SENTENCE

public static final int INITIAL_CAPS_SENTENCE

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.
Constant value: 2097152

Since: MIDP 2.0

---

INITIAL_CAPS_WORD

public static final int INITIAL_CAPS_WORD
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.
Constant value: \(1048576\)

Since: MIDP 2.0

---

**NON_PREDICTIVE**

```java
class NON_PREDICTIVE {
    public static final int NON_PREDICTIVE = 0x80000;
    Constant value: 524288
}
```

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.
Constant value: \(524288\)

Since: MIDP 2.0

---

**NUMERIC**

```java
class NUMERIC {
    public static final int NUMERIC = 2;
    Constant value: 2
}
```

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(java.lang.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.
Constant value: 2

---

**PASSWORD**

```java
class PASSWORD {
    public static final int PASSWORD = 2;
}
```

Since MIDP 2.0
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.
Constant value: 65536

---

PHONENUMBER

public static final int PHONENUMBER

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 method.

Constant 3 is assigned to PHONENUMBER.
Constant value: 3

---

SENSITIVE

public static final int SENSITIVE

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.
Constant value: 262144

Since: MIDP 2.0

---

UNEDITABLE

public static final int UNEDITABLE

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.
Constant value: 131072
Since: MIDP 2.0

**URL**

```java
public static final int URL
```

The user is allowed to enter a URL.

Constant 4 is assigned to URL.
Constant value: 4

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
</table>

**TextField**

```java
public TextField(String label,
                 String text,
                 int maxSize,
                 int constraints)
```

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

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
</table>

**delete**

```java
public void delete(int offset,
                   int length)
```

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**

```java
public int getCaretPosition()
```

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**

```java
public int getChars(char[] data)
```

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:** `TextField.setChars(char[], int, int)`

**getConstraints**

```java
public int getConstraints()
```

Gets the current input constraints of the TextField.

**Returns:**
- the current constraints value (see input constraints)

**See Also:** `TextField.setConstraints(int)`

**getMaxSize**

```java
public int getMaxSize()
```

Returns the maximum size (number of characters) that can be stored in this TextField.

**Returns:**
- the maximum size in characters

**See Also:** `TextField.setMaxSize(int)`

**getString**

```java
public java.lang.String getString()
```

Gets the contents of the TextField as a string value.

**Returns:**
- the current contents

**See Also:** `TextField.setString(String)`
public void insert(char[] data,  
    int offset,  
    int length,  
    int position)

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).

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

public void insert(String src,  
    int position)

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() 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

public void setChars(char[] data,  
    int offset,  
    int length)
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: TextField.getChars(char[])

**setConstraints**

public void setConstraints(int constraints)

Sets the input constraints of the TextField. If 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: TextField.getConstraints()

**setInitialInputMode**

public void setInitialInputMode(String characterSubset)

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**

public int setMaxSize(int maxSize)

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: `TextField.getMaxSize()`

**setString**

```java
class TextField {
    public void setString(String text) {
        // 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: `TextField.getString()`

**size**

```java
class TextField {
    public int size() {
        // Gets the number of characters that are currently stored in this TextField.
        // Returns:
        // number of characters in the TextField
    }
}
```
javax.microedition.lcdui

Ticker

Declaration

```java
public class 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()` 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

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public Ticker(String str)</code></td>
<td>Constructs a new Ticker object, given its initial contents string.</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>java.lang.String getString()</code></td>
<td>Gets the string currently being scrolled by the ticker.</td>
</tr>
<tr>
<td><code>void setString(String str)</code></td>
<td>Sets the string to be displayed by this ticker.</td>
</tr>
</tbody>
</table>

Methods inherited from class Object

- `equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Constructors
**Ticker**

```java
public Ticker(String str)
```

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**

```java
public java.lang.String getString()
```

Gets the string currently being scrolled by the ticker.

**Returns:**
- string of the ticker

**See Also:** Ticker.setString(String)

**setString**

```java
public void setString(String str)
```

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:** Ticker.getString()
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.

9.1 API Overview
The API is comprised of five classes:

9.1.1 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.

9.1.2 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.

9.1.3 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.

9.1.4 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.

9.1.5 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 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

<table>
<thead>
<tr>
<th>Classes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GameCanvas</strong></td>
<td>The GameCanvas class provides the basis for a game user interface.</td>
</tr>
<tr>
<td><strong>Layer</strong></td>
<td>A Layer is an abstract class representing a visual element of a game.</td>
</tr>
<tr>
<td><strong>LayerManager</strong></td>
<td>The LayerManager manages a series of Layers.</td>
</tr>
<tr>
<td><strong>Sprite</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>TiledLayer</strong></td>
<td>A TiledLayer is a visual element composed of a grid of cells that can be filled with a set of tile images.</td>
</tr>
</tbody>
</table>
GameCanvas

Declaration

```java
public abstract class GameCanvas extends javax.microedition.lcdui.Canvas
```

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` and `getHeight`.

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 typical 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

<table>
<thead>
<tr>
<th>Field Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td>The bit representing the DOWN key.</td>
</tr>
<tr>
<td>Value:</td>
</tr>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td>The bit representing the FIRE key.</td>
</tr>
<tr>
<td>Value:</td>
</tr>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td>The bit representing the GAME_A key (may not be supported on all devices).</td>
</tr>
<tr>
<td>Value:</td>
</tr>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td>The bit representing the GAME_B key (may not be supported on all devices).</td>
</tr>
<tr>
<td>Value:</td>
</tr>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td>The bit representing the GAME_C key (may not be supported on all devices).</td>
</tr>
<tr>
<td>Value:</td>
</tr>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td>The bit representing the GAME_D key (may not be supported on all devices).</td>
</tr>
<tr>
<td>Value:</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
</tbody>
</table>

Fields inherited from class [javax.microedition.lcdui.Canvas](https://developer.mozilla.org/en-US/docs/Web/API/javax/microedition/lcdui/Canvas)

DOWN, FIRE, GAME_A, GAME_B, GAME_C, GAME_D, KEY_NUM0, KEY_NUM1, KEY_NUM2, KEY_NUM3, KEY_NUM4, KEY_NUM5, KEY_NUM6, KEY_NUM7, KEY_NUM8, KEY_NUM9, KEY_POUND, KEY_STAR, LEFT, RIGHT, UP

Constructor Summary

```java
protected GameCanvas(boolean suppressKeyEvents)

Creates a new instance of a GameCanvas.
```

Method Summary

```java
void flushGraphics()

Flushes the off-screen buffer to the display.

void flushGraphics(int x, int y, int width, int height)

Flushes the specified region of the off-screen buffer to the display.

javax.microedition.lcdui.Graphics getGraphics()

Obtains the Graphics object for rendering a GameCanvas.

int getKeyStates()

Gets the states of the physical game keys.

void paint(Graphics g)

Paints this GameCanvas.
```


gameAction, getHeight, getKeyCode, getKeyName, getWidth, hasPointerEvents, hasPointerMotionEvents, hasRepeatEvents, hideNotify, isDoubleBuffered, keyPressed, keyReleased, keyRepeated, paint, pointerDragged, pointerPressed, pointerReleased, repaint, repaint, serviceRepaints, setFullScreenMode, showNotify, sizeChanged


addCommand, getHeight, getTicker, getTitle, getWidth, isShown, removeCommand, setCommandListener, setTicker, setTitle, sizeChanged


equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Fields
DOWN_PRESSED

public static final int DOWN_PRESSED

The bit representing the DOWN key. This constant has a value of 0x0040 (1 << Canvas.DOWN).
Constant value: 64

FIRE_PRESSED

public static final int FIRE_PRESSED

The bit representing the FIRE key. This constant has a value of 0x0100 (1 << Canvas.FIRE).
Constant value: 256

GAME_A_PRESSED

public static final int GAME_A_PRESSED

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).
Constant value: 512

GAME_B_PRESSED

public static final int GAME_B_PRESSED

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).
Constant value: 1024

GAME_C_PRESSED

public static final int GAME_C_PRESSED

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).
Constant value: 2048

GAME_D_PRESSED

public static final int GAME_D_PRESSED

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).
Constant value: 4096

LEFT_PRESSED

public static final int LEFT_PRESSED

The bit representing the LEFT key. This constant has a value of 0x0004 (1 << Canvas.LEFT).
Constant value: 4

RIGHT_PRESSED

public static final int RIGHT_PRESSED

The bit representing the RIGHT key. This constant has a value of 0x0020 (1 << Canvas.RIGHT).
Constant value: 32
**UP_PRESSED**

```java
public static final int UP_PRESSED
```

The bit representing the UP key. This constant has a value of 0x0002 (1 << Canvas.UP).

**Constant value: 2**

### Constructors

**GameCanvas**

```java
protected GameCanvas(boolean suppressKeyEvents)
```

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**

```java
public void flushGraphics()
```

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:** GameCanvas.flushGraphics(int, int, int, int)

**flushGraphics**

```java
public void flushGraphics(int x,
                           int y,
                           int width,
                           int height)
```
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:** GameCanvas.flushGraphics()

---

**getGraphics**

```java
protected javax.microedition.lcdui.Graphics getGraphics()
```

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();
- the stroke style is SOLID; 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:** GameCanvas.flushGraphics(), GameCanvas.flushGraphics(int, int, int, int)

---

**getKeyStates**

```java
public int getKeyStates()
```
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:

```java
// 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 Displayable.isShown(). 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:** GameCanvas.UP_PRESSED, GameCanvas.DOWN_PRESSED, GameCanvas.LEFT_PRESSED, GameCanvas.RIGHT_PRESSED, GameCanvas.FIRE_PRESSED, GameCanvas.GAME_A_PRESSED, GameCanvas.GAME_B_PRESSED, GameCanvas.GAME_C_PRESSED, GameCanvas.GAME_D_PRESSED

---

**paint**

```java
public void paint(Graphics g)
```

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.

**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:
  javax.microedition.lcdui.game.Sprite,
  javax.microedition.lcdui.game.TiledLayer

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 Layer.paint(Graphics) 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

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>int getHeight()</td>
</tr>
<tr>
<td>Gets the current height of this layer, in pixels.</td>
</tr>
<tr>
<td>int getWidth()</td>
</tr>
<tr>
<td>Gets the current width of this layer, in pixels.</td>
</tr>
<tr>
<td>int getX()</td>
</tr>
<tr>
<td>Gets the horizontal position of this Layer's upper-left corner in the painter's coordinate system.</td>
</tr>
<tr>
<td>int getY()</td>
</tr>
<tr>
<td>Gets the vertical position of this Layer's upper-left corner in the painter's coordinate system.</td>
</tr>
<tr>
<td>boolean isVisible()</td>
</tr>
<tr>
<td>Gets the visibility of this Layer.</td>
</tr>
<tr>
<td>void move(int dx, int dy)</td>
</tr>
<tr>
<td>Moves this Layer by the specified horizontal and vertical distances.</td>
</tr>
<tr>
<td>abstract void paint(Graphics g)</td>
</tr>
<tr>
<td>Paints this Layer if it is visible.</td>
</tr>
<tr>
<td>void setPosition(int x, int y)</td>
</tr>
<tr>
<td>Sets this Layer's position such that its upper-left corner is located at (x,y) in the painter's coordinate system.</td>
</tr>
<tr>
<td>void setVisible(boolean visible)</td>
</tr>
<tr>
<td>Sets the visibility of this Layer.</td>
</tr>
</tbody>
</table>
### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>getHeight</strong></td>
<td>Gets the current height of this layer, in pixels.</td>
</tr>
<tr>
<td><strong>getWidth</strong></td>
<td>Gets the current width of this layer, in pixels.</td>
</tr>
<tr>
<td><strong>getX</strong></td>
<td>Gets the horizontal position of this Layer's upper-left corner in the painter's coordinate system.</td>
</tr>
<tr>
<td><strong>getY</strong></td>
<td>Gets the vertical position of this Layer's upper-left corner in the painter's coordinate system.</td>
</tr>
<tr>
<td><strong>isVisible</strong></td>
<td>Gets the visibility of this Layer.</td>
</tr>
</tbody>
</table>

Methods inherited from class `Object`:
- equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### See Also:
- `Layer.getWidth()`
- `Layer.getHeight()`
- `Layer.getX()`, `Layer.setPosition(int, int)`, `Layer.move(int, int)`
- `Layer.getY()`, `Layer.setPosition(int, int)`, `Layer.move(int, int)`
- `isVisible()`
See Also: `Layer.setVisible(boolean)`

### move

```java
public void move(int dx,
    int dy)
```

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: `Layer.setPosition(int, int), Layer.getX(), Layer.getY()`

### paint

```java
public abstract void paint(Graphics g)
```

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

```java
public void setPosition(int x,
    int y)
```

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: `Layer.move(int, int), Layer.getX(), Layer.getY()`

### setVisible

```java
public void setVisible(boolean visible)
```

Sets the visibility of this Layer. A visible Layer is rendered when its `Layer.paint(Graphics)` method is called; an invisible Layer is not rendered.

**Parameters:**
- `visible` - true to make the Layer visible, false to make it invisible

See Also: `Layer.isVisible()`
javax.microedition.lcdui.game
LayerManager

Declaration

class 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 LayerManager.paint(Graphics, int, int) 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

**Constructor Summary**

<table>
<thead>
<tr>
<th>Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public LayerManager()</td>
</tr>
<tr>
<td>Creates a new LayerManager.</td>
</tr>
</tbody>
</table>

**Method Summary**

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>void append(Layer l)</td>
</tr>
<tr>
<td>Appends a Layer to this LayerManager.</td>
</tr>
<tr>
<td>javax.microedition.lcdui.game.Layer getLayerAt(int index)</td>
</tr>
<tr>
<td>Gets the Layer with the specified index.</td>
</tr>
<tr>
<td>int getSize()</td>
</tr>
<tr>
<td>Gets the number of Layers in this LayerManager.</td>
</tr>
<tr>
<td>void insert(Layer l, int index)</td>
</tr>
<tr>
<td>Inserts a new Layer in this LayerManager at the specified index.</td>
</tr>
<tr>
<td>void paint(Graphics g, int x, int y)</td>
</tr>
<tr>
<td>Renders the LayerManager's current view window at the specified location.</td>
</tr>
<tr>
<td>void remove(Layer l)</td>
</tr>
<tr>
<td>Removes the specified Layer from this LayerManager.</td>
</tr>
<tr>
<td>void setViewWindow(int x, int y, int width, int height)</td>
</tr>
<tr>
<td>Sets the view window on the LayerManager.</td>
</tr>
</tbody>
</table>

**Methods inherited from class Object**

<table>
<thead>
<tr>
<th>Methods inherited from class Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait</td>
</tr>
</tbody>
</table>
Constructors

LayerManager

public LayerManager()

Creates a new LayerManager.

Methods

append

public void append(Layer l)

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:
1 - the Layer to be added

Throws:
NullPointerException - if the Layer is null

See Also: LayerManager.insert(Layer, int), LayerManager.remove(Layer)

getLayerAt

public javax.microedition.lcdui.game.Layer getLayerAt(int index)

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

public int getSize()

Gets the number of Layers in this LayerManager.

Returns:
the number of Layers

insert

public void insert(Layer l, int index)

Inserts a new Layer in this LayerManager at the specified index. Inserting a Layer that was previously added to this LayerManager is equivalent to first removing it with LayerManager.remove() and then adding it with the insert() method at the specified index. Conditions for throwing IndexOutOfBoundsException are checked before LayerManager.remove() is called.

Parameters:
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 If the index is greater than the number of Layers already added to this LayerManager and the Layer has not been added to this LayerManager yet If the index is greater than the number of Layers already added to this LayerManager minus one and the Layer has already been added to this LayerManager.

See Also: LayerManager.append(Layer), LayerManager.remove(Layer)

paint

public void paint(Graphics g,  
    int x,  
    int y)

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: LayerManager.setViewWindow(int, int, int, int)

remove

public void remove(Layer l)

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: LayerManager.append(Layer), LayerManager.insert(Layer, int)
setViewWindow

public void setViewWindow(int x, int y, int width, int height)

Sets the view window on the LayerManager.

The view window specifies the region that the LayerManager draws when its
LayerManager.paint(Graphics, int, int) 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 Layer

Object

javax.microedition.lcdui.game.Layer

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 Sprite.getRawFrameCount() 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 \texttt{Sprite.setFrame(int)} , \texttt{Sprite.prevFrame()} , or \texttt{Sprite.nextFrame()} . 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 \texttt{Sprite.nextFrame()} 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 \texttt{setPosition(x,y)} , \texttt{getX()} , and \texttt{getY()} . 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 \textit{reference pixel}. The reference pixel is defined by specifying its location in the Sprite's untransformed frame using \texttt{defineReferencePixel(x,y)} . 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()** and **getRefPixelY()** 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)** 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)`.

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 `Sprite.getRefPixelX()` and `Sprite.getRefPixelY()` remain the same; however, the values returned by `getX()` and `getY()` 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 `Sprite.paint(Graphics)` 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

### Field Summary

<table>
<thead>
<tr>
<th>public static final</th>
<th>TRANS_MIRROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Causes the Sprite to appear reflected about its vertical center.</td>
<td></td>
</tr>
<tr>
<td>Value: 2</td>
<td></td>
</tr>
</tbody>
</table>
public static final TRANS_MIRROR_ROT180
Causes the Sprite to appear reflected about its vertical center and then rotated clockwise by 180 degrees.
Value: 1

public static final TRANS_MIRROR_ROT270
Causes the Sprite to appear reflected about its vertical center and then rotated clockwise by 270 degrees.
Value: 4

public static final TRANS_MIRROR_ROT90
Causes the Sprite to appear reflected about its vertical center and then rotated clockwise by 90 degrees.
Value: 7

public static final TRANS_NONE
No transform is applied to the Sprite.
Value: 0

public static final TRANS_ROT180
Causes the Sprite to appear rotated clockwise by 180 degrees.
Value: 3

public static final TRANS_ROT270
Causes the Sprite to appear rotated clockwise by 270 degrees.
Value: 6

public static final TRANS_ROT90
Causes the Sprite to appear rotated clockwise by 90 degrees.
Value: 5

Constructor Summary
public Sprite(Image image)
Creates a new non-animated Sprite using the provided Image.

public Sprite(Image image, int frameWidth, int frameHeight)
Creates a new animated Sprite using frames contained in the provided Image.

public Sprite(Sprite s)
Creates a new Sprite from another Sprite.

Method Summary
boolean collidesWith(Image image, int x, int y, boolean pixelLevel)
Checks for a collision between this Sprite and the specified Image with its upper left corner at the specified location.

boolean collidesWith(Sprite s, boolean pixelLevel)
Checks for a collision between this Sprite and the specified Sprite.

boolean collidesWith(TiledLayer t, boolean pixelLevel)
Checks for a collision between this Sprite and the specified TiledLayer.

void defineCollisionRectangle(int x, int y, int width, int height)
Defines the Sprite's bounding rectangle that is used for collision detection purposes.
void defineReferencePixel(int x, int y)
    Defines the reference pixel for this Sprite.

int getFrame()
    Gets the current index in the frame sequence.

int getFrameSequenceLength()
    Gets the number of elements in the frame sequence.

int getRawFrameCount()
    Gets the number of raw frames for this Sprite.

int getRefPixelX()
    Gets the horizontal position of this Sprite's reference pixel in the painter's
    coordinate system.

int getRefPixelY()
    Gets the vertical position of this Sprite's reference pixel in the painter's
    coordinate system.

void nextFrame()
    Selects the next frame in the frame sequence.

void paint(Graphics g)
    Draws the Sprite.

void prevFrame()
    Selects the previous frame in the frame sequence.

void setFrame(int sequenceIndex)
    Selects the current frame in the frame sequence.

void setFrameSequence(int[] sequence)
    Set the frame sequence for this Sprite.

void setImage(Image img, int frameWidth, int frameHeight)
    Changes the Image containing the Sprite's frames.

void setRefPixelPosition(int x, int y)
    Sets this Sprite's position such that its reference pixel is located at (x,y) in the
    painter's coordinate system.

void setTransform(int transform)
    Sets the transform for this Sprite.

Methods inherited from class javax.microedition.lcdui.game.Layer

getHeight, getWidth, getX, getY, isVisible, move, paint, setPosition, setVisible

Methods inherited from class Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Fields

TRANS_MIRROR

public static final int TRANS_MIRROR
Causes the Sprite to appear reflected about its vertical center. This constant has a value of 2.
Constant value: 2

**TRANS_MIRROR_ROT180**

```java
public static final int TRANS_MIRROR_ROT180
```

Causes the Sprite to appear reflected about its vertical center and then rotated clockwise by 180 degrees. This constant has a value of 1.
Constant value: 1

**TRANS_MIRROR_ROT270**

```java
public static final int TRANS_MIRROR_ROT270
```

Causes the Sprite to appear reflected about its vertical center and then rotated clockwise by 270 degrees. This constant has a value of 4.
Constant value: 4

**TRANS_MIRROR_ROT90**

```java
public static final int TRANS_MIRROR_ROT90
```

Causes the Sprite to appear reflected about its vertical center and then rotated clockwise by 90 degrees. This constant has a value of 7.
Constant value: 7

**TRANS_NONE**

```java
public static final int TRANS_NONE
```

No transform is applied to the Sprite. This constant has a value of 0.
Constant value: 0

**TRANS_ROT180**

```java
public static final int TRANS_ROT180
```

Causes the Sprite to appear rotated clockwise by 180 degrees. This constant has a value of 3.
Constant value: 3

**TRANS_ROT270**

```java
public static final int TRANS_ROT270
```

Causes the Sprite to appear rotated clockwise by 270 degrees. This constant has a value of 6.
Constant value: 6

**TRANS_ROT90**

```java
public static final int TRANS_ROT90
```

Causes the Sprite to appear rotated clockwise by 90 degrees. This constant has a value of 5.
Constant value: 5

**Constructors**

```java
public Sprite(Image image)
```
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**

```java
public Sprite(Image image,
              int frameWidth,
              int frameHeight)
```

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()` and `Layer.getHeight()` 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 `Sprite.setFrameSequence(int[])`.

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**

```java
public Sprite(Sprite s)
```

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

public final boolean collidesWith(Image image,
        int x,
        int y,
        boolean pixelLevel)

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

public final boolean collidesWith(Sprite s,
        boolean pixelLevel)

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

public final boolean collidesWith(TiledLayer t, boolean pixelLevel)

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

public void defineCollisionRectangle(int x, int y, int width, int height)

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

public void defineReferencePixel(int x, int y)
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 \(\text{getX}()\) and \(\text{getY}()\) 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:** Sprite.setRefPixelPosition(int, int), Sprite.getRefPixelX(), Sprite.getRefPixelY()

---

**getFrame**

```java
public final int getFrame()
```

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:** Sprite setFrameSequence(int[]), Sprite setFrame(int)

---

**getFrameSequenceLength**

```java
public int getFrameSequenceLength()
```

 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:** Sprite.getRawFrameCount()

---

**getRawFrameCount**

```java
public int getRawFrameCount()
```

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:** Sprite.getFrameSequenceLength()

---

**getRefPixelX**

```java
public int getRefPixelX()
```

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: Sprite.defineReferencePixel(int, int), Sprite.setRefPixelPosition(int, int), Sprite.getRefPixelY()

---

getRefPixelY

public int getRefPixelY()

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: Sprite.defineReferencePixel(int, int), Sprite.setRefPixelPosition(int, int), Sprite.getRefPixelX()

---

nextFrame

public void nextFrame()

Selects the next frame in the frame sequence.

The frame sequence is considered to be circular, i.e. if Sprite.nextFrame() is called when at the end of the sequence, this method will advance to the first entry in the sequence.

See Also: Sprite.setFrameSequence(int[]), Sprite.prevFrame()

---

paint

public final void paint(Graphics g)

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() and Layer.getY().

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.

Parameters:
  g - the graphics object to draw Sprite on

Throws:
  NullPointerException - if g is null

---

prevFrame

public void prevFrame()

Selects the previous frame in the frame sequence.

The frame sequence is considered to be circular, i.e. if Sprite.prevFrame() is called when at the start of the sequence, this method will advance to the last entry in the sequence.

See Also: Sprite.setFrameSequence(int[]), Sprite.nextFrame()

---

setFrame

public void setFrame(int sequenceIndex)
Selects the current frame in the frame sequence.

The current frame is rendered when `Sprite.paint(Graphics)` 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 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:** `Sprite.setFrameSequence(int[]), Sprite.getFrame()`

```java
public void setFrameSequence(int[] sequence)
```

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 `Sprite.getRawFrameCount()`
- `IllegalArgumentException` - if the array has less than 1 element

**See Also:** `Sprite.nextFrame(), Sprite.prevFrame(), Sprite.setFrame(int), Sprite.getFrame()`

```java
public void setImage(Image img, int frameWidth, int frameHeight)
```
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 `javax.microedition.lcdui.game.Sprite(Image, int, int)` for information on how the frames are created from the image. The values returned by `Layer.getWidth()` and `Layer.getHeight()` 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 `Sprite setFrameSequence(int [])`), 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 the 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

```java
public void setRefPixelPosition(int x,
    int y)
```

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:
- `Sprite.defineReferencePixel(int, int), Sprite.getRefPixelX()`,
- `Sprite.getRefPixelY()`

---

### setTransform

```java
public void setTransform(int transform)
```
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 Sprite.TRANS_NONE.

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() and Layer.getHeight() 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 Sprite.getRefPixelX() and Sprite.getRefPixelY() will be the same both before and after the transform is applied, but the values returned by getX() and getY() may change.

**Parameters:**

transform - the desired transform for this Sprite

**Throws:**

IllegalArgumentException - if the requested transform is invalid

**See Also:** Sprite.TRANS_NONE, Sprite.TRANS_ROT90, Sprite.TRANS_ROT180, Sprite.TRANS_ROT270, Sprite.TRANS_MIRROR, Sprite.TRANS_MIRROR_ROT90, Sprite.TRANS_MIRROR_ROT180, Sprite.TRANS_MIRROR_ROT270
javax.microedition.lcdui.game

TiledLayer

Declaration

public class TiledLayer extends Layer

Object

Object

javax.microedition.lcdui.game.Layer

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 TiledLayer.setStaticTileSet(Image, int, int) 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 TiledLayer.createAnimatedTile(int) 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 TiledLayer.setAnimatedTile(int, int) 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 TiledLayer.setCell(int, int, int) and TiledLayer.fillCells(int, int, int, int, int). 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.

```
Cells

0 0 1 1 0 0 0 0 0 0 0 0
0 1 4 4 3 0 0 0 1 2 2
1 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 = 5
```

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 1 0 0 0 0 0 0 0 0
0 1 4 4 3 0 0 0 1 2 2
1 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

### Constructor Summary

| public TiledLayer(int columns, int rows, Image image, int tileWidth, int tileHeight) |
| Creates a new TiledLayer. |

### Method Summary

| int createAnimatedTile(int staticTileIndex) |
| Creates a new animated tile and returns the index that refers to the new animated tile. |

| void fillCells(int col, int row, int numCols, int numRows, int tileIndex) |
| Fills a region cells with the specific tile. |

| int getAnimatedTile(int animatedTileIndex) |
| Gets the tile referenced by an animated tile. |

| int getCell(int col, int row) |
| Gets the contents of a cell. |

| int getCellHeight() |
| Gets the height of a single cell, in pixels. |

| int getCellWidth() |
| Gets the width of a single cell, in pixels. |

| int getColumns() |
| Gets the number of columns in the TiledLayer grid. |

| int getRows() |
| Gets the number of rows in the TiledLayer grid. |

| void paint(Graphics g) |
| Draws the TiledLayer. |

| void setAnimatedTile(int animatedTileIndex, int staticTileIndex) |
| Associates an animated tile with the specified static tile. |

| void setCell(int col, int row, int tileIndex) |
| Sets the contents of a cell. |

| void setStaticTileSet(Image image, int tileWidth, int tileHeight) |
| Change the static tile set. |
Methods inherited from class `javax.microedition.lcdui.game.Layer`

- getHeight, getWidth, getX, getY, isVisible, move, paint, setPosition, setVisible

Methods inherited from class `Object`

- equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

**TiledLayer**

```java
public TiledLayer(int columns,
    int rows,
    Image image,
    int tileWidth,
    int tileHeight)
```

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 `TiledLayer.setCell(int, int, int)` and `TiledLayer.fillCells(int, int, int, int, int)`.

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 `TiledLayer.setStaticTileSet(Image, int, int)`. 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**

```java
public int createAnimatedTile(int staticTileIndex)
```

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)
fillCells

public void fillCells(int col,
          int row,
          int numCols,
          int numRows,
          int tileIndex)

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: TiledLayer.setCell(int, int, int), TiledLayer.getCell(int, int)

getAnimatedTile

public int getAnimatedTile(int animatedTileIndex)

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: TiledLayer.setAnimatedTile(int, int)

cell

public int cell(int col,
               int row)

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: TiledLayer.setCell(int, int, int), TiledLayer.fillCells(int, int, int, int, int)

getCellHeight

public final int getCellHeight()

Gets the height of a single cell, in pixels.

Returns:
the height in pixels of a single cell in the TiledLayer grid

getCellWidth

public final int getCellWidth()

Gets the width of a single cell, in pixels.

Returns:
the width in pixels of a single cell in the TiledLayer grid

getColumns

public final int getColumns()

Gets the number of columns in the TiledLayer grid. The overall width of the TiledLayer, in pixels, may be obtained by calling Layer.getWidth().

Returns:
the width in columns of the TiledLayer grid

getRows

public final int getRows()

Gets the number of rows in the TiledLayer grid. The overall height of the TiledLayer, in pixels, may be obtained by calling Layer.getHeight().

Returns:
the height in rows of the TiledLayer grid

paint

public final void paint(Graphics g)

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() and Layer.getY(). 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.

Parameters:
g - the graphics object to draw the TiledLayer

Throws:
NullPointerException - if g is null
setAnimatedTile

public void setAnimatedTile(int animatedTileIndex, int staticTileIndex)

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: TiledLayer.getAnimatedTile(int)

setCell

public void setCell(int col, int row, int tileIndex)

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: TiledLayer.getCell(int, int), TiledLayer.fillCells(int, int, int, int, int)

setStaticTileSet

public void setStaticTileSet(Image image, int tileWidth, int tileHeight)

Change the static tile set.

Replaces the current static tile set with a new static tile set. See the constructor
javax.microedition.lcdui.game.TiledLayer(int, int, Image, int, int) 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
10.1 Introduction
J2METM 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 2.1 Media API

The first API is intended for J2METM 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.x). Furthermore, this subset API can be adopted to other J2METM profiles requiring sound support. In the following, a more detailed description of the background and requirements of the building block API is given.

10.2 Background of the Media API
Some J2METM 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 building 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 2.1 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 2.1 one needs to only implement the additional classes and methods from that API. In a compliant implementation, the MIDP subset of MMAPI (classes from the javax.microedition.media package) MUST comply with the MMAPI specification version 1.1 (or later).

10.2.1 Basic Concepts

The proposed audio building block system consists 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.

10.2.2 API Details

The createPlayer method is the top-level entry point to the API:

\[
\text{Player Manager.createPlayer(String url)}
\]

The url fully specifies the protocol and the content of the data:

\[
<\text{protocol}>:<\text{content location}>
\]

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:

\[
\text{Player.realize()}
\]
\[
\text{Player.prefetch()}
\]
\[
\text{Player.start()}
\]

Fine-grained control is an important feature of the API; therefore, each Player also provides type-specific
controls with the `getControls` and `getControl` methods:

```java
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.

### 10.2.3 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:

```java
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:

```java
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.

### 10.2.4 Usage Scenarios

In this section we demonstrate how the API could be used in four common scenarios.

#### 10.2.4.1 Scenario 1: Single-Tone Generation

```java
try {
    Manager.playTone(ToneControl.C4, 5000 /* ms */, 100 /* max vol */);
} catch (MediaException e) {
}
```

#### 10.2.4.2 Scenario 2: Simple Media Playback with Looping

Notice that in MIDP 2.1 the wav format is mandatory only in a case the device supports sampled audio.

```java
try {
    Player p = Manager.createPlayer("http://webserver/music.wav");
    p.setLoopCount(5);
    p.start();
} catch (IOException ioe) {
} catch (MediaException me) {
}
```

#### 10.2.4.3 Scenario 3: Playing Back from Media Stored in JAR

Notice that in MIDP 2.1 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) { }

10.2.4.4 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, 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

<table>
<thead>
<tr>
<th>Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
</tr>
<tr>
<td>Controllable</td>
</tr>
<tr>
<td>Player</td>
</tr>
<tr>
<td>PlayerListener</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manager</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>MediaException</td>
</tr>
</tbody>
</table>
javax.microedition.media

Control

Declaration

public interface Control

All Subinterfaces:

javax.microedition.media.control.ToneControl,
javax.microedition.media.control.VolumeControl

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.

Controls 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 Controls 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 Controls.

See Also: Controllable, Player
javax.microedition.media

Controllable

Declaration

public interface Controllable

All Subinterfaces:

javax.microedition.media.Player

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.

Method Summary

| javax.microedition.media.Control | getControl(String controlType) |
|----------------------------------|---------------------------------
|                                 | Obtain the object that implements the specified Control interface. |
| javax.microedition.media.Control[] | getControls() |
|                                 | Obtain the collection of Controls from the object that implements this interface. |

Methods

getControl

public javax.microedition.media.Control getControl(String controlType)

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 Controls 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 - Thrown if getControl is called in a wrong state.

See Player for more details.

getControls

public javax.microedition.media.Control[] getControls()
Obtain the collection of Controls 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:

```java
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` - 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/); 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 which is defined in the form: <scheme>:<scheme-specific-part>The "scheme" part of the locator string identifies the name of the protocol being used to deliver the data.
### Field Summary

<table>
<thead>
<tr>
<th>Access</th>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final</td>
<td>java.lang.String</td>
<td>TONE_DEVICE_LOCATOR</td>
<td>The locator to create a tone Player to play back tone sequences. Value: <code>device://tone</code></td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static javax.microedition.media.Player createPlayer(InputStream stream, String type)</td>
<td>Create a Player to play back media from an InputStream.</td>
</tr>
<tr>
<td>static javax.microedition.media.Player createPlayer(String locator)</td>
<td>Create a Player from an input locator.</td>
</tr>
<tr>
<td>static java.lang.String[] getSupportedContentTypes(String protocol)</td>
<td>Return the list of supported content types for the given protocol.</td>
</tr>
<tr>
<td>static java.lang.String[] getSupportedProtocols(String content_type)</td>
<td>Return the list of supported protocols given the content type.</td>
</tr>
<tr>
<td>static void playTone(int note, int duration, int volume)</td>
<td>Play back a tone as specified by a note and its duration.</td>
</tr>
</tbody>
</table>

### Methods inherited from class Object

- equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### Fields

**TONE_DEVICE_LOCATOR**

public static final java.lang.String TONE_DEVICE_LOCATOR
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 locato 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.

Constant value: device://tone

## Methods

**createPlayer**

```java
public static javax.microedition.media.Player createPlayer(InputStream stream,
    String type)
throws java.io.IOException,
    javax.microedition.media.MediaException
```

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 Content Type of the media.

**Returns:**
- A new Player.

**Throws:**
- IllegalArgumentException - Thrown if stream is null.
- MediaException - Thrown if a Player cannot be created for the given stream and type.
- 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**

```java
public static javax.microedition.media.Player createPlayer(String locator)
throws java.io.IOException,
    javax.microedition.media.MediaException
```

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 - Thrown if a Player cannot be created for the given locator.
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

public static java.lang.String[] getSupportedContentTypes(String protocol)

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

public static java.lang.String[] getSupportedProtocols(String content_type)

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

public static void playTone(int note, int duration, int volume)
throws javax.microedition.media.MediaException
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:

\[
\text{SEMITONE\_CONST} = 17.31234049066755 = \frac{1}{(\ln(2^{\frac{1}{12}}))}
\]

\[
\text{note} = \ln(\text{freq}/8.176) \times \text{SEMITONE\_CONST}
\]

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** - 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.

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public MediaException()</td>
<td>Constructs a MediaException with null as its error detail message.</td>
</tr>
<tr>
<td>public MediaException(String reason)</td>
<td>Constructs a MediaException with the specified detail message. The error message string can later be retrieved by the Throwable.getMessage() method of class java.lang.Throwable.</td>
</tr>
</tbody>
</table>

Methods inherited from class Throwable

getMessage, printStackTrace, toString

Methods inherited from class Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

MediaException

public MediaException()

        Constructs a MediaException with null as its error detail message.

MediaException

public MediaException(String reason)

        Constructs a MediaException with the specified detail message. The error message string 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

All Superinterfaces:
javax.microedition.media.Controllable

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
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.

**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 occurred. 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) {
}
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>long getMediaTime()</code></td>
<td>Gets this Player's current media time.</td>
</tr>
<tr>
<td><code>int getState()</code></td>
<td>Gets the current state of this Player.</td>
</tr>
<tr>
<td><code>void prefetch()</code></td>
<td>Acquires the scarce and exclusive resources and processes as much data as necessary to reduce the start latency.</td>
</tr>
<tr>
<td><code>void realize()</code></td>
<td>Constructs portions of the Player without acquiring the scarce and exclusive resources.</td>
</tr>
<tr>
<td><code>void removePlayerListener(PlayerListener playerListener)</code></td>
<td>Remove a player listener for this player.</td>
</tr>
<tr>
<td><code>void setLoopCount(int count)</code></td>
<td>Set the number of times the Player will loop and play the content.</td>
</tr>
<tr>
<td><code>long setMediaTime(long now)</code></td>
<td>Sets the Player's media time.</td>
</tr>
<tr>
<td><code>void start()</code></td>
<td>Starts the Player as soon as possible.</td>
</tr>
<tr>
<td><code>void stop()</code></td>
<td>Stops the Player.</td>
</tr>
</tbody>
</table>

Methods inherited from interface `javax.microedition.media.Controllable`  
`getControl`, `getControls`

### Fields

**CLOSED**

```java
public static final int CLOSED
```

The state of the Player indicating that the Player is closed.

Value 0 is assigned to `CLOSED`.
Constant value: 0

**PREFETCHED**

```java
public static final int PREFETCHED
```

The state of the Player indicating that it has acquired all the resources to begin playing.

Value 300 is assigned to `PREFETCHED`.
Constant value: 300

**REALIZED**

```java
public static final int REALIZED
```
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.
Constant value: 200

**STARTED**

```java
public static final int STARTED
```

The state of the Player indicating that the Player has already started.

Value 400 is assigned to STARTED.
Constant value: 400

**TIME_UNKNOWN**

```java
public static final long TIME_UNKNOWN
```

The returned value indicating that the requested time is unknown.

Value -1 is assigned to TIME_UNKNOWN.
Constant value: -1

**UNREALIZED**

```java
public static final int UNREALIZED
```

The state of the Player indicating that it has not acquired the required information and resources to function.

Value 100 is assigned to UNREALIZED.
Constant value: 100

### Methods

**addPlayerListener**

```java
public void addPlayerListener(PlayerListener playerListener)
```

Add a player listener for this player.

**Parameters:**

- `playerListener` - the listener to add. If `null` is used, the request will be ignored.

**Throws:**

- `IllegalStateException` - Thrown if the Player is in the CLOSED state.

**See Also:** `Player.removePlayerListener(PlayerListener)`

**close**

```java
public void close()
```

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 PlayerListeners.

If close is called on a closed Player the request is ignored.

**deallocate**

```java
public void deallocate()
```
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` - Thrown if the Player is in the `CLOSED` state.

---

### `getContentType`

**public java.lang.String getContentType()**

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` - Thrown if the Player is in the `UNREALIZED` or `CLOSED` state.

---

### `getDuration`

**public long getDuration()**

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` - Thrown if the Player is in the `CLOSED` state.

---

### `getMediaTime`

**public long getMediaTime()**

Gets this Player's current media time.

`getMediaTime` may return `TIME_UNKNOWN` to indicate that the media time cannot be determined. However, once `getMediaTime` returns a known time (time not equals to `TIME_UNKNOWN`), subsequent calls to `getMediaTime` must not return `TIME_UNKNOWN`.

**Returns:**

- The current media time in microseconds or `TIME_UNKNOWN`.

**Throws:**

- `IllegalStateException` - Thrown if the Player is in the `CLOSED` state.

**See Also:** `Player.setMediaTime(long)`

---

### `getState`

**public int getState()**
See Also: Player.addPlayerListener(PlayerListener)

The current state of this Player. The possible states are: UNREALIZED, REALIZED, PREFETCHED, STARTED, CLOSED.

Returns:
The Player's current state.

prefetch

public void prefetch() throws javax.microedition.media.MediaException

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 - Thrown if the Player is in the CLOSED state.
- MediaException - Thrown if the Player cannot be prefetched.
- SecurityException - Thrown if the caller does not have security permission to prefetch the Player.

realize

public void realize() throws javax.microedition.media.MediaException

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, PREFETCHED or STARTED state, the request will be ignored.

Throws:
- IllegalStateException - Thrown if the Player is in the CLOSED state.
- MediaException - Thrown if the Player cannot be realized.
- SecurityException - Thrown if the caller does not have security permission to realize the Player.

removePlayerListener

public void removePlayerListener(PlayerListener playerListener)

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 - Thrown if the Player is in the CLOSED state.

See Also: Player.addPlayerListener(PlayerListener)
setLoopCount

```java
public void setLoopCount(int count)
```

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 indefinitely.

Throws:
- IllegalArgumentException - Thrown if the given count is invalid.
- IllegalStateException - Thrown if the Player is in the STARTED or CLOSED state.

setMediaTime

```java
public long setMediaTime(long now)
```

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 - Thrown if the Player is in the UNREALIZED or CLOSED state.
- MediaException - Thrown if the media time cannot be set.

See Also: Player.getMediaTime()

start

```java
public void start()
```

Throws javax.microedition.media.MediaException
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 PlayerListeners. 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 - Thrown if the Player is in the CLOSED state.
- MediaException - Thrown if the Player cannot be started.
- SecurityException - Thrown if the caller does not have security permission to start the Player.

---

public void stop() throws javax.microedition.media.MediaException

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 PlayerListeners.

If stop is called on a stopped Player, the request is ignored.

Throws:
- IllegalStateException - Thrown if the Player is in the CLOSED state.
- MediaException - Thrown if the Player cannot be stopped.
**javax.microedition.media**

**PlayerListener**

### Declaration

```java
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()**

```java
if (eventType == PlayerListener.STARTED) {...}
```

**Code fragment for catching proprietary events in playerUpdate()**

```java
if (eventType.equals("com.company.myEvent")) {...}
```

**See Also:** Player

---

### Field Summary

<table>
<thead>
<tr>
<th>Visibility</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>CLOSED</td>
<td>Posted when a Player is closed.</td>
</tr>
<tr>
<td></td>
<td>Value: closed</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>DEVICE_AVAILABLE</td>
<td>Posted when the system or another higher priority application has released an exclusive device which is now available to the Player.</td>
</tr>
<tr>
<td></td>
<td>Value: deviceAvailable</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>DEVICE_UNAVAILABLE</td>
<td>Posted when the system or another higher priority application has temporarily taken control of an exclusive device which was previously available to the Player.</td>
</tr>
<tr>
<td></td>
<td>Value: deviceUnavailable</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>DURATION_UPDATED</td>
<td>Posted when the duration of a Player is updated.</td>
</tr>
<tr>
<td></td>
<td>Value: durationUpdated</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>END_OF_MEDIA</td>
<td>Posted when a Player has reached the end of the media.</td>
</tr>
<tr>
<td></td>
<td>Value: endOfMedia</td>
<td></td>
</tr>
<tr>
<td>public static final</td>
<td>ERROR</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------</td>
<td></td>
</tr>
<tr>
<td>Posted when an error had occurred.</td>
<td>Value: error</td>
<td></td>
</tr>
<tr>
<td>public static final</td>
<td>STARTED</td>
<td></td>
</tr>
<tr>
<td>Posted when a Player is started.</td>
<td>Value: started</td>
<td></td>
</tr>
<tr>
<td>public static final</td>
<td>STOPPED</td>
<td></td>
</tr>
<tr>
<td>Posted when a Player stops in response to the stop method call.</td>
<td>Value: stopped</td>
<td></td>
</tr>
<tr>
<td>public static final</td>
<td>VOLUME_CHANGED</td>
<td></td>
</tr>
<tr>
<td>Posted when the volume of an audio device is changed.</td>
<td>Value: volumeChanged</td>
<td></td>
</tr>
</tbody>
</table>

**Method Summary**

```java
void playerUpdate(Player player, String event, Object eventData)
This method is called to deliver an event to a registered listener when a Player event is observed.
```

**Fields**

**CLOSED**

```
public static final java.lang.String CLOSED
```

Posted when a Player is closed. When this event is received, the eventData parameter is null.

Value closed is assigned to CLOSED.
Constant value: closed

**DEVICE_AVAILABLE**

```
public static final java.lang.String DEVICE_AVAILABLE
```

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.
Constant value: deviceAvailable

**DEVICE_UNAVAILABLE**

```
public static final java.lang.String DEVICE_UNAVAILABLE
```
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.
Constant value: deviceUnavailable

DURATION_UPDATED

public static final java.lang.String DURATION_UPDATED

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.
Constant value: durationUpdated

END_OF_MEDIA

public static final java.lang.String END_OF_MEDIA

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.
Constant value: endOfMedia

ERROR

public static final java.lang.String ERROR

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.
Constant value: error

STARTED

public static final java.lang.String STARTED

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.
Constant value: started

STOPPED

public static final java.lang.String STOPPED
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.
Constant value: stopped

---

**VOLUME_CHANGED**

```java
public static final java.lang.String VOLUME_CHANGED
```

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.
Constant value: volumeChanged

---

**Methods**

**playerUpdate**

```java
public void playerUpdate(Player player,
            String event,
            Object eventData)
```

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.
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

<table>
<thead>
<tr>
<th>Interfaces</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ToneControl</strong></td>
<td>ToneControl is the interface to enable playback of a user-defined monotonic tone sequence.</td>
</tr>
<tr>
<td><strong>VolumeControl</strong></td>
<td>VolumeControl is an interface for manipulating the audio volume of a Player.</td>
</tr>
</tbody>
</table>
javax.microedition.media.control
ToneControl

Declaration

public interface ToneControl extends javax.microedition.media.Control

All Superinterfaces:
javax.microedition.media.Control

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. In addition, the tone sequence format specified below can also be used as a file format to define tone sequences. A file containing a tone sequence as specified must use ".jts" as the file extension. "audio/x-tone-seq" designates the MIME type for this format.

The syntax of a tone sequence is described in Augmented BNF 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.
Following table shows the valid range of the parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Valid Range</th>
<th>Effective Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>tempo_modifier</td>
<td>5 &lt;= tempo_modifier &lt;= 127</td>
<td>20bpm to 508bpm</td>
<td>120bpm</td>
</tr>
<tr>
<td>resolution_unit</td>
<td>1 &lt;= resolution_unit &lt;= 127</td>
<td>1/1 note to 1/127 note</td>
<td>1/64 note</td>
</tr>
<tr>
<td>block_number</td>
<td>0 &lt;= block_number &lt;= 127</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>note</td>
<td>0 &lt;= note &lt;= 127 or SILENCE</td>
<td>C-1 to G9 or rest</td>
<td>-</td>
</tr>
<tr>
<td>duration</td>
<td>1 &lt;= duration &lt;= 127</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>volume</td>
<td>0 &lt;= volume &lt;= 100</td>
<td>0% to 100% volume</td>
<td>100%</td>
</tr>
<tr>
<td>multiplier</td>
<td>2 &lt;= multiplier &lt;= 127</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

The frequency of the note can be calculated from the following formula:

\[
\text{SEMITONE\_CONST} = 17.31234049066755 = \frac{1}{\ln(2^{0.5})}
\]

\[
\text{note} = \ln(\text{freq}/8.176) \times \text{SEMITONE\_CONST}
\]

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:

\[
\text{duration} \times 60 \times 1000 \times 4 / (\text{resolution} \times \text{tempo})
\]

The following table lists some common durations in musical notes:

<table>
<thead>
<tr>
<th>Note Length</th>
<th>Duration, Resolution=64</th>
<th>Duration, Resolution=96</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/1</td>
<td>64</td>
<td>96</td>
</tr>
<tr>
<td>1/4</td>
<td>16</td>
<td>24</td>
</tr>
<tr>
<td>1/4 dotted</td>
<td>24</td>
<td>36</td>
</tr>
<tr>
<td>1/8</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>1/8 triplets</td>
<td>-</td>
<td>8</td>
</tr>
<tr>
<td>4/1</td>
<td>REPEAT 4 &lt;note&gt; 64</td>
<td>REPEAT 4 &lt;note&gt; 96</td>
</tr>
</tbody>
</table>

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) { }

---

**Field Summary**

<table>
<thead>
<tr>
<th>Field Summary</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static</td>
<td>BLOCK_END</td>
</tr>
<tr>
<td>final</td>
<td>Defines an ending point for a block.</td>
</tr>
<tr>
<td></td>
<td>Value: -6</td>
</tr>
<tr>
<td>public static</td>
<td>BLOCK_START</td>
</tr>
<tr>
<td>final</td>
<td>Defines a starting point for a block.</td>
</tr>
<tr>
<td></td>
<td>Value: -5</td>
</tr>
<tr>
<td>public static</td>
<td>C4</td>
</tr>
<tr>
<td>final</td>
<td>Middle C.</td>
</tr>
<tr>
<td></td>
<td>Value: 60</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>setSequence(byte[] sequence)</td>
<td>Sets the tone sequence.</td>
</tr>
</tbody>
</table>

## Fields

### BLOCK_END

```java
public static final byte BLOCK_END
```

Defines an ending point for a block.

Value: -6 is assigned to BLOCK_END.
Constant value: -6

### BLOCK_START

```java
public static final byte BLOCK_START
```

Defines a starting point for a block.

Value: -5 is assigned to BLOCK_START.
Constant value: -5

### C4

```java
public static final byte C4
```
Middle C.
Value 60 is assigned to C₄.
Constant value: 60

---

**PLAY_BLOCK**

```java
public static final byte PLAY_BLOCK
```

Play a defined block.
Value -7 is assigned to PLAY_BLOCK.
Constant value: -7

---

**REPEAT**

```java
public static final byte REPEAT
```

The REPEAT event tag.
Value -9 is assigned to REPEAT.
Constant value: -9

---

**RESOLUTION**

```java
public static final byte RESOLUTION
```

The RESOLUTION event tag.
Value -4 is assigned to RESOLUTION.
Constant value: -4

---

**SET_VOLUME**

```java
public static final byte SET_VOLUME
```

The SET_VOLUME event tag.
Value -8 is assigned to SET_VOLUME.
Constant value: -8

---

**SILENCE**

```java
public static final byte SILENCE
```

Silence.
Value -1 is assigned to SILENCE.
Constant value: -1

---

**TEMPO**

```java
public static final byte TEMPO
```

The TEMPO event tag.
Value -3 is assigned to TEMPO.
Constant value: -3
The VERSION attribute tag.

Value -2 is assigned to VERSION.
Constant value: -2

Methods

setSequence

public void setSequence(byte[] sequence)

Sets the tone sequence.

Parameters:

sequence - The sequence to set.

Throws:

IllegalArgumentException - Thrown if the sequence is null or invalid.
IllegalStateException - 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

All Superinterfaces:

javax.microedition.media.Control

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: Control, Player, PlayerListener

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int getLevel()</td>
<td>Get the current volume level set.</td>
</tr>
<tr>
<td>boolean isMuted()</td>
<td>Get the mute state of the signal associated with this VolumeControl.</td>
</tr>
<tr>
<td>int setLevel(int level)</td>
<td>Set the volume using a linear point scale with values between 0 and 100.</td>
</tr>
<tr>
<td>void setMute(boolean mute)</td>
<td>Mute or unmute the Player associated with this VolumeControl.</td>
</tr>
</tbody>
</table>

Methods
getLevel

public int getLevel()

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: VolumeControl.setLevel(int)

isMuted

public boolean isMuted()

Get the mute state of the signal associated with this VolumeControl.

Returns:
The mute state.

See Also: VolumeControl.setMute(boolean)

setLevel

public int setLevel(int level)

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: VolumeControl.getLevel()

setMute

public void setMute(boolean mute)

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.

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: VolumeControl.isMuted()
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.

12.1 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.

12.1.1 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.

12.2 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.

12.2.1 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.

12.2.2 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.

### 12.2.3 MIDlet Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDlet-Name</td>
<td>The name of the MIDlet suite that identifies the MIDlets to the user.</td>
</tr>
<tr>
<td>MIDlet-Version</td>
<td>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.</td>
</tr>
<tr>
<td>MIDlet-Vendor</td>
<td>The organization that provides the MIDlet suite.</td>
</tr>
<tr>
<td>MIDlet-Icon</td>
<td>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.</td>
</tr>
<tr>
<td>MIDlet-Description</td>
<td>The description of the MIDlet suite.</td>
</tr>
<tr>
<td>MIDlet-Info-URL</td>
<td>A URL for information further describing the MIDlet suite. The syntax and meaning MUST conform to RFC2396 and RFCs that define each scheme.</td>
</tr>
<tr>
<td>MIDlet-&lt;n&gt;</td>
<td>The name, icon, and class of the nth MIDlet in the JAR file separated by a comma. The lowest value of &lt;n&gt; 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.</td>
</tr>
</tbody>
</table>

1. Name is used to identify this MIDlet to the user. The name must be present and be non-null.
2. 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.

If an implementation displays MIDlet icons (using associated MIDlet-Icon attribute), the following requirements apply:

- The application manager MUST always be able to display a MIDlet-Icon of 16 by 16 pixels in the application list.
• The application manager SHOULD always be able to display a MIDlet-Icon of 24 by 24 pixels in the application list.
• A MIDlet-Icon larger than 24 by 24 MAY be displayed properly in full size in the application list.
• A MIDlet-Icon MUST NOT be scaled down in the application list, but the application manager MAY use centered cropping to fit the icon into the list.

3. Class is the name of the class extending the `javax.microedition.midlet.MIDlet` class for the nth MIDlet. The class name 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.

<table>
<thead>
<tr>
<th>MIDlet-Jar-URL</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDlet-Jar-Size</td>
<td>The number of bytes in the JAR file.</td>
</tr>
<tr>
<td>MIDlet-Data-Size</td>
<td>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.</td>
</tr>
<tr>
<td>MicroEdition-Profile</td>
<td>The J2ME profiles required, using the same format and value as the System property <code>microedition.profiles</code> (for example &quot;MIDP-2.0&quot;). 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).</td>
</tr>
<tr>
<td>MicroEdition-Configuration</td>
<td>The J2ME Configuration required using the same format and value as the System property <code>microedition.configuration</code> (for example &quot;CLDC-1.0&quot;).</td>
</tr>
<tr>
<td>MIDlet-Permissions</td>
<td>Zero or more permissions that are critical to the function of the MIDlet suite. See the MIDlet Suite Security section for details of usage.</td>
</tr>
<tr>
<td>MIDlet-Permissions-Opt</td>
<td>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.</td>
</tr>
<tr>
<td>MIDlet-Push-&lt;n&gt;</td>
<td>Register a MIDlet to handle inbound connections. Refer to <code>javax.microedition.io.PushRegistry</code> for details.</td>
</tr>
<tr>
<td>MIDlet-Install-Notify</td>
<td>Refer to the OTA Specification for details.</td>
</tr>
<tr>
<td>MIDlet-Delete-Notify</td>
<td>Refer to the OTA Specification for details.</td>
</tr>
<tr>
<td>MIDlet-Delete-Confirm</td>
<td>Refer to the OTA Specification for details.</td>
</tr>
</tbody>
</table>
Runtime-Execution-Environment: This is an optional attribute that indicates the runtime execution environment REQUIRED by the application. This attribute MAY have the value MIDP.CLDC. If the MIDlet suite does not specify the attribute, the implicit value is MIDP.CLDC. When the value of the Runtime-Execution-Environment attribute is MIDP.CLDC, the behavior of the implementation MUST adhere to the requirements presented in the following more detailed description. Additional values for this attribute MAY be defined in future.

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.

12.2.3.1 Runtime-Execution-Environment

When an application defines the Runtime-Execution-Environment attribute value MIDP.CLDC or does not define this attribute, the behavior of the mobile handset implementation during execution of this application MUST adhere to the following requirements:

- The set of supported APIs and the API behavior, as well as the behavior of the underlying virtual machine, MUST comply with the CLDC 1.1 specification.
- A mobile handset implementation MAY alternatively use the CDC configuration of Java ME. However, if CDC is used as the underlying configuration, applications running on top of this platform MUST see an environment that is semantically and functionally equivalent to a CLDC 1.1 platform. CDC-specific APIs or CDC-specific behavior MUST NOT be exposed to applications or application developers.
- When an application defines the Runtime-Execution-Environment attribute value MIDP.CLDC the application MUST also define a CLDC platform in the MicroEdition-Configuration attribute. If this is not the case, the application MUST NOT be installed and Status Code 905 "Attribute Mismatch" MUST be used when reporting the installation suite.
- If an application defines a value for the Runtime-Execution-Environment or MicroEdition-Configuration attribute that is not supported by the implementation, the application MUST NOT be installed. All mobile handset implementations MUST support the "MIDP.CLDC" value for the Runtime-Execution-Environment attribute.
- A mobile handset implementation MUST support the Runtime-Execution-Environment attribute.

12.2.3.2 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 [Versioning Specification]. 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.

12.2.3.3 JAR Manifest

The manifest provides information about the contents of the JAR file. JAR file formats and specifications are available at [JAR Manifest Specification]. 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
MicroEdition-Profile: MIDP-2.0
MicroEdition-Configuration: CLDC-1.0
Solitaire-Author: John Q. Public
```

### 12.2.3.4 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`.

### 12.2.4 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 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`

In addition, to improve the user experience (by rejecting incompatible suites without needing to download the JAR file), the application descriptor SHOULD contain the following attributes:

- `MicroEdition-Profile`
- `MicroEdition-Configuration`
- `MIDlet-Permissions`

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 like the
following example:

```
MIDlet-Name: CardGames
MIDlet-Version: 1.1.9
MIDlet-Vendor: CardsRUS
MIDlet-1: Solitaire, /Solitare.png, com.cardsrus.org.Solitare
MicroEdition-Profile: MIDP-2.0
MicroEdition-Configuration: CLDC-1.0
```
12.2.5 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.

<table>
<thead>
<tr>
<th>Class in javax.microedition.midlet</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDlet</td>
<td>Extended by a MIDlet to allow the application management software to start, stop, and destroy it.</td>
</tr>
<tr>
<td>MIDletStateException</td>
<td>Thrown when the application cannot make the change requested.</td>
</tr>
</tbody>
</table>

12.2.6 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

12.3 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 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.
12.4 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:

<table>
<thead>
<tr>
<th>State Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 following diagram illustrates the state transitions and requirements related to the `MIDlet` lifecycle.

### 12.5 MIDlet Lifecycle Model

A typical sequence of `MIDlet` execution is:

<table>
<thead>
<tr>
<th>Application Management Software</th>
<th>MIDlet</th>
</tr>
</thead>
<tbody>
<tr>
<td>The application management software creates a new instance of a <code>MIDlet</code>.</td>
<td>The default (no argument) constructor for the <code>MIDlet</code> is called; it is in the Paused state.</td>
</tr>
</tbody>
</table>
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 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 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.

### 12.6 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

### 12.7 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.

### 12.8 Example MIDlet Application

The example uses the MIDlet lifecycle to do a simple measurement of the speed of the Java Virtual Machine.

```java
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
```
public class MethodTimes extends MIDlet implements Runnable {

    private static class State {
        // The state for the timing thread.
        Thread thread;
        State() {
            // The state for the timing thread.
        }
    }

    public static State state;

    public static State startUp() {
        State state = new State();
        return state;
    }

    public static State pauseApp() {
        state = null;
        return state;
    }

    public static State destroyApp() {
        state = null;
        return state;
    }

    public void startApp() {
        state = new State();
        state.thread = new Thread(this);
        state.thread.start();
    }

    public void pauseApp() {
        state = null;
    }

    public void destroyApp() {
        state = null;
    }

    public void run() {
        Thread curr = Thread.currentThread(); // Remember which thread is
        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();
    }

    public void empty() {
        // An Empty method.
    }
}
void empty() {
}

Since: MIDP 1.0
# Class Summary

<table>
<thead>
<tr>
<th>Classes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDlet</td>
<td>A MIDlet is a MID Profile application.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exceptions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDletStateChangeException</td>
<td>Signals that a requested MIDlet state change failed.</td>
</tr>
</tbody>
</table>
javax.microedition.midlet
MIDlet

Declaration

public abstract class MIDlet

Object

private 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 MIDlets 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.

Constructor Summary

<table>
<thead>
<tr>
<th>protected MIDlet()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protected constructor for subclasses.</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>int checkPermission(String permission)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the status of the specified permission.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>abstract void destroyApp(boolean unconditional)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signals the MIDlet to terminate and enter the Destroyed state.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String getAppProperty(String key)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provides a MIDlet with a mechanism to retrieve named properties from the application management software.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void notifyDestroyed()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Used by an MIDlet to notify the application management software that it has entered into the Destroyed state.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void notifyPaused()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notifies the application management software that the MIDlet does not want to be active and has entered the Paused state.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>abstract void pauseApp()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signals the MIDlet to enter the Paused state.</td>
</tr>
</tbody>
</table>
boolean platformRequest(String URL)

Requests that the device handle (for example, display or install) the indicated URL.

void resumeRequest()

Provides a MIDlet with a mechanism to indicate that it is interested in entering the Active state.

abstract void startApp()

Signals the MIDlet that it has entered the Active state.

### Methods inherited from class Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### Constructors

**MIDlet**

protected MIDlet()

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:**

java.lang.SecurityException - unless the application management software is creating the MIDlet.

### Methods

**checkPermission**

public final int checkPermission(String permission)

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**

protected abstract void destroyApp(boolean unconditional)

throws javax.microedition.midlet.MIDletStateChangeException
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.

MIDlets 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 the exception is ignored and the MIDlet is put into the Destroyed state.

When an uncaught exception occurs in a thread, the thread MUST be stopped in accordance with the Java Language Specification. In addition, further handling of the uncaught exception MAY occur as follows:

- If an implementation can identify the MIDlet that created the thread (the MIDlet to which the thread belongs), the MIDlet.destroyApp(false) method of this MIDlet MAY be called.
- If an implementation can only identify the MIDlet suite that created the thread (the MIDlet suite to which the thread belongs), the MIDlet.destroyApp(false) methods of all MIDlets from this suite MAY be called.
- If an implementation cannot identify either a MIDlet or a MIDlet suite that owns the offending thread, MIDlet.destroyApp(false) methods of all MIDlets executing in this VM instance MAY be called.

In all of these cases, the behavior after a call to MIDlet.destroyApp(false) is as follows:

- If a MIDlet attempts to recover by throwing javax.microedition.midlet.MIDletStateChangeException, the implementation MAY allow the MIDlet to continue running.
- If a call to MIDlet.destroyApp(false) results in a different exception of any kind, the implementation MUST terminate the MIDlet.

**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** - 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**

```java
public final java.lang.String getAppProperty(String key)
```

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**

```java
public final void notifyDestroyed()
```

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**

```java
public final void notifyPaused()
```

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**

```java
protected abstract void pauseApp()
```

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 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**

```java
public final boolean platformRequest(String URL)
    throws javax.microedition.io.ConnectionNotFoundException
```

- **notifyDestroyed**
- **notifyPaused**
- **pauseApp**
- **platformRequest**

-443-
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, 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 acknowledge 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

public final void resumeRequest ()

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

protected abstract void startApp()
  throws javax.microedition.midlet.MIDletStateChangeException

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 - is thrown if the MIDlet cannot start now but might be able to start at a later time.
javax.microedition.midlet
MIDletStateException

Declaration

public class MIDletStateException extends Exception

Object
   |-- Throwable
   |   |-- Exception
   |   |   |-- javax.microedition.midlet.MIDletStateException

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

Constructor Summary

| public MIDletStateException() |
| Constructs an exception with no specified detail message. |
| public MIDletStateException(String s) |
| Constructs an exception with the specified detail message. |

Methods inherited from class Throwable

getMessage, printStackTrace, toString

Methods inherited from class Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

MIDletStateException

public MIDletStateException()

   Constructs an exception with no specified detail message.

MIDletStateException

public MIDletStateException(String s)

   Constructs an exception with the specified detail message.

Parameters:
   s - the detail message
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
- [RFC2459] - Internet X.509 Public Key Infrastructure

13.1 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.

13.1.1 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.

### 13.1.2 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.

### 13.1.3 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.

### 13.1.4 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 | Interface common to certificates. |

### Exceptions

| CertificateException | 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 version of the distinguished 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 "=" (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

<table>
<thead>
<tr>
<th>Object ID</th>
<th>Binary</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>id-at-commonName</td>
<td>55:04:03</td>
<td>CN</td>
</tr>
<tr>
<td>id-at-surname</td>
<td>55:04:04</td>
<td>SN</td>
</tr>
<tr>
<td>id-at-countryName</td>
<td>55:04:06</td>
<td>C</td>
</tr>
<tr>
<td>id-at-localityName</td>
<td>55:04:07</td>
<td>L</td>
</tr>
<tr>
<td>id-at-stateOrProvinceName</td>
<td>55:04:08</td>
<td>ST</td>
</tr>
<tr>
<td>id-at-streetAddress</td>
<td>55:04:09</td>
<td>STREET</td>
</tr>
<tr>
<td>id-at-organizationName</td>
<td>55:04:0A</td>
<td>O</td>
</tr>
<tr>
<td>id-at-organizationUnitName</td>
<td>55:04:0B</td>
<td>OU</td>
</tr>
<tr>
<td>emailAddress</td>
<td>2A:86:48:86:F7:0D:01:09:01</td>
<td>EmailAddress</td>
</tr>
</tbody>
</table>

Example of a printable distinguished name:C=US;O=Any Company, Inc.;CN=www.anycompany.com

Since: MIDP 2.0

Method Summary
java.lang.String  getIssuer()
    Gets the name of this certificate's issuer.

long  getNotAfter()
    Gets the time after which this Certificate may not be used from the validity period.

long  getNotBefore()
    Gets the time before which this Certificate may not be used from the validity period.

java.lang.String  getSerialNumber()
    Gets the printable form of the serial number of this Certificate.

java.lang.String  getSigAlgName()
    Gets the name of the algorithm used to sign the Certificate.

java.lang.String  getSubject()
    Gets the name of this certificate's subject.

java.lang.String  getType()
    Get the type of the Certificate.

java.lang.String  getVersion()
    Gets the version number of this Certificate.

---

Methods

getIssuer

public java.lang.String  getIssuer()

    Gets the name of this certificate's issuer.

    Returns:
    The issuer of the Certificate; the value MUST NOT be null.

getNotAfter

public long  getNotAfter()

    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

public long  getNotBefore()

    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
public java.lang.String getSerialNumber()

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
public java.lang.String getSigAlgName()

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
public java.lang.String getSubject()

Gets the name of this certificate's subject.

Returns:
The subject of this Certificate; the value MUST NOT be null.

getType
public java.lang.String getType()

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
public java.lang.String getVersion()

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 IOException

Object
+- Throwable
  +- Exception
  |  +- 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

Field Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final</td>
<td>BAD_EXTENSIONS</td>
<td>Indicates a certificate has unrecognized critical extensions. Value: 1</td>
</tr>
<tr>
<td>public static final</td>
<td>BROKEN_CHAIN</td>
<td>Indicates a certificate in a chain was not issued by the next authority in the chain. Value: 11</td>
</tr>
<tr>
<td>public static final</td>
<td>CERTIFICATE_CHAIN_TOO_LONG</td>
<td>Indicates the server certificate chain exceeds the length allowed by an issuer's policy. Value: 2</td>
</tr>
<tr>
<td>public static final</td>
<td>EXPIRED</td>
<td>Indicates a certificate is expired. Value: 3</td>
</tr>
<tr>
<td>public static final</td>
<td>INAPPROPRIATE_KEY_USAGE</td>
<td>Indicates a certificate public key has been used in way deemed inappropriate by the issuer. Value: 10</td>
</tr>
<tr>
<td>public static final</td>
<td>MISSING_SIGNATURE</td>
<td>Indicates a certificate object does not contain a signature. Value: 5</td>
</tr>
<tr>
<td>public static final</td>
<td>NOT_YET_VALID</td>
<td>Indicates a certificate is not yet valid. Value: 6</td>
</tr>
<tr>
<td>public static final</td>
<td>ROOT_CA_EXPIRED</td>
<td>Indicates the root CA’s public key is expired.</td>
</tr>
<tr>
<td>public static final</td>
<td>SITENAME_MISMATCH</td>
<td>Indicates a certificate does not contain the correct site name.</td>
</tr>
<tr>
<td>public static final</td>
<td>UNAUTHORIZED_INTERMEDIATE_CA</td>
<td>Indicates an intermediate certificate in the chain does not have the authority to be an intermediate CA.</td>
</tr>
<tr>
<td>public static final</td>
<td>UNRECOGNIZED_ISSUER</td>
<td>Indicates a certificate was issued by an unrecognized entity.</td>
</tr>
<tr>
<td>public static final</td>
<td>UNSUPPORTED_PUBLIC_KEY_TYPE</td>
<td>Indicates that type of the public key in a certificate is not supported by the device.</td>
</tr>
<tr>
<td>public static final</td>
<td>UNSUPPORTED_SIGALG</td>
<td>Indicates a certificate was signed using an unsupported algorithm.</td>
</tr>
<tr>
<td>public static final</td>
<td>VERIFICATION_FAILED</td>
<td>Indicates a certificate failed verification.</td>
</tr>
</tbody>
</table>

**Constructor Summary**

public `CertificateException(Certificate certificate, byte status)`

Create a new exception with a `Certificate` and specific error reason.

public `CertificateException(String message, Certificate certificate, byte status)`

Create a new exception with a message, `Certificate`, and specific error reason.

**Method Summary**

`javax.microedition.pki.Certificate` getCertificate()

Get the `Certificate` that caused the exception.

`byte` getReason()

Get the reason code.

**Methods inherited from class `Throwable`**

getMessage, printStackTrace, toString

**Methods inherited from class `Object`**
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
<table>
<thead>
<tr>
<th>Fields</th>
<th>Description</th>
<th>Constant value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BAD_EXTENSIONS</strong></td>
<td>Indicates a certificate has unrecognized critical extensions. The value is 1.</td>
<td>1</td>
</tr>
<tr>
<td><strong>BROKEN_CHAIN</strong></td>
<td>Indicates a certificate in a chain was not issued by the next authority in the chain. The value is 11.</td>
<td>11</td>
</tr>
<tr>
<td><strong>CERTIFICATE_CHAIN_TOO_LONG</strong></td>
<td>Indicates the server certificate chain exceeds the length allowed by an issuer's policy. The value is 2.</td>
<td>2</td>
</tr>
<tr>
<td><strong>EXPIRED</strong></td>
<td>Indicates a certificate is expired. The value is 3.</td>
<td>3</td>
</tr>
<tr>
<td><strong>INAPPROPRIATE_KEY_USAGE</strong></td>
<td>Indicates a certificate public key has been used in way deemed inappropriate by the issuer. The value is 10.</td>
<td>10</td>
</tr>
<tr>
<td><strong>MISSING_SIGNATURE</strong></td>
<td>Indicates a certificate object does not contain a signature. The value is 5.</td>
<td>5</td>
</tr>
<tr>
<td><strong>NOT_YET_VALID</strong></td>
<td>Indicates a certificate is not yet valid. The value is 6.</td>
<td>6</td>
</tr>
<tr>
<td><strong>ROOT_CA_EXPIRED</strong></td>
<td>Indicates the root CA's public key is expired. The value is 12.</td>
<td>12</td>
</tr>
</tbody>
</table>
SITENAME_MISMATCH

public static final byte SITENAME_MISMATCH

Indicates a certificate does not contain the correct site name. The value is 7.
Constant value: 7

UNAUTHORIZED_INTERMEDIATE_CA

public static final byte UNAUTHORIZED_INTERMEDIATE_CA

Indicates an intermediate certificate in the chain does not have the authority to be a intermediate CA. The value is 4.
Constant value: 4

UNRECOGNIZED_ISSUER

public static final byte UNRECOGNIZED_ISSUER

Indicates a certificate was issued by an unrecognized entity. The value is 8.
Constant value: 8

UNSUPPORTED_PUBLIC_KEY_TYPE

public static final byte UNSUPPORTED_PUBLIC_KEY_TYPE

Indicates that type of the public key in a certificate is not supported by the device. The value is 13.
Constant value: 13

UNSUPPORTED_SIGALG

public static final byte UNSUPPORTED_SIGALG

Indicates a certificate was signed using an unsupported algorithm. The value is 9.
Constant value: 9

VERIFICATION_FAILED

public static final byte VERIFICATION_FAILED

Indicates a certificate failed verification. The value is 14.
Constant value: 14

Constructors

CertificateException

public CertificateException (Certificate certificate, byte status)

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

public CertificateException(String message,
    Certificate certificate,
    byte status)

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

public javax.microedition.pki.Certificate getCertificate()

Get the Certificate that caused the exception.

Returns:
    the Certificate that included the failure.

getReason

public byte getReason()

Get the reason code.

Returns:
    the reason code
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.

14.1 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.

14.1.1 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 accessibility 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.

14.1.2 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.

14.1.3 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.

```java
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;

/**<n
* 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();
            return true;
        } catch (IOException e) {
            throw new IllegalArgumentException(e);
        }
    }
}
```
catch (EOFException eofe) {
    System.out.println(eofe);
    eofe.printStackTrace();
}

try {
    score1 = inputStream1.readInt();
    score2 = inputStream2.readInt();
} catch (EOFException eofe) {
    System.out.println(eofe);
    eofe.printStackTrace();
} catch (IOException eofe) {
    System.out.println(eofe);
    eofe.printStackTrace();
}

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() {
    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) {
    }
}

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

<table>
<thead>
<tr>
<th>Interfaces</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RecordComparator</td>
<td>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.</td>
</tr>
<tr>
<td>RecordEnumeration</td>
<td>An interface representing a bidirectional record store Record enumerator.</td>
</tr>
<tr>
<td>RecordFilter</td>
<td>An interface defining a filter which examines a record to see if it matches (based on an application-defined criteria).</td>
</tr>
<tr>
<td>RecordListener</td>
<td>A listener interface for receiving Record Changed/Added/Deleted events from a record store.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Classes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RecordStore</td>
<td>A class representing a record store.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exceptions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvalidRecordIDException</td>
<td>Thrown to indicate an operation could not be completed because the record ID was invalid.</td>
</tr>
<tr>
<td>RecordStoreException</td>
<td>Thrown to indicate a general exception occurred in a record store operation.</td>
</tr>
<tr>
<td>RecordStoreFullException</td>
<td>Thrown to indicate an operation could not be completed because the record store system storage is full.</td>
</tr>
<tr>
<td>RecordStoreNotFoundException</td>
<td>Thrown to indicate an operation could not be completed because the record store could not be found.</td>
</tr>
<tr>
<td>RecordStoreNotOpenException</td>
<td>Thrown to indicate that an operation was attempted on a closed record store.</td>
</tr>
</tbody>
</table>
javax.microedition.rms
InvalidRecordIDException

Declaration

public class InvalidRecordIDException extends RecordStoreException

Object
    +--Throwable
    +--Exception
    +--javax.microedition.rms.RecordStoreException
        +--javax.microedition.rms.InvalidRecordIDException

Description
Thrown to indicate an operation could not be completed because the record ID was invalid.
Since: MIDP 1.0

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>InvalidRecordIDException(String message)</td>
</tr>
<tr>
<td></td>
<td>Constructs a new InvalidRecordIDException</td>
</tr>
<tr>
<td></td>
<td>with the specified detail message.</td>
</tr>
<tr>
<td>public</td>
<td>InvalidRecordIDException()</td>
</tr>
<tr>
<td></td>
<td>Constructs a new InvalidRecordIDException</td>
</tr>
<tr>
<td></td>
<td>with no detail message.</td>
</tr>
</tbody>
</table>

Methods inherited from class Throwable
getMessage, printStackTrace, toString

Methods inherited from class Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

InvalidRecordIDException

public InvalidRecordIDException()

Constructs a new InvalidRecordIDException with no detail message.

InvalidRecordIDException

public InvalidRecordIDException(String message)

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:

```java
RecordComparator c = new AddressRecordComparator();
if (c.compare(recordStore.getRecord(rec1), recordStore.getRecord(rec2))
    == RecordComparator.PRECEDES)
    return rec1;
```

Since: MIDP 1.0

Field Summary

<table>
<thead>
<tr>
<th>public static final</th>
<th>EQUIVALENT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EQUIVALENT means that in terms of search or sort order, the two records are the same.</td>
</tr>
<tr>
<td></td>
<td>Value: 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>FOLLOWS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FOLLOWS means that the left (first parameter) record follows the right (second parameter) record in terms of search or sort order.</td>
</tr>
<tr>
<td></td>
<td>Value: 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>PRECEDES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PRECEDES means that the left (first parameter) record precedes the right (second parameter) record in terms of search or sort order.</td>
</tr>
<tr>
<td></td>
<td>Value: -1</td>
</tr>
</tbody>
</table>

Method Summary

```java
int compare(byte[] rec1, byte[] rec2)
```

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.

Fields

**EQUIVALENT**

public static final int EQUIVALENT
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.
Constant value: 0

**FOLLOWS**

`public static final int FOLLOWS`

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.
Constant value: 1

**PRECEDES**

`public static final int PRECEDES`

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.
Constant value: -1

**Methods**

**compare**

`public int compare(byte[] rec1, byte[] rec2)`

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

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void destroy()</td>
<td>Frees internal resources used by this RecordEnumeration.</td>
</tr>
<tr>
<td>boolean hasNextElement()</td>
<td>Returns true if more elements exist in the next direction.</td>
</tr>
<tr>
<td>boolean hasPreviousElement()</td>
<td>Returns true if more elements exist in the previous direction.</td>
</tr>
<tr>
<td>boolean isKeptUpdated()</td>
<td>Returns true if the enumeration keeps its enumeration current with any changes in the records.</td>
</tr>
</tbody>
</table>
void keepUpdated(boolean keepUpdated)
    Used to set whether the enumeration will be keep its internal index up to date with the record store record additions/deletions/changes.

byte[] nextRecord()
    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.

int nextRecordId()
    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.

int numRecords()
    Returns the number of records available in this enumeration's set.

byte[] previousRecord()
    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.

int previousRecordId()
    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.

void rebuild()
    Request that the enumeration be updated to reflect the current record set.

void reset()
    Returns the enumeration index to the same state as right after the enumeration was created.

**Methods**

**destroy**

public void destroy()

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**

public boolean hasNextElement()

Returns true if more elements exist in the next direction.

**Returns:**
true if more elements exist in the next direction

**hasPreviousElement**

public boolean hasPreviousElement()

Returns true if more elements exist in the previous direction.

**Returns:**
true if more elements exist in the previous direction
isKeptUpdated

public boolean isKeptUpdated()

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

public void keepUpdated(boolean keepUpdated)

Used to set whether the enumeration will 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: RecordEnumeration.rebuild()

nextRecord

public byte[] nextRecord()
throws javax.microedition.rms.InvalidRecordIDException,
javax.microedition.rms.RecordStoreNotOpenException,
javax.microedition.rms.RecordStoreException

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 - 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 - if the record store is not open
RecordStoreException - if a general record store exception occurs

nextRecordId

public int nextRecordId()
throws javax.microedition.rms.InvalidRecordIDException

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 - 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

public int numRecords()

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

public byte[] previousRecord() throws javax.microedition.rms.InvalidRecordIDException, javax.microedition.rms.RecordStoreNotOpenException, javax.microedition.rms.RecordStoreException

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 - 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 - if the record store is not open
RecordStoreException - if a general record store exception occurs.

previousRecordId

public int previousRecordId() throws javax.microedition.rms.InvalidRecordIDException

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 - 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

public void rebuild()

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: RecordEnumeration.keepUpdated(boolean)

reset

public void reset()

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:

```java
RecordFilter f = new DateRecordFilter(); // class implements RecordFilter
if (f.matches(recordStore.getRecord(theRecordID)) == true)
    DoSomethingUseful(theRecordID);
```

Since: MIDP 1.0

Method Summary

<table>
<thead>
<tr>
<th>boolean matches(byte[] candidate)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns true if the candidate matches the implemented criterion.</td>
</tr>
</tbody>
</table>

Methods

matches

```java
public boolean matches(byte[] candidate)
```

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)

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>recordAdded(RecordStore recordStore, int recordId)</td>
<td>Called when a record has been added to a record store.</td>
</tr>
<tr>
<td>recordChanged(RecordStore recordStore, int recordId)</td>
<td>Called after a record in a record store has been changed.</td>
</tr>
<tr>
<td>recordDeleted(RecordStore recordStore, int recordId)</td>
<td>Called after a record has been deleted from a record store.</td>
</tr>
</tbody>
</table>

Methods

recordAdded

public void recordAdded(RecordStore recordStore, int recordId)

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

public void recordChanged(RecordStore recordStore, int recordId)

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

public void recordDeleted(RecordStore recordStore, int recordId)
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
Field Summary

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final</td>
<td>AUTHMODE_ANY</td>
<td>Authorization to allow access to any MIDlet suites. Value: 1</td>
</tr>
<tr>
<td>public static final</td>
<td>AUTHMODE_PRIVATE</td>
<td>Authorization to allow access only to the current MIDlet suite. Value: 0</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int addRecord(byte[] data, int offset, int numBytes)</td>
<td>Adds a new record to the record store.</td>
</tr>
<tr>
<td>void addRecordListener(RecordListener listener)</td>
<td>Adds the specified RecordListener.</td>
</tr>
<tr>
<td>void closeRecordStore()</td>
<td>This method is called when the MIDlet requests to have the record store closed.</td>
</tr>
<tr>
<td>void deleteRecord(int recordId)</td>
<td>The record is deleted from the record store.</td>
</tr>
<tr>
<td>static void deleteRecordStore(String recordStoreName)</td>
<td>Deletes the named record store.</td>
</tr>
<tr>
<td>javax.microedition.rms.RecordEnumeration enumerateRecords(RecordFilter filter, RecordComparator comparator, boolean keepUpdated)</td>
<td>Returns an enumeration for traversing a set of records in the record store in an optionally specified order.</td>
</tr>
<tr>
<td>long getLastModified()</td>
<td>Returns the last time the record store was modified, in the format used by System.currentTimeMillis().</td>
</tr>
<tr>
<td>java.lang.String getName()</td>
<td>Returns the name of this RecordStore.</td>
</tr>
<tr>
<td>int getNextRecordID()</td>
<td>Returns the recordId of the next record to be added to the record store.</td>
</tr>
<tr>
<td>int getNumRecords()</td>
<td>Returns the number of records currently in the record store.</td>
</tr>
<tr>
<td>byte[] getRecord(int recordId)</td>
<td>Returns a copy of the data stored in the given record.</td>
</tr>
<tr>
<td>int getRecord(int recordId, byte[] buffer, int offset)</td>
<td>Returns the data stored in the given record.</td>
</tr>
<tr>
<td>int getRecordSize(int recordId)</td>
<td>Returns the size (in bytes) of the MIDlet data available in the given record.</td>
</tr>
<tr>
<td>int getSize()</td>
<td>Returns the amount of space, in bytes, that the record store occupies.</td>
</tr>
<tr>
<td>int getSizeAvailable()</td>
<td>Returns the amount of additional room (in bytes) available for this record store to grow.</td>
</tr>
</tbody>
</table>
int getVersion()

Each time a record store is modified (by addRecord, setRecord, or deleteRecord methods) its version is incremented.

static java.lang.String[] listRecordStores()

Returns an array of the names of record stores owned by the MIDlet suite.

static javax.microedition.rms.RecordStore openRecordStore(String recordStoreName, boolean createIfNecessary)

Open (and possibly create) a record store associated with the given MIDlet suite.

static javax.microedition.rms.RecordStore openRecordStore(String recordStoreName, boolean createIfNecessary, int authmode, boolean writable)

Open (and possibly create) a record store that can be shared with other MIDlet suites.

static javax.microedition.rms.RecordStore openRecordStore(String recordStoreName, String vendorName, String suiteName)

Open a record store associated with the named MIDlet suite.

void removeRecordListener(RecordListener listener)

Removes the specified RecordListener.

void setMode(int authmode, boolean writable)

Changes the access mode for this RecordStore.

void setRecord(int recordId, byte[] newData, int offset, int numBytes)

Sets the data in the given record to that passed in.

Methods inherited from class Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

---

Fields

**AUTHMODE_ANY**

public static final int AUTHMODE_ANY

Authorization to allow access to any MIDlet suites. AUTHMODE_ANY has a value of 1.
Constant value: 1

**AUTHMODE_PRIVATE**

public static final int AUTHMODE_PRIVATE

Authorization to allow access only to the current MIDlet suite. AUTHMODE_PRIVATE has a value of 0.
Constant value: 0

---

Methods
**addRecord**

```java
public int addRecord(byte[] data,
        int offset,
        int numBytes)
throws javax.microedition.rms.RecordStoreNotOpenException,
        javax.microedition.rms.RecordStoreException,
        javax.microedition.rms.RecordStoreFullException
```

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` - if the record store is not open
- `RecordStoreException` - if a different record store-related exception occurred
- `RecordStoreFullException` - 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**

```java
public void addRecordListener(RecordListener listener)
```

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:** `RecordStore.removeRecordListener(RecordListener)`

---

**closeRecordStore**

```java
public void closeRecordStore()
throws javax.microedition.rms.RecordStoreNotOpenException,
        javax.microedition.rms.RecordStoreException
```

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` - if the record store is not open
- `RecordStoreException` - if a different record store-related exception occurred

---

**deleteRecord**

```java
public void deleteRecord(int recordId)
throws javax.microedition.rms.RecordStoreNotOpenException,
        javax.microedition.rms.InvalidRecordIDException,
        javax.microedition.rms.RecordStoreException
```

-478-
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 - if the record store is not open
InvalidRecordIDException - if the recordId is invalid
RecordStoreException - if a general record store exception occurs
SecurityException - if the MIDlet has read-only access to the RecordStore

deleteRecordStore

public static void deleteRecordStore(String recordStoreName)
throws javax.microedition.rms.RecordStoreException,
javax.microedition.rms.RecordStoreNotFoundException

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 - if a record store-related exception occurred
RecordStoreNotFoundException - if the record store could not be found

enumerateRecords

public javax.microedition.rms.RecordEnumeration enumerateRecords(RecordFilter filter, RecordComparator comparator, boolean keepUpdated)
throws javax.microedition.rms.RecordStoreNotOpenException

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
getLastModified

public long getLastModified() throws javax.microedition.rms.RecordStoreNotOpenException

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 - if the record store is not open

getName

public java.lang.String getName() throws javax.microedition.rms.RecordStoreNotOpenException

Returns the name of this RecordStore.

Returns: the name of this RecordStore

Throws: RecordStoreNotOpenException - if the record store is not open

getNextRecordID

public int getNextRecordID() throws javax.microedition.rms.RecordStoreNotOpenException, javax.microedition.rms.RecordStoreException

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 - if the record store is not open
RecordStoreException - if a different record store-related exception occurred

getNumRecords

public int getNumRecords() throws javax.microedition.rms.RecordStoreNotOpenException

Returns the number of records currently in the record store.

Returns: the number of records currently in the record store

Throws: RecordStoreNotOpenException - if the record store is not open
getRecord

public byte[] getRecord(int recordId)
throws javax.microedition.rms.RecordStoreNotOpenException,
javax.microedition.rms.InvalidRecordIDException,
javax.microedition.rms.RecordStoreException

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 - if the record store is not open
  InvalidRecordIDException - if the recordId is invalid
  RecordStoreException - if a general record store exception occurs

See Also: RecordStore.setRecord(int, byte[], int, int)

getRecord

public int getRecord(int recordId,
byte[] buffer,
int offset)
throws javax.microedition.rms.RecordStoreNotOpenException,
javax.microedition.rms.InvalidRecordIDException,
javax.microedition.rms.RecordStoreException

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 - if the record store is not open
  InvalidRecordIDException - if the recordId is invalid
  RecordStoreException - if a general record store exception occurs
  ArrayIndexOutOfBoundsException - if the record is larger than the buffer supplied

See Also: RecordStore.setRecord(int, byte[], int, int)

getRecordSize

public int getRecordSize(int recordId)
throws javax.microedition.rms.RecordStoreNotOpenException,
javax.microedition.rms.InvalidRecordIDException,
javax.microedition.rms.RecordStoreException

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 - if the record store is not open
  InvalidRecordIDException - if the recordId is invalid
  RecordStoreException - if a general record store exception occurs
**getSize**

```java
public int getSize()
    throws javax.microedition.rms.RecordStoreNotOpenException
```

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 - if the record store is not open

---

**getSizeAvailable**

```java
public int getSizeAvailable()
    throws javax.microedition.rms.RecordStoreNotOpenException
```

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 - if the record store is not open

---

**getVersion**

```java
public int getVersion()
    throws javax.microedition.rms.RecordStoreNotOpenException
```

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 - if the record store is not open

---

**listRecordStores**

```java
public static java.lang.String[] listRecordStores()
```

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**

```java
public static javax.microedition.rms.RecordStore openRecordStore(String recordStoreName,
                      boolean createIfNecessary)
throws javax.microedition.rms.RecordStoreException,
      javax.microedition.rms.RecordStoreFullException,
      javax.microedition.rms.RecordStoreNotFoundException
```

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` - if a record store-related exception occurred
- `RecordStoreNotFoundException` - if the record store could not be found
- `RecordStoreFullException` - if the operation cannot be completed because the record store is full
- `IllegalArgumentException` - if `recordStoreName` is invalid

---

**openRecordStore**

```java
public static javax.microedition.rms.RecordStore openRecordStore(String recordStoreName,
                      boolean createIfNecessary,
                      int authmode,
                      boolean writable)
throws javax.microedition.rms.RecordStoreException,
      javax.microedition.rms.RecordStoreFullException,
      javax.microedition.rms.RecordStoreNotFoundException
```

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` - if a record store-related exception occurred
Since: MIDP 2.0

openRecordStore

public static javax.microedition.rms.RecordStore openRecordStore(String recordStoreName, String vendorName, String suiteName) throws javax.microedition.rms.RecordStoreException, javax.microedition.rms.RecordStoreNotFoundException

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)

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 - if a record store-related exception occurred
- RecordStoreNotFoundException - 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

public void removeRecordListener(RecordListener listener)

Removes the specified RecordListener. If the specified listener is not registered, this method does nothing.

Parameters:
- listener - the RecordChangedListener

See Also: RecordStore.addRecordListener(RecordListener)

setMode

public void setMode(int authmode, boolean writable) throws javax.microedition.rms.RecordStoreException
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 - 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

```java
public void setRecord(int recordId, byte[] newData, int offset, int numBytes)
throws javax.microedition.rms.RecordStoreNotOpenException,
javax.microedition.rms.InvalidRecordIDException,
javax.microedition.rms.RecordStoreException,
javax.microedition.rms.RecordStoreFullException
```

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 - if the record store is not open
- InvalidRecordIDException - if the recordId is invalid
- RecordStoreException - if a general record store exception occurs
- RecordStoreFullException - 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:** `RecordStore.getRecord(int, byte[], int)`
javax.microedition.rms

RecordStoreException

Declaration

public class RecordStoreException extends Exception

Object
   +--Throwable
   |   +--Exception
   |   |   +--javax.microedition.rms.RecordStoreException

Direct Known Subclasses:
javax.microedition.rms.InvalidRecordIDException,
javax.microedition.rms.RecordStoreFullException,
javax.microedition.rms.RecordStoreNotFoundException,
javax.microedition.rms.RecordStoreNotOpenException

Description
Thrown to indicate a general exception occurred in a record store operation.

Since: MIDP 1.0

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>RecordStoreException(String message)</td>
<td>Constructs a new RecordStoreException with the specified detail message.</td>
</tr>
<tr>
<td>public</td>
<td>RecordStoreException()</td>
<td>Constructs a new RecordStoreException with no detail message.</td>
</tr>
</tbody>
</table>

Methods inherited from class Throwable

getMessage, printStackTrace, toString

Methods inherited from class Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

RecordStoreException

public RecordStoreException()

   Constructs a new RecordStoreException with no detail message.

RecordStoreException

public RecordStoreException(String message)

   Constructs a new RecordStoreException with the specified detail message.
Parameters:

message - the detail message
javax.microedition.rms

RecordStoreFullException

Declaration

public class RecordStoreFullException extends RecordStoreException

Object
  ↓--Throwable
  ↓--Exception
  ↓--javax.microedition.rms.RecordStoreException
     ↓--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

Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Constructor FullException(String message)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Constructs a new RecordStoreFullException with the specified detail message.</td>
</tr>
<tr>
<td>Public</td>
<td>Constructor FullException()</td>
</tr>
<tr>
<td></td>
<td>Constructs a new RecordStoreFullException with no detail message.</td>
</tr>
</tbody>
</table>

Methods inherited from class Throwable

getMessage, printStackTrace, toString

Methods inherited from class Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

RecordStoreFullException

public RecordStoreFullException()

Constructs a new RecordStoreFullException with no detail message.

RecordStoreFullException

public RecordStoreFullException(String message)

Constructs a new RecordStoreFullException with the specified detail message.

Parameters:
  message - the detail message
javax.microedition.rms

RecordStoreNotFoundException

Declaration

public class RecordStoreNotFoundException extends RecordStoreException

Object

+-- Throwable
|   +-- Exception
|       +-- javax.microedition.rms.RecordStoreException
|               +-- 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

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>RecordStoreNotFoundException(String message)</td>
</tr>
<tr>
<td></td>
<td>Constructs a new RecordStoreNotFoundException with the specified detail message.</td>
</tr>
<tr>
<td>public</td>
<td>RecordStoreNotFoundException()</td>
</tr>
<tr>
<td></td>
<td>Constructs a new RecordStoreNotFoundException with no detail message.</td>
</tr>
</tbody>
</table>

Methods inherited from class Throwable

getMessage, printStackTrace, toString

Methods inherited from class Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

RecordStoreNotFoundException

public RecordStoreNotFoundException()

Constructs a new RecordStoreNotFoundException with no detail message.

RecordStoreNotFoundException

public RecordStoreNotFoundException(String message)

Constructs a new RecordStoreNotFoundException with the specified detail message.

Parameters:

message - the detail message
javax.microedition.rms

RecordStoreNotOpenException

Declaration

public class RecordStoreNotOpenException extends RecordStoreException

Object

|-- Throwable
   `-- Exception
       |-- javax.microedition.rms.RecordStoreException
           `-- javax.microedition.rms.RecordStoreNotOpenException

Description

Thrown to indicate that an operation was attempted on a closed record store.

Since: MIDP 1.0

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Method details</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>RecordStoreNotOpenException(String message)</td>
</tr>
<tr>
<td></td>
<td>Constructs a new RecordStoreNotOpenException with the specified detail message.</td>
</tr>
<tr>
<td>public</td>
<td>RecordStoreNotOpenException()</td>
</tr>
<tr>
<td></td>
<td>Constructs a new RecordStoreNotOpenException with no detail message.</td>
</tr>
</tbody>
</table>

Methods inherited from class Throwable

getMessage, printStackTrace, toString

Methods inherited from class Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

RecordStoreNotOpenException

public RecordStoreNotOpenException() 

Constructs a new RecordStoreNotOpenException with no detail message.

RecordStoreNotOpenException

public RecordStoreNotOpenException(String message) 

Constructs a new RecordStoreNotOpenException with the specified detail message.

Parameters:
message - the detail message
The Recommended Security Policy for GSM/UMTS Compliant Devices

Addendum to the Mobile Information Device Profile version 2.1

Scope of This Document

This addendum is informative. However, all implementations of MIDP 2.1 on GSM/UMTS compliant devices are expected to comply with this addendum.

This addendum supersedes The Recommended Security Policy in MIDP 2.0 and JTWI (JSR 185)

MIDP 2.1 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 2.1 specifies the common rules for APIs that can be used together with MIDP but are specified outside of MIDP. The MIDP 2.1 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 2.1 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 2.1
- 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 15.2 and 15.3 establish the relationship between the device security policy, different protection domains, and requirements concerning certificate storage on smart cards. Section 15.4 specifies the function groups and identifies the permissions and the APIs that need to be protected using the MIDP 2.1 security framework. Sections 15.5 and 15.6 specify rules that MUST be followed when permissions are granted, and also requirements of user notifications. Section 15.7 specifies the MIDlet behavior during roaming and after changing the smart card. Finally Section 15.8 specifies Revocation Checking
References

1. Connected Limited Device Configuration (CLDC)  
   http://jcp.org/jsr/detail/30.jsp
2. Mobile Information Device Profile (MIDP) 2.1  
   http://jcp.org/jsr/detail/118.jsp
3. HTTP 1.1 Specification  
   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
5. WAP Smart Card Provisioning (SCPROV) WAP-186-ProvSC-20010710-a  
   http://www.wapforum.org/what/technical.htm
6. PKCS#15 v.1.1  
   http://www.rsasecurity.com/rsalabs/pkcs/pkcs-15/
7. USIM, 3GPP TS 31.102: "Characteristics of the USIM applications"  
   http://www.3gpp.org
8. RFC 3280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile  
   http://www.ietf.org/rfc/rfc3280.txt
9. RFC 2119 "Key words for use in RFCs to Indicate Requirement Levels"  
   http://www.ietf.org/rfc/rfc2119.txt
10. Online Certificate Status Protocol Mobile Profile  
    http://www.openmobilealliance.org
11. Online Certificate Status Protocol  
    http://www.ietf.org/rfc/rfc2560.txt?number=2560

Definitions

This document uses definitions based upon those specified in RFC 2119.

<table>
<thead>
<tr>
<th>MUST</th>
<th>The associated definition is an absolute requirement of this specification.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUST NOT</td>
<td>The definition is an absolute prohibition of this specification.</td>
</tr>
<tr>
<td>SHOULD</td>
<td>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.</td>
</tr>
<tr>
<td>SHOULD NOT</td>
<td>Indicates a non-recommended practice. There may exist valid reasons in particular circumstances when the behaviour is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behaviour described with this label.</td>
</tr>
<tr>
<td>MAY</td>
<td>Indicates that an item is truly optional.</td>
</tr>
</tbody>
</table>

15.1 General

GSM/UMTS compliant devices implementing this Recommended Security Policy MUST follow the security framework specified in the MIDP 2.1. Additionally, devices that support trusted MIDlets MUST follow the PKI-based authentication scheme as defined in MIDP 2.1 specification.
15.2 Protection Domains

A protection domain is a set of permissions that can be granted to a MIDlet. A MIDlet suite MUST belong to one and only one protection domain. The representation of a domain and its security policy is implementation-specific.

This document defines four protection domains: the Unidentified Third Party Protection Domain, the Identified Third Party Protection Domain, the Operator Protection Domain and the Manufacturer Protection Domain.

Applications bound to the Unidentified Third Party Protection Domain and the Identified Third Party Protection Domain run in an unprivileged environment wherein the user decides on the permissions given to the application (either by the choice of preferences or by answering to user prompts). In contrast to this, applications in the Operator Protection Domain and the Manufacturer Protection Domain run in a privileged environment. However, the privileges given to an application might still depend on the specific Protection Domain to which the application is bound.

15.3 Authorities on Protection Domains

This section specifies how a MIDlet suite’s protection domain is determined, and introduces Authorities on Protection Domains. It also defines mechanisms for provisioning the device with certificates that are Authority Certificates on the different Protection Domains, as well as device behaviour when certificates change in either of these domains. Finally, recommendations on the operation of Authorities on these domains are given.

A MIDlet suite is assigned to a single Protection Domain based on its signature. As a result, one or more Authorities are responsible for applications in the Protection Domain, as specified in the following section.

Applications that are unsigned as per the MIDP 2.1 security framework MUST be installed under the Unidentified Third Party Protection Domain. For any X.509 Certificate under which an application is signed, the device MUST decide whether the certificate is:

- an Authority Certificate on the Identified Third Party Protection Domain. Applications signed “under the authority[1]” of such a certificate, as per the MIDP 2.1 security framework, MUST be run under the Identified Third Party Protection Domain.
- an Authority Certificate on the Operator Protection Domain. Applications signed “under the authority[1]” of such a certificate, as per the MIDP 2.1 security framework, MUST be run under the Operator Protection Domain.
- an Authority Certificate on the Manufacturer Protection Domain. Applications signed “under the authority[1]” of such a certificate, as per the MIDP 2.1 security framework, MUST be run under the Manufacturer Protection Domain.
- another Certificate Applications signed with a certificate not bound to any protection domains, MUST NOT be installed on the device.

15.3.1 Manufacturer and Operator Protection Domains

Recommendation: Permissions in the Manufacturer and Operator Protection Domains are marked as Allowed so that downloaded MIDlets suites perform consistently with MIDlet suites pre-installed by the manufacturer, in terms of security and prompts to the user, whenever actions that require user acknowledgement are requested by a MIDlet suite. The organization holding the private keys corresponding to an Authority Certificate on a Manufacturer or Operator Protection Domain SHOULD ensure that MIDlets that are signed under their authority seek permission from the user when accessing security vulnerable APIs and functions. Permissions defined by MIDP 2.1 and other APIs, as well as section 15.4 below, give the guidelines of which functions need protection.

15.3.1.1 Manufacturer Protection Domain
The Manufacturer Protection Domain Root Certificate is used to verify manufacturer MIDlet suites. The Manufacturer Protection Domain Root Certificate MUST be mapped onto the security policy for the Manufacturer Protection Domain on the device. A device MUST support the security policy for the Manufacturer Protection Domain.

If the Manufacturer Protection Domain Root Certificate is NOT available on the device, the Manufacturer Protection Domain MUST be disabled.

The Manufacturer Protection Domain Root Certificate can be added, deleted or modified only 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 Protection Domain on the device. MIDlet suites verified by a previous Manufacturer Protection Domain Root Certificate MUST be disabled.

At installation of a MIDlet suite signed under the Authority of the Manufacturer Protection Domain, a compliant implementation MUST present the user with the Organization and Country fields within the Subject field of the signing certificate of a MIDlet suite if these fields are present. If these fields are absent, the implementation SHOULD 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 Organization and Country in all cases. This user notification MUST take place before application installation.

The Manufacturer Protection Domain imposes no restriction on the capabilities specified in the MIDP 2.1 and other JSRs.

15.3.1.2 Operator Protection Domain

An Operator Protection Domain Root Certificate is used to verify operator MIDlet suites. Operator Protection Domain Root Certificates MUST be mapped onto the security policy for the Operator Protection Domain on the device.

A device MUST support the security policy for the Operator Protection Domain.

A device MUST support the mechanism [SCPROV] to read Authority Certificates for the Operator Protection Domain stored in the smart card (for example, SIM, USIM or WIM).

Additionally, the device MUST support Authority Certificates for the Operator Protection Domain stored on the device; for example, a device resident Operator Protection Domain Root Certificate.

There is at a maximum one Operator Protection Domain Root Certificates available at either of two specified locations: the smart card or on the device. For example there may be up to one enabled Operator Protection Domain Root Certificate per smart card and up to one enabled device resident Operator Protection Domain Root Certificate.

However, if Operator Protection Domain Root Certificates are found on the device and on the smart card (e.g. SIM, USIM or WIM) at the same time, the root certificate found on the smart card will have a higher priority than a device resident root certificate. Therefore, it is recommended for an operator to put the same root certificate on its smart cards as root certificates which were previously put on its devices (if that's the case for a progressive introduction of root certificates on the SIM card for example). In this way the operator ensures that existing MIDlets will continue to be able to execute when introducing the smart card which contains the root certificate.

If an Operator Protection Domain Root Certificate is NOT available at the specified location in either the smart card or on the device the Operator Protection Domain MUST be disabled.

Potential consequences of a change of the smart card are described in section 15.7.

The Operator Protection Domain Root Certificate MUST only be deleted or modified by the operator, who may use an update mechanism whose details are outside the scope of this specification.
The user MUST NOT be able to delete Authority Certificates for the Operator Protection Domain.

Any new or updated Operator Protection Domain Root Certificate MUST be associated with the security policy for the Operator Protection Domain on the device.

MIDlet suites verified by a non valid (for example, disabled) Operator Protection Domain Root Certificate MUST be disabled.

Before invoking a MIDlet the device MUST check whether or not the Protection Domain Root Certificate is still valid. If the Protection Domain Root Certificate that verified the MIDlet is no longer valid then the MIDlet MUST NOT be invoked. The device SHOULD then present a suitable message to the user.

Only if the Operator Protection Domain Root Certificate is valid during the installation process the device MUST continue the installation process.

If this condition is not fulfilled the device MUST NOT install the MIDlet and MUST act as described in JSR 118, “Over the Air User Initiated Provisioning Specification” by sending the appropriate error code.

Authority Certificates are read from the Certificate Directory File (CDF) for trusted certificates [WIM]. Authority Certificates found in the trustedCertificates file on the WIM are considered to be Authority Certificates on the Operator Protection Domain or onto the Identified Third Party Protection Domain (see section 3.2.0.1), 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 considered an Authority Certificate on the Operator Protection Domain.

If the trustedUsage field contains the OID:

```
"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)identifiedParty(3)"
```

the certificate is to be considered an Authority Certificate on the Identified Third Party Protection Domain (see section 3.2.1).

Figure 1 summarizes assigning Authority Certificates obtained through [SCPROV] to Protection Domains.
At installation of a MIDlet suite signed under the Authority of the Operator Protection Domain, a compliant implementation MUST present the user with the Organization and Country fields within the Subject field of the signing certificate of a MIDlet suite if these fields are present. If these fields are absent, the implementation SHOULD 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 Organization and Country in all cases. This user notification MUST take place before application installation.

MIDlet suites installed in the Operator Protection Domain MUST store, along with the application itself, a hash of the Authority Certificate under which the certificate used to sign the application was issued. The hash algorithm to be used is the following: starting with the Authority 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 have the correct value and MUST compute the hash themselves. In the event of a smart card change this hash MUST be used by the device to decide when a given MIDlet suite should be disabled, as specified in Section 15.7.

15.3.2 Third Party Protection Domains

Recommendation: All permissions in the Identified Third Party and the Unidentified Third Party Protection Domains are subject to user permission according to the security policies. Differences between the two Third Party Protection Domains are:

- The identification of the Third Party which created the application
- Different default and other user settings

15.3.2.1 Identified Third Party Protection Domain

MIDlet suites in this domain MUST be granted permissions as per the policy set in section 15.4 below. This policy mixes user prompting at runtime as well as user settings in order to ensure the safety of all applications in this protection domain.

Applications that are authenticated using Authority Certificates on the Identified Third Party Protection Domain are mapped to the Identified Third Party Protection Domain. There is no explicit limitation on the number of Identified Third Party Protection Domain Root Certificates available either on the device or at the specified location in the smart card.

All Identified Third Party Protection Domain Root Certificates MUST be mapped onto the security policy for the Identified Third Party Protection Domain on the device. A device MUST implement the Identified Third Party Protection Domain. However, if there are no Identified Third Party Authority Certificates available either on the device or at the specified location in the smart card the Identified Third Party Protection Domain MUST be disabled.

A device SHOULD support the mechanism [SCPROV] to read Authority Certificates on the Identified Third Party Domain stored in the (U)ICC. Additionally, Authority Certificates for the Identified Third Party Domain MAY be stored on the device. A device MUST support at least one Authority Certificate storage mechanism. If both mechanisms are supported, the device MUST use Authority Certificates from both mechanisms.

Authority Certificates are read from the Certificate Directory File (CDF) for trusted certificates [WIM]. Authority Certificates found in the trustedCertificates file on the WIM are considered to be Authority Certificates on the Operator Protection Domain (see section 3.1.2) or onto the Identified Third Party Protection 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
Authority Certificates may be placed in the trustedCertificates Certificate Directory File (CDF) of a WIM, SIM, or USIM or on the device. If Authority 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]. Authority 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.

Any Authority Certificates obtained after device manufacture MUST NOT be used for authentication of MIDlet suites. This does NOT prevent obtaining Identified Third Party Authority Certificates from the specified location in the SIM, USIM or WIM.

At MIDlet suite installation, an implementation MUST present the user with the Organization and Country fields within the Subject field of the signing certificate of a MIDlet suite if these fields are present. If these fields are absent, the implementation SHOULD 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 Organization and Country in all cases. This user notification MUST take place before MIDlet suite installation. When the user is prompted to grant permissions for a MIDlet suite to function groups, the prompt MUST identify the identified source with the appropriate fields within the Subject field of the signing certificate as stated above.

The user MUST be able to delete or disable Identified Third Party Authority Certificates. The deletion of these certificates is only possible for Authority Certificates stored on the device and not for those obtained using [SCPROV]. If an Identified Third Party Authority Certificate is to be deleted, the implementation MUST adequately warn the user of the consequence of the deletion. A disabled Identified Third Party Authority Certificate MUST NOT be used to verify downloaded MIDlet suites. The user MUST be able to re-enable a disabled Identified Third Party Authority Certificate that was previously disabled by the user. Furthermore, if an Identified Third Party Authority Certificate is deleted or disabled (for example, revoked, deleted, or disabled by the user) the Identified Third Party Protection Domain MUST no longer be associated with this Authority Certificate. If the user chooses to delete or disable an Identified Third Party Authority Certificate, the implementation MAY provide an option to delete the MIDlet suites authenticated to it.

The security policy for the Identified Third Party Protection Domain MUST NOT grant any permission on the device as Allowed. All permissions granted by the Identified Third Party Protection Domain MUST 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 Identified Third Party Protection Domain. Table 2, Table 3, and Table 6 specify the mapping of permissions and APIs onto different function groups.

15.3.2.2 Unidentified Third Party Protection Domain

MIDlet suites in this domain MUST be granted permissions as per the policy set in section 15.4 below. This policy mixes user prompting at runtime as well as user settings in order to ensure the safety of all applications in this protection domain. It should be noted that versions of MIDP prior to 2.0.1 referred to this Domain as “Untrusted.”

MIDlets suites that are unsigned will belong to the Unidentified Third Party Protection Domain. A device MUST support the security policy for the Unidentified Third Party Protection Domain. The implementation MUST inform the user whenever a new MIDlet suite is installed in the Unidentified Third Party Protection Domain. The notification MUST indicate that the source of the application cannot be verified. In order for the user to make an informed decision before granting permissions to an application, appropriate information about the source and trustworthiness of the MIDlet suite MUST be made available to the user.

When the user is prompted to grant permissions to an application, the prompt MUST visually indicate whether
the application comes from a trusted source or not. Implementations are recommended to indicate this by showing an icon on the security prompt screen for MIDlet suites that are authenticated by Authority Certificates on the Manufacturer, Operator or Identified Third Party Protection Domain that is different than for MIDlet suites that are not signed or can not be authenticated by all Authority Certificates on the Manufacturer, Operator or Identified Third Party Protection Domain. The recommended icons are shown in figure 2 below.

![Recommended icons for the security prompt for authenticated (left) and unauthenticated MIDlets (right)](image)

### 15.4 Permissions for Downloaded MIDlet Suites

#### 15.4.1 Mapping MIDP 2.1 Permissions onto Function Groups in Protected Domains

A device with a small display may not be able to present all permissions on an API level 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. These so-called function groups are as follows:

**Network/cost-related groups:**

- **Phone Call** – the group represents permissions to any function that results in a voice call.
- **Call Control** – the group represents permissions to any function that results call setup or teardown of a restricted network connection.
- **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.
- **Low Level Net Access** – the group represents permissions to any function that results in an active low level network data connection (for example Sockets, 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.)
- **Restricted Messaging** – the group represents permissions to any function that allows sending or receiving messages to a restricted messaging service (for example, Cell Broadcast, 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.)
- **Authentication** - the group represents permissions to any function that gives a MIDlet suite access to authentication functionality.

**User-privacy-related groups:**

- **Multimedia recording** – the group represents permissions to any function that gives a MIDlet suite the ability
to do any kind of multimedia recording (for example 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.

Smart Card Communication – the group represents permissions to any function that gives a MIDlet suite the ability to communicate with the smart card.

Location – the group represents permissions to any function that gives a MIDlet suite access to Location information.

Landmark - the group represents permissions to any function that gives a MIDlet suite access to Landmark information.

Whenever new features are added to 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 a new function group MUST be defined in this document by requesting an update to this document from MSA or MIDP as appropriate.

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. The new function group SHOULD be sufficiently future-proof to contain new features added by future APIs and should not only concern the features being initially included in it.

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 SHOULD be enforced using the security framework as defined in MIDP 2.1. 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 both Third Party Protection Domains. 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.

Table 2 presents individual permissions defined in the MIDP 2.1, and map to the function groups specified in this section. An individual permission MUST occur in only one function group.

It is recommended that MIDlet suites in the Manufacturer and Operator Protection Domains 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 for Third Party Protection Domains:
<table>
<thead>
<tr>
<th>Function group</th>
<th>Identified Third Party Protection Domain</th>
<th>Unidentified Third Party Protection Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phone Call</td>
<td>default setting</td>
<td>default setting</td>
</tr>
<tr>
<td></td>
<td>Oneshot</td>
<td>Oneshot</td>
</tr>
<tr>
<td></td>
<td>other settings</td>
<td>other settings</td>
</tr>
<tr>
<td></td>
<td>Blanket, Session, No</td>
<td>No</td>
</tr>
<tr>
<td>Net Access</td>
<td>default setting</td>
<td>default setting</td>
</tr>
<tr>
<td></td>
<td>Session</td>
<td>Oneshot</td>
</tr>
<tr>
<td></td>
<td>other settings</td>
<td>other settings</td>
</tr>
<tr>
<td></td>
<td>Blanket, Oneshot, No</td>
<td>Session, No</td>
</tr>
<tr>
<td>Low Level Net Access</td>
<td>default setting</td>
<td>default setting</td>
</tr>
<tr>
<td></td>
<td>Session</td>
<td>Oneshot</td>
</tr>
<tr>
<td></td>
<td>other settings</td>
<td>other settings</td>
</tr>
<tr>
<td></td>
<td>Blanket, Oneshot, No</td>
<td>Session, No</td>
</tr>
<tr>
<td>Messaging</td>
<td>default setting</td>
<td>default setting</td>
</tr>
<tr>
<td></td>
<td>Oneshot</td>
<td>Oneshot</td>
</tr>
<tr>
<td></td>
<td>other settings</td>
<td>other settings</td>
</tr>
<tr>
<td></td>
<td>Blanket, Session, No</td>
<td>No</td>
</tr>
<tr>
<td>Restricted Messaging</td>
<td>default setting</td>
<td>default setting</td>
</tr>
<tr>
<td></td>
<td>Oneshot</td>
<td>Oneshot</td>
</tr>
<tr>
<td></td>
<td>other settings</td>
<td>other settings</td>
</tr>
<tr>
<td></td>
<td>Blanket, Session, No</td>
<td>No</td>
</tr>
<tr>
<td>Application Auto Invocation</td>
<td>default setting</td>
<td>default setting</td>
</tr>
<tr>
<td></td>
<td>Oneshot</td>
<td>Oneshot</td>
</tr>
<tr>
<td></td>
<td>other settings</td>
<td>other settings</td>
</tr>
<tr>
<td></td>
<td>Blanket, Session, No</td>
<td>Session, No</td>
</tr>
<tr>
<td>Local Connectivity</td>
<td>default setting</td>
<td>default setting</td>
</tr>
<tr>
<td></td>
<td>Session</td>
<td>Oneshot</td>
</tr>
<tr>
<td></td>
<td>other settings</td>
<td>other settings</td>
</tr>
<tr>
<td></td>
<td>Blanket, Oneshot, No</td>
<td>Blanket, Session, No</td>
</tr>
<tr>
<td>Multimedia recording</td>
<td>default setting</td>
<td>default setting</td>
</tr>
<tr>
<td></td>
<td>Session</td>
<td>Oneshot</td>
</tr>
<tr>
<td></td>
<td>other settings</td>
<td>other settings</td>
</tr>
<tr>
<td></td>
<td>Blanket, Oneshot, No</td>
<td>Session, No</td>
</tr>
<tr>
<td>Read User Data Access</td>
<td>default setting</td>
<td>default setting</td>
</tr>
<tr>
<td></td>
<td>Oneshot</td>
<td>Oneshot</td>
</tr>
<tr>
<td></td>
<td>other settings</td>
<td>other settings</td>
</tr>
<tr>
<td></td>
<td>Blanket, Session, No</td>
<td>No</td>
</tr>
<tr>
<td>Write User Data Access</td>
<td>default setting</td>
<td>default setting</td>
</tr>
<tr>
<td></td>
<td>Oneshot</td>
<td>Oneshot</td>
</tr>
<tr>
<td></td>
<td>other settings</td>
<td>other settings</td>
</tr>
<tr>
<td></td>
<td>Blanket, Session, No</td>
<td>No</td>
</tr>
<tr>
<td>Location</td>
<td>default setting</td>
<td>default setting</td>
</tr>
<tr>
<td></td>
<td>Session</td>
<td>Oneshot</td>
</tr>
<tr>
<td></td>
<td>other settings</td>
<td>other settings</td>
</tr>
<tr>
<td></td>
<td>Blanket, Oneshot, No</td>
<td>Session, No</td>
</tr>
<tr>
<td>Function group</td>
<td>Identified Third Party Protection Domain</td>
<td>Unidentified Third Party Protection Domain</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>default setting</td>
<td>default setting</td>
</tr>
<tr>
<td>Landmark</td>
<td>Session</td>
<td>Oneshot</td>
</tr>
<tr>
<td></td>
<td>Blanket, Oneshot, No</td>
<td>Session, No</td>
</tr>
<tr>
<td>Smart Card Communication</td>
<td>default setting</td>
<td>default setting</td>
</tr>
<tr>
<td></td>
<td>Session</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Blanket, Oneshot, No</td>
<td>No</td>
</tr>
<tr>
<td>Authentication</td>
<td>default setting</td>
<td>default setting</td>
</tr>
<tr>
<td></td>
<td>Session</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Blanket, Oneshot, No</td>
<td>No</td>
</tr>
<tr>
<td>Call Control</td>
<td>default setting</td>
<td>default setting</td>
</tr>
<tr>
<td></td>
<td>Oneshot</td>
<td>Oneshot</td>
</tr>
<tr>
<td></td>
<td>Blanket, Session, No</td>
<td>No</td>
</tr>
</tbody>
</table>

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 the future for MIDlet suites from the same signer. 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 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. Implementations that support this kind of a feature MUST inform the user during application installation that the security settings for the given signer has previously been saved on the device and will be applied for the newly downloaded application.

If the security policy for the default and other settings of the domain for the Net Access and Low Level Net Access function groups are identical then the low level network access group may be merged into the Net Access function group. When these two function groups are merged, the implementation MUST behave as if all of the permissions are included in the Net Access function group.

If the security policy for the default and other settings of the domain for the Messaging and Restricted Messaging function groups are identical then the Restricted Messaging group may be merged into the Messaging function group. When these two function groups are merged, the implementation MUST behave as if all of the permissions are included in the Messaging function group.

If the security policy for the default and other settings of the domain for the Phone Call and Call Control function groups are identical then the Call Control function group may be merged into the Phone Call function group. When these two function groups are merged, the implementation MUST behave as if all of the permissions are included in the Phone Call function group.

Blanket permission given for some combinations of Function groups can lead to higher risks for the user. For MIDlet suites in the Identified Third Party Protection 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, Low Level Net Access, Messaging, Restricted Messaging, Call Control, or Local Connectivity set to Blanket in combination with any of Multimedia recording or Read User Data Access
This restriction does not apply to the Unidentified Third Party Protection Domain, since these combinations are forbidden in this domain 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 prompt, the implementation MUST present the user with the destination phone number or the destination name 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.

When the "No" setting is selected for the function group permission the implementation MUST behave as follows:

- During application installation it is treated as a user permission. In particular the application installation MUST NOT fail due to this setting being used.
- During application execution the implementation MUST NOT present user prompts when the application tries to access the protected APIs from the function group and the corresponding API call MUST result in a SecurityException.

For the user prompts in Blanket and Session interaction modes the implementation MUST give the user a choice to deny the permission. If the user denies the permission, the implementation MUST remember this answer and MUST NOT present further user prompts until the application is unplugged (in Blanket mode) or terminated (in Session mode).

Table 2: Assigning permissions specified in MIDP 2.1 (JSR 118) to function groups:

<table>
<thead>
<tr>
<th>Permission</th>
<th>Protocol</th>
<th>Function group</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.microedition.io.Connector.https</td>
<td>https</td>
<td>Net Access</td>
</tr>
<tr>
<td>javax.microedition.io.Connector.datagram</td>
<td>datagram</td>
<td>Low Level Net Access</td>
</tr>
<tr>
<td>javax.microedition.io.Connector.datagramreceiver</td>
<td>datagram server (without host)</td>
<td>Low Level Net Access</td>
</tr>
<tr>
<td>javax.microedition.io.Connector.socket</td>
<td>socket</td>
<td>Low Level Net Access</td>
</tr>
<tr>
<td>javax.microedition.io.Connector.serversocket</td>
<td>server socket (without host)</td>
<td>Low Level Net Access</td>
</tr>
<tr>
<td>javax.microedition.io.Connector.ssl</td>
<td>ssl</td>
<td>Low Level Net Access</td>
</tr>
</tbody>
</table>
15.4.2 Network Access Requirements

Unidentified third party applications MUST use the normal HttpConnection and HttpsConnection APIs to access web and secure web services. There are no restrictions on web server port numbers through these interfaces. The implementations augment the protocol so that web servers can identify unidentified third party applications. The Product-Token “UNTRUSTED” refers to the Unidentified Third Party Protection Domain and is retained to maintain backward compatibility with JTWI. The following MUST be implemented:

- The implementation of HttpConnection and HttpsConnection MUST include in the User-Agent header with the Product-Token “UNTRUSTED/1.0”. User-Agent headers supplied by the application MUST NOT be deleted.
- The implementation of SocketConnection using TCP sockets MUST throw java.lang.SecurityException when a MIDlet suite belonging to the Unidentified Third Party Protection Domain attempts to connect on ports 80 and 8080 (http) and 443 (https).
- The implementation of SecureConnection using TCP sockets MUST throw java.lang.SecurityException when a MIDlet suite belonging to the Unidentified Third Party Protection Domain attempts to connect on port 443 (https).
- The implementation of the method DatagramConnection.send MUST throw java.lang.SecurityException when a MIDlet suite belonging to the Unidentified Third Party Protection Domain attempts to send datagrams to any of the ports 9200-9203 (WAP Gateway).
- The above requirements SHOULD be applied regardless of the API used to access the network. For example, the javax.microedition.io.Connector.open and javax.microedition.media.Manager.createPlayer methods SHOULD throw java.lang.SecurityException if access is attempted to these port numbers through a means other than the normal HttpConnection and HttpsConnection APIs.

15.4.3 Wireless Messaging Third Party Domain Policy

This section defines the Third Party Domain policy for the Wireless Messaging API. The permissions in Messaging Function Group are defined in WMA 1.1, Appendix D.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Protocol</th>
<th>Function group</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.wireless.messaging.sms.send</td>
<td>sms</td>
<td>Messaging</td>
</tr>
<tr>
<td>javax.wireless.messaging.sms.receive</td>
<td>sms</td>
<td>Messaging</td>
</tr>
<tr>
<td>javax.microedition.io.Connector.sms</td>
<td>sms</td>
<td>Messaging</td>
</tr>
<tr>
<td>javax.microedition.io.Connector.cbs</td>
<td>cbs</td>
<td>Restricted Messaging</td>
</tr>
<tr>
<td>javax.microedition.io.Connector.mms</td>
<td>mms</td>
<td>Messaging</td>
</tr>
<tr>
<td>javax.wireless.messaging.cbs.receive</td>
<td>cbs</td>
<td>Restricted Messaging</td>
</tr>
</tbody>
</table>
15.4.4 User Prompting
The Third Party Domain messaging policy requires that the user MUST be provided with the destination phone number or a corresponding name when prompting for the messaging function group. If a single API call is mapped to multiple messages (i.e. the implementation supports disassembly or reassembly), then the implementation MUST provide the user with the number of messages that are to be sent. This requirement is to ensure that the user always understands the network costs.

15.4.5 Wireless Messaging Interaction Modes
If the interaction mode is oneshot for the Messaging or Restricted Messaging function group the connections are assigned the blanket mode as shown in TABLE 4.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Function Group Interaction Mode</th>
<th>Permission Interaction Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.microedition.io.Connector.sms</td>
<td>Oneshot</td>
<td>Blanket</td>
</tr>
<tr>
<td>javax.microedition.io.Connector.cbs</td>
<td>Oneshot</td>
<td>Blanket</td>
</tr>
<tr>
<td>javax.microedition.io.Connector.mms</td>
<td>Oneshot</td>
<td>Blanket</td>
</tr>
</tbody>
</table>

If the interaction mode is oneshot for the Messaging or Restricted Messaging function group, then the mode for the permissions for receiving messages is blanket as shown in TABLE 5. Also, the permissions for sending messages are the same as shown in TABLE 5. Thus the interaction mode is oneshot for each message sent by the MIDlet suite. When the interaction mode is no, it applies to all the individual permissions within these Messaging function groups. These requirements apply regardless of the intended recipient, whether it is a user or another application.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Function Group Interaction Mode</th>
<th>Permission Interaction Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.wireless.messaging.sms.send</td>
<td>Oneshot</td>
<td>Oneshot</td>
</tr>
<tr>
<td>javax.wireless.messaging.sms.receive</td>
<td>Oneshot</td>
<td>Blanket</td>
</tr>
<tr>
<td>javax.wireless.messaging.cbs.receive</td>
<td>Oneshot</td>
<td>Blanket</td>
</tr>
<tr>
<td>javax.wireless.messaging.mms.send</td>
<td>Oneshot</td>
<td>Oneshot</td>
</tr>
<tr>
<td>javax.wireless.messaging.mms.receive</td>
<td>Oneshot</td>
<td>Blanket</td>
</tr>
</tbody>
</table>

15.4.6 Mobile Media Third Party Domain Policy
Table 6: Assigning Permissions in the Mobile Media API to Function Groups

<table>
<thead>
<tr>
<th>Permission</th>
<th>Permitted API Calls</th>
<th>Function group</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.microedition.media.control.RecordControl</td>
<td>RecordControl.setRecordLocation()</td>
<td>Multimedia recording</td>
</tr>
<tr>
<td></td>
<td>RecordControl.setRecordStream()</td>
<td></td>
</tr>
<tr>
<td>javax.microedition.media.control.VideoControl</td>
<td>VideoControl.getSnapshot()</td>
<td>Multimedia recording</td>
</tr>
</tbody>
</table>

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 and HTTPS.

15.5 Permissions Granted to a MIDlet Suite by the Authorization Mechanism

Permissions granted to a MIDlet suite are effectively the intersection of the domain permissions, permissions requested in MIDlet-Permissions and the MIDlet-Permissions-Opt attributes. See the section “Granting permissions to trusted MIDlet suites”: The permissions granted to the MIDlet suite are the intersection of the requested permissions with the union of the allowed and user granted permissions. For security reasons the permission attributes SHOULD be included in both the JAR manifest and in the JAD so that the device will have information about the requested permissions before the JAR is downloaded. These attribute values MUST be identical or otherwise the MIDlet suite MUST NOT be installed or invoked. All JAD-only implementations of these attributes (both in accepting policy for devices and in MIDlet suites) SHOULD be deprecated. 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 to any setting available for a given MIDlet suite permission, provided they are in accordance with the implementation notes in Section 15.5 and with default and available sets of user permission types provided as guides in the tables in Section 15.5. 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, the 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.
- The device MUST have a set of function group settings for each MIDlet suite that is in the Unidentified Third Party Protection Domain. The implementation MUST NOT offer a way to set function group settings for several MIDlet suites in the Unidentified Third Party protection domain at the same time.
- A compliant implementation MUST be able to present the MIDlet suite name, version number and Function Group settings to the user for each installed MIDlet suite. An implementation MAY present any additional security related information to the user.
- A function group cannot be a union of permissions with different default settings and other settings. Therefore the tables in Section 15.4 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 security related data MUST NOT be accessible by any Third Party Protection Domain MIDlet. The data MUST include at least the following:

- 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 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.

The user MUST be able to view the stored security related data for an installed application presented in a user-friendly manner.

### 15.6 Requirements on Restricted APIs

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 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 the MIDP 2.1 security framework MUST have permissions defined in those JSRs, and follow the naming rules identified in the MIDP 2.1 Specification, titled “Security for MIDP Applications.”
- The functions that are not deemed security-protected by specification can be accessed explicitly by 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 release 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.

### Presenting Function Group Information

MIDlet suites that are in the Unidentified Third Party Protection domain may request permissions using the attributes in the application descriptor or JAR manifest. The requested user permissions are presented using function groups. The presentation is implementation-specific, but the following rules MUST be followed:

- The user MUST be able to change each function group setting to any allowed setting for the function group listed in Table 1.
- Each function group has allowed settings that apply to each individual permission within the function group. Tables 2 through 6 define the function group for each permission. When the setting for the function group is changed it applies consistently to each permission. The initial value of the setting is the default listed in Table 1.
- The device MUST have a set of function group settings for each untrusted MIDlet suite. The implementation MUST NOT offer a way to set function group settings for several MIDlet suites from the untrusted domain at the same time.
- A compliant implementation MUST be able to present the MIDlet suite name, version number and Function Group settings to the user for each installed MIDlet suite. An implementation MAY present any additional security related information to the user.
15.7 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 one of the two Third Party Protection Domains MUST be preceded by user notification in accordance with user permission settings, for example, showing the phone number or corresponding name the MIDlet is dialing, 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 SHOULD 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.

15.8 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 domain security 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 or at the specified location on the smart card (for example, SIM, USIM or WIM) and are authorized in accordance with the security policy.

If device roaming or a smart card change causes a 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 and Unidentified Third Party Protection Domain MUST NOT be affected by changes of the smart card but MIDlet suites installed in the Operator or Identified Third Party Protection Domain MUST NOT execute if, after a smart card change, the Authority certificate that was used to authenticate the MIDlet suite to the Identified Third Party or Operator Protection Domain is no longer available and until the corresponding Authority certificate becomes available again.

Furthermore, whether a MIDlet suite in the Operator Protection Domain can be executed depends on the verification steps highlighted in the diagram below.
Figure 3: rules to apply before executing an operator domain related MIDlet

Figure 3 serves the purpose of showing possible scenarios and their outcomes with respect to execution of already installed MIDlets in the Operator Protection Domain. In order to conclude whether a MIDlet in the Operator Protection Domain may execute or will have to be disabled the following steps MUST be performed:

1. Compare "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 the Operator Protection Domain Root Certificate that verified a given MIDlet. Compute the 20-byte SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits) to the Authority Certificate stored on the new smart card with the Operator-domain key-usage field (Operator-domain root key hashes). A MIDlet in the Operator Protection Domain is disabled if its authenticating root key hash does not correspond to the new Operator Protection Domain root key hashes generated after the smart card was changed.

2. Compare the MCC+MNC portion of the IMSI that is stored alongside the Operator Protection Domain Root Certificate on the device with the one read from the newly inserted smart card. A MIDlet in the Operator Protection Domain MUST be disabled if the values do not match.

Please note that depending on the possible scenario one or both of the above steps MUST be performed immediately after the smart card change or before execution of the MIDlet after a smart card change.

If a MIDlet suite (either related to Operator or Identified Third Party Protection Domain) cannot be executed due to the smart card change, 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. If the device cannot inform the user in advance of the possibility to execute a MIDlet suite, it MUST inform the user when he tries to execute the MIDlet suite 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 or Identified Third Party Protection Domain. This information SHOULD include the \textit{Subject} field of the Authority Certificate.

An implementation MAY additionally perform steps 1 and/or 2 at any time and accordingly disable MIDlets suites in the Operator Protection

\section*{15.9 Revocation Checking}

For all signed applications, the implementation SHOULD check revocation status using the Online Certificate Status Protocol (OCSP). A compliant implementation SHOULD support OCSP Mobile Profile as specified in the OMA [10]. Alternatively, an implementation MAY implement OCSP according to RFC 2560 [\[1\] Meaning: signed by a certificate issued under the certification hierarchy of this certificate]
ALMANAC LEGEND
The almanac presents classes and interfaces 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. |
| 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. |

<table>
<thead>
<tr>
<th>Modifiers</th>
<th>Access Modifiers</th>
<th>Constructors and Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>○ abstract</td>
<td>♦ protected</td>
<td>✿ constructor</td>
</tr>
<tr>
<td>● final</td>
<td></td>
<td>✿ field</td>
</tr>
<tr>
<td>□ static</td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ static final</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
1.3

Object

Thread

Runnable

javex.realtime

Modifers

Access Modifiers

Constructors and Fields

1.3

RealtimeThread

scheduler

void

addToFeasibility()

currentRealtimeThread()

getScheduler()

RealtimeThread()

RealtimeThread(SchedulingParameters scheduling)

void

sleep(Clock clock, HighResolutionTime time)

throws InterruptedException
```
## Alert

<table>
<thead>
<tr>
<th>Method/Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void addCommand(Command cmd)</td>
<td></td>
</tr>
<tr>
<td>void Alert(String title)</td>
<td></td>
</tr>
<tr>
<td>void Alert(String title, String alertText, Image alertImage, AlertType alertType)</td>
<td></td>
</tr>
<tr>
<td>int FOREVER</td>
<td></td>
</tr>
<tr>
<td>int getDefaultTimeout()</td>
<td></td>
</tr>
<tr>
<td>Image getImage()</td>
<td></td>
</tr>
<tr>
<td>Gauge getIndicator()</td>
<td></td>
</tr>
<tr>
<td>String getString()</td>
<td></td>
</tr>
<tr>
<td>int getTimeOut()</td>
<td></td>
</tr>
<tr>
<td>AlertType getType()</td>
<td></td>
</tr>
<tr>
<td>void removeCommand(Command cmd)</td>
<td></td>
</tr>
<tr>
<td>void setCommandListener(CommandListener l)</td>
<td></td>
</tr>
<tr>
<td>void setImage(Image img)</td>
<td></td>
</tr>
<tr>
<td>void setIndicator(Gauge indicator)</td>
<td></td>
</tr>
<tr>
<td>void setString(String str)</td>
<td></td>
</tr>
<tr>
<td>void setTimeout(int time)</td>
<td></td>
</tr>
<tr>
<td>void setType(AlertType type)</td>
<td></td>
</tr>
</tbody>
</table>

## AlertType

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlertType</td>
<td></td>
</tr>
<tr>
<td>ALARM</td>
<td></td>
</tr>
<tr>
<td>ERROR</td>
<td></td>
</tr>
<tr>
<td>INFO</td>
<td></td>
</tr>
<tr>
<td>CONFIRMATION</td>
<td></td>
</tr>
</tbody>
</table>

[Java Microedition CLDC](https://developer.mozilla.org/en-US/docs/Mobile/Java/Microedition/CLDC)
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean playSound(Display display)</td>
<td>Audio playback method</td>
</tr>
<tr>
<td>AlertType WARNING</td>
<td>Alert type constant</td>
</tr>
</tbody>
</table>

**Canvas**

Canvas is an interface in the javax.microedition.lcdui package that represents a 2D canvas on the device.

```java
// Canvas Interface

public interface Canvas {
    // Constructor
    Canvas();

    // Methods
    void hideNotify();
    boolean isDoubleBuffered();
    boolean hasPointerEvents();
    boolean hasPointerMotionEvents();
    boolean hasRepeatEvents();

    // Constants
    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;

    // Other methods
    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();
}
```
void keyPressed(int keyCode)
void keyReleased(int keyCode)
void keyRepeated(int keyCode)

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)

void serviceRepaints()

void setFullScreenMode(boolean mode)

void showNotify()

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()
Object
  ➞ Throwable
    ➞ Exception
      ➞ 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)
void deleteAll()
int EXCLUSIVE
int getFitPolicy()
Font getFont(int elementNum)
Image getImage(int elementNum)
int getSelectedFlags(boolean[] selectedArray)

int getSelectedIndex()

String getString(int elementNum)

int IMPLICIT

void insert(int elementNum, String stringPart, Image imagePart)

boolean isSelected(int elementNum)

int MULTIPLE

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

javax.microedition.lcdui

ChoiceGroup

Object

Item

ChoiceGroup

Choice

int append(String stringPart, Image imagePart)

ChoiceGroup(String label, int choiceType)

ChoiceGroup(String label, int choiceType, java.lang.String[] stringElements, javax.microedition.lcdui.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)

int getSelectedIndex()

String getString(int elementNum)
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void insert(int elementNum, String stringPart, Image imagePart)</td>
<td>Insert function</td>
</tr>
<tr>
<td>boolean isSelected(int elementNum)</td>
<td>isSelected function</td>
</tr>
<tr>
<td>void set(int elementNum, String stringPart, Image imagePart)</td>
<td>Set function</td>
</tr>
<tr>
<td>void setFitPolicy(int fitPolicy)</td>
<td>Set fit policy function</td>
</tr>
<tr>
<td>void setFont(int elementNum, Font font)</td>
<td>Set font function</td>
</tr>
<tr>
<td>void setSelectedFlags(boolean[] selectedArray)</td>
<td>Set selected flags function</td>
</tr>
<tr>
<td>void setSelectedIndex(int elementNum, boolean selected)</td>
<td>Set selected index function</td>
</tr>
<tr>
<td>int size()</td>
<td>Size function</td>
</tr>
</tbody>
</table>

**Command**

```java
javax.microedition.lcdui
```  

Object

- Command

```java
midp 2.0
```  

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int BACK</td>
<td>Back function</td>
</tr>
<tr>
<td>int CANCEL</td>
<td>Cancel function</td>
</tr>
<tr>
<td>Command(String label, int commandType, int priority)</td>
<td>Command function with label and priority</td>
</tr>
<tr>
<td>Command(String shortLabel, String longLabel, int commandType, int priority)</td>
<td>Command function with short and long label and priority</td>
</tr>
<tr>
<td>int EXIT</td>
<td>Exit function</td>
</tr>
<tr>
<td>int getCommandType()</td>
<td>Get command type function</td>
</tr>
<tr>
<td>String getLabel()</td>
<td>Get label function</td>
</tr>
<tr>
<td>String getLongLabel()</td>
<td>Get long label function</td>
</tr>
<tr>
<td>int getPriority()</td>
<td>Get priority function</td>
</tr>
<tr>
<td>int HELP</td>
<td>Help function</td>
</tr>
<tr>
<td>int ITEM</td>
<td>Item function</td>
</tr>
<tr>
<td>int OK</td>
<td>Ok function</td>
</tr>
<tr>
<td>int SCREEN</td>
<td>Screen function</td>
</tr>
<tr>
<td>int STOP</td>
<td>Stop function</td>
</tr>
</tbody>
</table>

**CommandListener**

```java
javax.microedition.lcdui
```  

CommandListener

```java
midp 2.0
```  

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void commandAction(Command c, Displayable d)</td>
<td>Command action function</td>
</tr>
</tbody>
</table>

**CommConnection**

```java
javax.microedition.io
```  

CommConnection

- StreamConnection
<table>
<thead>
<tr>
<th><strong>Connector</strong></th>
<th>javax.microedition.io</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object</strong></td>
<td><strong>Connector</strong></td>
</tr>
<tr>
<td>Connection</td>
<td>open(String name)</td>
</tr>
<tr>
<td>Connection</td>
<td>open(String name,int mode)</td>
</tr>
<tr>
<td>Connection</td>
<td>open(String name,int mode,boolean timeouts)</td>
</tr>
<tr>
<td>DataInputStream</td>
<td>openDataInputStream(String name)</td>
</tr>
<tr>
<td>DataOutputStream</td>
<td>openDataOutputStream(String name)</td>
</tr>
<tr>
<td>InputStream</td>
<td>openInputStream(String name)</td>
</tr>
<tr>
<td>OutputStream</td>
<td>openOutputStream(String name)</td>
</tr>
<tr>
<td>int</td>
<td>READ</td>
</tr>
<tr>
<td>int</td>
<td>READ_WRITE</td>
</tr>
<tr>
<td>int</td>
<td>WRITE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Control</strong></th>
<th>javax.microedition.media</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Controllable</strong></th>
<th>javax.microedition.media</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controllable</td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>getControl(String controlType)</td>
</tr>
<tr>
<td>Control[]</td>
<td>getControls()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>CustomItem</strong></th>
<th>javax.microedition.lcdui</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>Item</td>
</tr>
<tr>
<td></td>
<td>CustomItem</td>
</tr>
<tr>
<td>CustomItem(String label)</td>
<td>getGameAction(int keyCode)</td>
</tr>
<tr>
<td>CustomItem</td>
<td>getInteractionModes()</td>
</tr>
<tr>
<td>CustomItem</td>
<td>getMinContentHeight()</td>
</tr>
<tr>
<td>CustomItem</td>
<td>getMinContentWidth()</td>
</tr>
<tr>
<td>CustomItem</td>
<td>getPrefContentHeight(int width)</td>
</tr>
</tbody>
</table>
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
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DateField(String label, int mode)</td>
<td></td>
</tr>
<tr>
<td>DateField(String label, int mode, TimeZone timeZone)</td>
<td></td>
</tr>
<tr>
<td>Date getDate()</td>
<td></td>
</tr>
<tr>
<td>int getInputMode()</td>
<td></td>
</tr>
<tr>
<td>void setDate(Date date)</td>
<td></td>
</tr>
<tr>
<td>void setInputMode(int mode)</td>
<td></td>
</tr>
</tbody>
</table>

### Display

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>Display</td>
</tr>
<tr>
<td>javax.microedition.lcdui</td>
<td></td>
</tr>
<tr>
<td>midp 2.0 int ALERT</td>
<td></td>
</tr>
<tr>
<td>void callSerially(Runnable r)</td>
<td></td>
</tr>
<tr>
<td>midp 2.0 int CHOICE_GROUP_ELEMENT</td>
<td></td>
</tr>
<tr>
<td>midp 2.0 int COLOR_BACKGROUND</td>
<td></td>
</tr>
<tr>
<td>midp 2.0 int COLOR_BORDER</td>
<td></td>
</tr>
<tr>
<td>midp 2.0 int COLOR_FOREGROUND</td>
<td></td>
</tr>
<tr>
<td>midp 2.0 int COLOR_HIGHLIGHTED_BACKGROUND</td>
<td></td>
</tr>
<tr>
<td>midp 2.0 int COLOR_HIGHLIGHTED_BORDER</td>
<td></td>
</tr>
<tr>
<td>midp 2.0 int COLOR_HIGHLIGHTED_FOREGROUND</td>
<td></td>
</tr>
<tr>
<td>midp 2.0 boolean flashBacklight(int duration)</td>
<td></td>
</tr>
<tr>
<td>midp 2.0 int getBestImageHeight(int imageType)</td>
<td></td>
</tr>
<tr>
<td>midp 2.0 int getBestImageWidth(int imageType)</td>
<td></td>
</tr>
<tr>
<td>midp 2.0 int getBorderStyle(boolean highlighted)</td>
<td></td>
</tr>
<tr>
<td>midp 2.0 int getColor(int colorSpecifier)</td>
<td></td>
</tr>
<tr>
<td>Displayable int getCurrent()</td>
<td></td>
</tr>
<tr>
<td>boolean isColor()</td>
<td></td>
</tr>
<tr>
<td>midp 2.0 int LIST_ELEMENT</td>
<td></td>
</tr>
<tr>
<td>midp 2.0 int numAlphaLevels()</td>
<td></td>
</tr>
<tr>
<td>int numColors()</td>
<td></td>
</tr>
<tr>
<td>void setCurrent(Alert alert, Displayable nextDisplayable)</td>
<td></td>
</tr>
<tr>
<td>void setCurrent(Displayable nextDisplayable)</td>
<td></td>
</tr>
</tbody>
</table>
midp 2.0  void  setCurrentItem(Item item)
midp 2.0  boolean  vibrate(int duration)

<table>
<thead>
<tr>
<th>Displayable</th>
<th>javax.microedition.lcdui</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>➞ Displayable</td>
</tr>
<tr>
<td></td>
<td>void addCommand(Command cmd)</td>
</tr>
<tr>
<td>midp 2.0</td>
<td>int getHeight()</td>
</tr>
<tr>
<td>midp 2.0</td>
<td>Ticker getTicker()</td>
</tr>
<tr>
<td>midp 2.0</td>
<td>String getTitle()</td>
</tr>
<tr>
<td>midp 2.0</td>
<td>int getWidth()</td>
</tr>
<tr>
<td></td>
<td>boolean isShown()</td>
</tr>
<tr>
<td>midp 2.0</td>
<td>void removeCommand(Command cmd)</td>
</tr>
<tr>
<td></td>
<td>void setCommandListener(CommandListener l)</td>
</tr>
<tr>
<td>midp 2.0</td>
<td>void setTicker(Ticker ticker)</td>
</tr>
<tr>
<td>midp 2.0</td>
<td>void setTitle(String s)</td>
</tr>
<tr>
<td>midp 2.0</td>
<td>➞ void sizeChanged(int w,int h)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Font</th>
<th>javax.microedition.lcdui</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>➞ Font</td>
</tr>
<tr>
<td></td>
<td>int charsWidth(char[],int offset,int length)</td>
</tr>
<tr>
<td></td>
<td>int charWidth(char ch)</td>
</tr>
<tr>
<td></td>
<td>int FACE_MONOSPACEX</td>
</tr>
<tr>
<td></td>
<td>int FACE_PROPORTIONAL</td>
</tr>
<tr>
<td></td>
<td>int FACE_SYSTEM</td>
</tr>
<tr>
<td>midp 2.0</td>
<td>int FONT_INPUT_TEXT</td>
</tr>
<tr>
<td>midp 2.0</td>
<td>int FONT_STATIC_TEXT</td>
</tr>
<tr>
<td></td>
<td>int getBaselinePosition()</td>
</tr>
<tr>
<td></td>
<td>Font getDefaultFont()</td>
</tr>
<tr>
<td></td>
<td>int getFace()</td>
</tr>
<tr>
<td>midp 2.0</td>
<td>Font getFont(int fontSpecifier)</td>
</tr>
<tr>
<td></td>
<td>Font getFont(int face,int style,int size)</td>
</tr>
<tr>
<td></td>
<td>int getHeight()</td>
</tr>
</tbody>
</table>
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 item)

void delete(int itemNum)

void deleteAll()

* Form(String title)

* Form(String title,javax.microedition.lcdui.Item[] items)

int get(int itemNum)

int getHeight()

int getWidth()

void insert(int itemNum,Item item)

void set(int itemNum,Item item)

void setItemStateListener(ItemStateListener iListener)
### GameCanvas

Object

- Displayable
- Canvas
- GameCanvas

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int size()</td>
<td></td>
</tr>
<tr>
<td>void flushGraphics()</td>
<td></td>
</tr>
<tr>
<td>void flushGraphics(int x, int y, int width, int height)</td>
<td></td>
</tr>
<tr>
<td>int DOWN_PRESSED</td>
<td></td>
</tr>
<tr>
<td>int FIRE_PRESSED</td>
<td></td>
</tr>
<tr>
<td>GameCanvas(boolean suppressKeyEvents)</td>
<td></td>
</tr>
<tr>
<td>Graphics getGraphics()</td>
<td></td>
</tr>
<tr>
<td>int getKeyStates()</td>
<td></td>
</tr>
<tr>
<td>int LEFT_PRESSED</td>
<td></td>
</tr>
<tr>
<td>void paint(Graphics g)</td>
<td></td>
</tr>
<tr>
<td>int RIGHT_PRESSED</td>
<td></td>
</tr>
<tr>
<td>int UP_PRESSED</td>
<td></td>
</tr>
</tbody>
</table>

### Gauge

Object

- Item
- Gauge

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>midp 2.0 int CONTINUOUS_IDALE</td>
<td></td>
</tr>
<tr>
<td>midp 2.0 int CONTINUOUS_RUNNING</td>
<td></td>
</tr>
<tr>
<td>Gauge(String label, boolean interactive, int maxvalue, int initialValue)</td>
<td></td>
</tr>
<tr>
<td>int getMaxValue()</td>
<td></td>
</tr>
<tr>
<td>int getValue()</td>
<td></td>
</tr>
<tr>
<td>midp 2.0 int INCREMENTAL_IDALE</td>
<td></td>
</tr>
<tr>
<td>midp 2.0 int INCREMENTAL_UPDATING</td>
<td></td>
</tr>
<tr>
<td>midp 2.0 int INDEFINITE</td>
<td></td>
</tr>
<tr>
<td>boolean isInteractive()</td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td></td>
</tr>
<tr>
<td>void maxValue(int maxValue)</td>
<td></td>
</tr>
<tr>
<td>void setValue(int value)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Graphics javax.microedition.lcdui</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object ➤ Graphics</td>
</tr>
<tr>
<td>✓ int BASELINE</td>
</tr>
<tr>
<td>✓ int BOTTOM</td>
</tr>
<tr>
<td>✓ void clipRect(int x, int y, int width, int height)</td>
</tr>
<tr>
<td>✓ copyArea(int x_src, int y_src, int width, int height, int x_dest, int y_dest, int anchor)</td>
</tr>
<tr>
<td>✓ int DOTTED</td>
</tr>
<tr>
<td>✓ void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)</td>
</tr>
<tr>
<td>✓ void drawChar(char character, int x, int y, int anchor)</td>
</tr>
<tr>
<td>✓ void drawChars(char[] data, int offset, int length, int x, int y, int anchor)</td>
</tr>
<tr>
<td>✓ void drawImage(Image img, int x, int y, int anchor)</td>
</tr>
<tr>
<td>✓ void drawLine(int x1, int y1, int x2, int y2)</td>
</tr>
<tr>
<td>✓ void drawRect(int x, int y, int width, int height)</td>
</tr>
<tr>
<td>✓ drawRegion(Image src, int x_src, int y_src, int width, int height, int x_dest, int y_dest, int anchor)</td>
</tr>
<tr>
<td>✓ void drawRGB(int[] rgbData, int offset, int scanlength, int x, int y, int width, int height, boolean processAlpha)</td>
</tr>
<tr>
<td>✓ void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)</td>
</tr>
<tr>
<td>✓ void drawString(String str, int x, int y, int anchor)</td>
</tr>
<tr>
<td>✓ void drawSubstring(String str, int offset, int len, int x, int y, int anchor)</td>
</tr>
<tr>
<td>✓ int getBlueComponent()</td>
</tr>
<tr>
<td>✓ int getClipHeight()</td>
</tr>
</tbody>
</table>
int getClipWidth()
int getClipX()
int getClipY()
int getColor()
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

<table>
<thead>
<tr>
<th>HttConnection</th>
<th>javax.microedition.io</th>
</tr>
</thead>
<tbody>
<tr>
<td>HttpConnection</td>
<td>ContentConnection</td>
</tr>
</tbody>
</table>

<p>| String               | GET                      |
| long                 | getDate()                |
| long                 | getExpiration()          |
| String               | getFile()                |
| String               | getHeaderValue(int n)    |</p>
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>String getHeaderField(String name)</code></td>
<td>Returns the value of the header field by name.</td>
</tr>
<tr>
<td><code>long getHeaderFieldDate(String name,long def)</code></td>
<td>Returns the value of the header field by name,</td>
</tr>
<tr>
<td></td>
<td>with a default value.</td>
</tr>
<tr>
<td><code>int getHeaderFieldInt(String name,int def)</code></td>
<td>Returns the value of the header field by name,</td>
</tr>
<tr>
<td></td>
<td>with a default value.</td>
</tr>
<tr>
<td><code>String getHeaderFieldKey(int n)</code></td>
<td>Returns the key of the header field by index.</td>
</tr>
<tr>
<td><code>String getHost()</code></td>
<td>Returns the host of the request.</td>
</tr>
<tr>
<td><code>long getLastModified()</code></td>
<td>Returns the last modified time of the response.</td>
</tr>
<tr>
<td><code>int getPort()</code></td>
<td>Returns the port of the request.</td>
</tr>
<tr>
<td><code>String getProtocol()</code></td>
<td>Returns the protocol of the request.</td>
</tr>
<tr>
<td><code>String getQuery()</code></td>
<td>Returns the query part of the request URL.</td>
</tr>
<tr>
<td><code>String getRef()</code></td>
<td>Returns the reference part of the request URL.</td>
</tr>
<tr>
<td><code>String getRequestMethod()</code></td>
<td>Returns the request method.</td>
</tr>
<tr>
<td><code>String getRequestProperty(String key)</code></td>
<td>Returns the value of the request property by key.</td>
</tr>
<tr>
<td><code>int getResponseCode()</code></td>
<td>Returns the response code.</td>
</tr>
<tr>
<td><code>String getResponseMessage()</code></td>
<td>Returns the response message.</td>
</tr>
<tr>
<td><code>String getURL()</code></td>
<td>Returns the URL of the request.</td>
</tr>
<tr>
<td><code>String HEAD</code></td>
<td>Returns the string <code>HEAD</code>.</td>
</tr>
<tr>
<td><code>int HTTP_ACCEPTED</code></td>
<td>Returns the HTTP status code for accepted.</td>
</tr>
<tr>
<td><code>int HTTP_BAD_GATEWAY</code></td>
<td>Returns the HTTP status code for bad gateway.</td>
</tr>
<tr>
<td><code>int HTTP_BAD_METHOD</code></td>
<td>Returns the HTTP status code for bad method.</td>
</tr>
<tr>
<td><code>int HTTP_BAD_REQUEST</code></td>
<td>Returns the HTTP status code for bad request.</td>
</tr>
<tr>
<td><code>int HTTP_CLIENT_TIMEOUT</code></td>
<td>Returns the HTTP status code for client timeout.</td>
</tr>
<tr>
<td><code>int HTTP_CONFLICT</code></td>
<td>Returns the HTTP status code for conflict.</td>
</tr>
<tr>
<td><code>int HTTP_CREATED</code></td>
<td>Returns the HTTP status code for created.</td>
</tr>
<tr>
<td><code>int HTTP_ENTITY_TOO_LARGE</code></td>
<td>Returns the HTTP status code for entity too large</td>
</tr>
<tr>
<td><code>int HTTP_EXPECT_FAILED</code></td>
<td>Returns the HTTP status code for expect failed.</td>
</tr>
<tr>
<td><code>int HTTP_FORBIDDEN</code></td>
<td>Returns the HTTP status code for forbidden.</td>
</tr>
<tr>
<td><code>int HTTP_GATEWAY_TIMEOUT</code></td>
<td>Returns the HTTP status code for gateway timeout.</td>
</tr>
<tr>
<td><code>int HTTP_GONE</code></td>
<td>Returns the HTTP status code for gone.</td>
</tr>
<tr>
<td><code>int HTTP_INTERNAL_ERROR</code></td>
<td>Returns the HTTP status code for internal error.</td>
</tr>
<tr>
<td><code>int HTTP_LENGTH_REQUIRED</code></td>
<td>Returns the HTTP status code for length required.</td>
</tr>
<tr>
<td><code>int HTTP_MOVED_PERM</code></td>
<td>Returns the HTTP status code for moved permanent.</td>
</tr>
<tr>
<td><code>int HTTP_MOVED_TEMP</code></td>
<td>Returns the HTTP status code for moved temporary.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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)
void setRequestProperty(String key, String value)

HttpsConnection
javax.microedition.io
HttpConnection

int getPort()
SecurityInfo getSecurityInfo()

IllegalStateException
java.lang
<table>
<thead>
<tr>
<th>Throwable</th>
<th>Exception</th>
<th>RuntimeException</th>
<th>IllegalStateException</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td></td>
<td></td>
<td>IllegalStateException()</td>
</tr>
<tr>
<td>*</td>
<td></td>
<td></td>
<td>IllegalStateException(String s)</td>
</tr>
</tbody>
</table>

**Image**  
**javax.microedition.lcdui**

<table>
<thead>
<tr>
<th>Object</th>
<th>javax.microedition.lcdui.Image</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Image</td>
</tr>
<tr>
<td></td>
<td>createImage(byte[] imageData, int imageOffset, int imageLength)</td>
</tr>
<tr>
<td></td>
<td>createImage(Image source)</td>
</tr>
<tr>
<td></td>
<td>createImage(Image image, int x, int y, int width, int height, int transform)</td>
</tr>
<tr>
<td>midp 2.0</td>
<td>createImage(InputStream stream)</td>
</tr>
<tr>
<td>midp 2.0</td>
<td>createImage(int width, int height)</td>
</tr>
<tr>
<td>midp 2.0</td>
<td>createImage(String name)</td>
</tr>
<tr>
<td>midp 2.0</td>
<td>createRGBImage(int[] rgb, int width, int height, boolean processAlpha)</td>
</tr>
<tr>
<td>Graphics</td>
<td>getGraphics()</td>
</tr>
<tr>
<td></td>
<td>int getHeight()</td>
</tr>
<tr>
<td></td>
<td>getRGB(int[] rgbData, int offset, int scanlength, int x, int y, int width, int height)</td>
</tr>
<tr>
<td></td>
<td>void rgbData(int width, int height)</td>
</tr>
<tr>
<td></td>
<td>int getWidth()</td>
</tr>
<tr>
<td></td>
<td>boolean isMutable()</td>
</tr>
</tbody>
</table>

**ImageItem**  
**javax.microedition.lcdui**

<table>
<thead>
<tr>
<th>Object</th>
<th>javax.microedition.lcdui.ImageItem</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Item</td>
</tr>
<tr>
<td></td>
<td>ImageItem</td>
</tr>
<tr>
<td></td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>getAltText()</td>
</tr>
<tr>
<td></td>
<td>int getAppearanceMode()</td>
</tr>
<tr>
<td></td>
<td>Image getImage()</td>
</tr>
<tr>
<td></td>
<td>int getLayout()</td>
</tr>
<tr>
<td>midp 2.0</td>
<td>ImageItem(String label, Image img, int layout, String altText)</td>
</tr>
<tr>
<td>midp 2.0</td>
<td>ImageItem(String label, Image image, int layout, String altText, int appearanceMode)</td>
</tr>
</tbody>
</table>
### InvalidRecordIDException

<table>
<thead>
<tr>
<th>javax.microedition.rms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
</tr>
<tr>
<td>➡ Throwable</td>
</tr>
<tr>
<td>➡ Exception</td>
</tr>
<tr>
<td>➡ RecordStoreException</td>
</tr>
<tr>
<td>➡ InvalidRecordIDException</td>
</tr>
</tbody>
</table>

| * | InvalidRecordIDException() |
| * | InvalidRecordIDException(String message) |

### Item

<table>
<thead>
<tr>
<th>javax.microedition.lcdui</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
</tr>
<tr>
<td>➡ Item</td>
</tr>
</tbody>
</table>

| 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            |

| int LAYOUT_CENTER |
| int LAYOUT_DEFAULT |
| int LAYOUT_LEFT |
| int LAYOUT_NEWFLINE_AFTER |
| int LAYOUT_NEWFLINE_BEFORE |
| int LAYOUT_RIGHT |

void setAltText(String text)
void setImage(Image img)
void setLayout(int layout)
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)
midp 2.0 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()
```
<table>
<thead>
<tr>
<th>boolean</th>
<th>isVisible()</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>move(dx,dy)</td>
</tr>
<tr>
<td>void</td>
<td>paint(g)</td>
</tr>
<tr>
<td>void</td>
<td>setPosition(x,y)</td>
</tr>
<tr>
<td>void</td>
<td>setVisible(visible)</td>
</tr>
</tbody>
</table>

**LayerManager**

```java
javax.microedition.lcdui.game
```

- **Object**
  - LayerManager

```java
void append(Layer l)
Layer getLayerAt(int index)
int getSize()
void insert(Layer l, int index)
LayerManager()
void paint(Graphics g, int x, int y)
void remove(Layer l)
void setViewWindow(int x, int y, int width, int height)
```

**List**

```java
javax.microedition.lcdui
```

- **Object**
  - Displayable
  - Screen
  - List
  - Choice

```java
int append(String stringPart, Image imagePart)
void delete(int elementNum)
void deleteAll()
int getFitPolicy()
Font getFont(int elementNum)
Image getImage(int elementNum)
int getSelectedFlags(boolean[] selectedArray)
int getSelectedIndex()
String getString(int elementNum)
void insert(int elementNum, String stringPart, Image imagePart)
boolean isSelected(int elementNum)
List(String title, int listType)
```
<table>
<thead>
<tr>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>List(String title, int listType, java.lang.String[] stringElements, javax.microedition.lcdui.Image[] imageElements)</td>
</tr>
<tr>
<td>void removeCommand(Command cmd)</td>
</tr>
<tr>
<td>Command SELECT_COMMAND</td>
</tr>
<tr>
<td>void set(int elementNum, String stringPart, Image imagePart)</td>
</tr>
<tr>
<td>void setFitPolicy(int fitPolicy)</td>
</tr>
<tr>
<td>void setFont(int elementNum, Font font)</td>
</tr>
<tr>
<td>void setSelectCommand(Command command)</td>
</tr>
<tr>
<td>void setSelectedFlags(boolean[] selectedArray)</td>
</tr>
<tr>
<td>void setSelectedIndex(int elementNum, boolean selected)</td>
</tr>
<tr>
<td>int size()</td>
</tr>
</tbody>
</table>

### Manager

<table>
<thead>
<tr>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manager javax.microedition.media</td>
</tr>
<tr>
<td>void createPlayer(InputStream stream, String type)</td>
</tr>
<tr>
<td>void createPlayer(String locator)</td>
</tr>
<tr>
<td>String[] getSupportedContentTypes(String protocol)</td>
</tr>
<tr>
<td>String[] getSupportedProtocols(String content_type)</td>
</tr>
<tr>
<td>void playTone(int note, int duration, int volume)</td>
</tr>
<tr>
<td>String TONE_DEVICE_LOCATOR</td>
</tr>
</tbody>
</table>

### MediaException

<table>
<thead>
<tr>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MediaException javax.microedition.media</td>
</tr>
<tr>
<td>void checkPermission(String permission)</td>
</tr>
<tr>
<td>void destroyApp(boolean unconditional)</td>
</tr>
<tr>
<td>String getAppProperty(String key)</td>
</tr>
</tbody>
</table>
**MIDlet()**

void notifyDestroyed()

void notifyPaused()

void pauseApp()

boolean platformRequest(String URL)

void resumeRequest()

void startApp()

---

**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()

int PREFETCHED

void realize()

int REALIZED

void removePlayerListener(PlayerListener playerListener)

void setLoopCount(int count)

long setMediaTime(long now)

void start()
void stop()

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

PushRegistry

Object
javax.microedition.io

PushRegistry

getFilter(String connection)

getMIDlet(String connection)

listConnections(boolean available)

registerAlarm(String midlet,long time)

registerConnection(String connection,String midlet,String filter)

unregisterConnection(String connection)

RecordComparator
javax.microedition.rms

RecordComparator

int compare(byte[] rec1,byte[] rec2)

int EQUIVALENT

int Follows

int PRECEDES
### RecordEnumeration

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void destroy()</td>
<td></td>
</tr>
<tr>
<td>boolean hasNextElement()</td>
<td></td>
</tr>
<tr>
<td>boolean hasPreviousElement()</td>
<td></td>
</tr>
<tr>
<td>boolean isKeptUpdated()</td>
<td></td>
</tr>
<tr>
<td>void keepUpdated(boolean updated)</td>
<td></td>
</tr>
<tr>
<td>byte[] nextRecord()</td>
<td></td>
</tr>
<tr>
<td>int nextRecordId()</td>
<td></td>
</tr>
<tr>
<td>int numRecords()</td>
<td></td>
</tr>
<tr>
<td>byte[] previousRecord()</td>
<td></td>
</tr>
<tr>
<td>int previousRecordId()</td>
<td></td>
</tr>
<tr>
<td>void rebuild()</td>
<td></td>
</tr>
<tr>
<td>void reset()</td>
<td></td>
</tr>
</tbody>
</table>

### RecordFilter

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean matches(byte[] candidate)</td>
<td></td>
</tr>
</tbody>
</table>

### RecordListener

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void recordAdded(RecordStore recordStore, int recordId)</td>
<td></td>
</tr>
<tr>
<td>void recordChanged(RecordStore recordStore, int recordId)</td>
<td></td>
</tr>
<tr>
<td>void recordDeleted(RecordStore recordStore, int recordId)</td>
<td></td>
</tr>
</tbody>
</table>

### RecordStore

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int addRecord(byte[] data, int offset, int numBytes)</td>
<td></td>
</tr>
<tr>
<td>void addRecordListener(RecordListener listener)</td>
<td></td>
</tr>
<tr>
<td>int AUTHMODE_ANY</td>
<td></td>
</tr>
<tr>
<td>int AUTHMODE_PRIVATE</td>
<td></td>
</tr>
<tr>
<td>void closeRecordStore()</td>
<td></td>
</tr>
<tr>
<td>void deleteRecord(int recordId)</td>
<td></td>
</tr>
<tr>
<td>void deleteRecordStore(String recordStoreName)</td>
<td></td>
</tr>
</tbody>
</table>
enumerateRecords(RecordFilter filter, RecordComparator comparator, boolean keepUpdated)

long getLastModified()

String getName()

int getNextRecordId()

int getNumRecords()

byte[] getRecord(int recordId)

int getRecord(int recordId, byte[] buffer, int offset)

int getRecordSize(int recordId)

int getSize()

int getSizeAvailable()

int getVersion()

String[] listRecordStores()

javax.microedition.rms

RecordStoreException

javax.microedition.rms

RecordStoreException(String message)

javax.microedition.rms

RecordStoreFullException

javax.microedition.rms
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RecordStoreFullException()</td>
<td></td>
</tr>
<tr>
<td>RecordStoreFullException(String message)</td>
<td></td>
</tr>
<tr>
<td>RecordStoreNotFoundException()</td>
<td></td>
</tr>
<tr>
<td>RecordStoreNotFoundException(String message)</td>
<td></td>
</tr>
<tr>
<td>RecordStoreNotOpenException()</td>
<td></td>
</tr>
<tr>
<td>RecordStoreNotOpenException(String message)</td>
<td></td>
</tr>
</tbody>
</table>

### RecordStoreNotFoundException

- `java.microedition.rms`
- `Object` (extends `Throwable`, `Exception`, `RecordStoreException`, `RecordStoreNotFoundException`)

### RecordStoreNotOpenException

- `java.microedition.rms`
- `Object` (extends `Throwable`, `Exception`, `RecordStoreException`, `RecordStoreNotOpenException`)

### Screen

- `javax.microedition.lcdui`
- `Object` (extends `Displayable`, `Screen`)

### SecureConnection

- `javax.microedition.io`
- `SecureConnection` (extends `SocketConnection`, `SecurityInfo`, `getSecurityInfo()`)

### SecurityInfo

- `javax.microedition.io`
- `SecurityInfo`

- `String` `getCipherSuite()`
- `String` `getProtocolName()`
- `String` `getProtocolVersion()`
- `Certificate` `getServerCertificate()`

### ServerSocketConnection

- `javax.microedition.io`
- `ServerSocketConnection` (extends `StreamConnectionNotifier`, `String`, `getLocalAddress()`)`
### SocketConnection

**Description:**

- `byte DELAY` - String `getAddress()`
- `String getAddress()`
- `String getLocalAddress()`
- `int getLocalPort()`
- `int getPort()`
- `int getSocketOption(byte option)`
- `byte KEEPALIVE`
- `byte LINGER`
- `byte RCVBUF`
- `void setSocketOption(byte option, int value)`
- `byte SNDBUF`

**Class:**

- `SocketConnection`
- `StreamConnection`

**Package:**

- `javax.microedition.io`

### Spacer

**Description:**

- `void addCommand(Command cmd)`
- `void setDefaultCommand(Command cmd)`
- `void setLabel(String label)`
- `void setMinimumSize(int minWidth, int minHeight)`
- `void Spacer(int minWidth, int minHeight)`

**Class:**

- `Spacer`

**Package:**

- `javax.microedition.lcdui`

### Sprite

**Description:**

- `boolean collidesWith(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)`

**Class:**

- `Sprite`

**Package:**

- `javax.microedition.lcdui.game`
```java
int getFrame()
int getFrameSequenceLength()
int getRawFrameCount()
int getRefPixelX()
int getRefPixelY()
void nextFrame()

void paint(Graphics g)
void prevFrame()
void setFrame(int sequenceIndex)
void setFrameSequence(int[] sequence)
void setImage(Image img, int frameWidth, int frameHeight)
void setRefPixelPosition(int x, int y)
void setTransform(int transform)

Sprite(Image image)
Sprite(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

getAppearanceMode()
setFont(Font font)
```
void setText(String text)

StringItem(String label, String text)

midp 2.0

StringItem(String label, String text, int appearanceMode)

**TextBox**

Object

➥ Displayable

➥ Screen

➥ TextBox

java.util.*

java.microedition.lcdui

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)

void setInitialInputMode(String characterSubset)

int setMaxSize(int maxSize)

void setString(String text)

int size()

void setString(String title, String text, int maxSize, int constraints)

**TextField**

Object

➥ Item

➥ TextField

java.util.*

java.microedition.lcdui

int ANY

int CONSTRAINT_MASK

int DECIMAL

void delete(int offset, int length)

int EMAILADDR

int getCaretposition()
**TextField(String label,String text,int maxSize,int constraints)**

- `int getChars(char[] data)`
- `int getConstraints()`
- `int getMaxSize()`
- `String getString()`

**midp 2.0**

- `int INITIAL_CAPS_SENTENCE`
- `int INITIAL_CAPS_WORD`
- `void insert(char[] data,int offset,int length,int position)`
- `void insert(String src,int position)`
- `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)`
- `void setInitialInputMode(String characterSubset)`
- `int setSize()`
- `void setString(String text)`
- `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**
```java
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(Graphics g)
void setAnimatedTile(int animatedTileIndex, int staticTileIndex)
void setCell(int col, int row, int tileIndex)
void setStaticTileSet(Image image, int tileWidth, int tileHeight)

TiledLayer(int columns, int rows, 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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte BLOCK_END</td>
<td>setSequence(byte[] sequence)</td>
</tr>
<tr>
<td>byte BLOCK_START</td>
<td></td>
</tr>
<tr>
<td>byte C4</td>
<td>void setSequence(byte[] sequence)</td>
</tr>
<tr>
<td>byte PLAY_BLOCK</td>
<td></td>
</tr>
<tr>
<td>byte REPEAT</td>
<td>void SILENCE</td>
</tr>
<tr>
<td>byte RESOLUTION</td>
<td>void TEMPO</td>
</tr>
<tr>
<td>byte SET_VOLUME</td>
<td>void VERSION</td>
</tr>
</tbody>
</table>

## UDPDataGramConnection

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String getLocalAddress()</td>
<td></td>
</tr>
<tr>
<td>int getLocalPort()</td>
<td></td>
</tr>
</tbody>
</table>

## VolumeControl

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int getLevel()</td>
<td></td>
</tr>
<tr>
<td>boolean isMuted()</td>
<td></td>
</tr>
<tr>
<td>int setLevel(int level)</td>
<td></td>
</tr>
<tr>
<td>void setMute(boolean mute)</td>
<td></td>
</tr>
</tbody>
</table>
Index

A
addCommand(Command)
of javax.microedition.lcdui.Displayable 224
of javax.microedition.lcdui.Item 296
of javax.microedition.lcdui.Alert 142
of javax.microedition.lcdui.Spacer 315
addPlayerListener(PlayerListener)
of javax.microedition.media.Player 405
addRecord(byte[], int, int)
of javax.microedition.rms.RecordStore 478
addRecordListener(RecordListener)
of javax.microedition.rms.RecordStore 478
ALARM
of javax.microedition.lcdui.AlertType 147
ALERT
of javax.microedition.lcdui.Display 214
ALERT 138
of javax.microedition.lcdui 138
Alert(String)
of javax.microedition.lcdui 141
Alert(String, String, Image, AlertType)
of javax.microedition.lcdui 141
AlertType 146
AlertType()
of javax.microedition.lcdui 147
ANY
of javax.microedition.lcdui.TextField 334
append(Image)
of javax.microedition.lcdui.Form 240
append(Item)
of javax.microedition.lcdui.Form 240
append(Layer)
of javax.microedition.lcdui.game.LayerManager 360
append(String)
of javax.microedition.lcdui.Form 240
append(String, Image)
of javax.microedition.lcdui.List 307
of javax.microedition.lcdui.ChoiceGroup 178
of javax.microedition.lcdui.Choice 170
AUTHMODE_ANY
of javax.microedition.rms.RecordStore 477
AUTHMODE_PRIVATE
of javax.microedition.rms.RecordStore 477
B
BACK
of javax.microedition.lcdui.Command 186
BAD_EXTENSIONS
of javax.microedition.pki.CertificateException 455
BASELINE
of javax.microedition.lcdui.Graphics 256
BLOCK_END
of javax.microedition.media.control.ToneControl 420
BLOCK_START
of javax.microedition.media.control.ToneControl 420
BOTTOM
of javax.microedition.lcdui.Graphics 256
BROKEN_CHAIN
of javax.microedition.pki.CertificateException 455
BUTTON
of javax.microedition.lcdui.Item 293
C
C4
of javax.microedition.media.control.ToneControl 420
callSerially(Runnable)
of javax.microedition.lcdui.Display 216
CANCEL
of javax.microedition.lcdui.Command 187
cancel()
of java.util.TimerTask 54
of java.util.Timer 49
Canvas
of javax.microedition.lcdui 149
Canvas 149
Canvas()
of javax.microedition.lcdui 158
Certificate 450
Certificate
of javax.microedition.pki 450
CERTIFICATE_CHAIN_TOO_LONG
of javax.microedition.pki.CertificateException 455
CertificateException
of javax.microedition.pki 453
CertificateException 453
CertificateException(Certificate, byte)
of javax.microedition.pki 456
CertificateException(String, Certificate, byte)
of javax.microedition.pki 457
charsWidth(char[], int, int)
of javax.microedition.lcdui.Font 231
charWidth(char)
of javax.microedition.lcdui.Font 231
checkPermission(String)
of javax.microedition.midlet.MIDlet 441
Choice
of javax.microedition.lcdui 165
Choice 165
CHOICE_GROUP_ELEMENT
of javax.microedition.lcdui.Display 214
ChoiceGroup 175
ChoiceGroup
of javax.microedition.lcdui 175
ChoiceGroup(String, int)
of javax.microedition.lcdui 177
ChoiceGroup(String, int, String[], Image[])
of javax.microedition.lcdui 177
Class Summary 464
clipRect(int, int, int, int)
of javax.microedition.lcdui.Graphics 258
close()
of javax.microedition.media.Player 405
CLOSED
of javax.microedition.media.PlayerListener 411
of javax.microedition.media.Player 404
closeRecordStore()
of javax.microedition.rms.RecordStore 478
collidesWith(Image, int, int, boolean)
of javax.microedition.lcdui.game.Sprite 372
collidesWith(Sprite, boolean)
of javax.microedition.lcdui.game.Sprite 372
collidesWith(TiledLayer, boolean)
of javax.microedition.lcdui.game.Sprite 373
COLOR_BACKGROUND
of javax.microedition.lcdui.Display 214
COLOR_BORDER
of javax.microedition.lcdui.Display 214
COLOR_FOREGROUND
of javax.microedition.lcdui.Display 215
COLOR_HIGHLIGHTED_BACKGROUND
of javax.microedition.lcdui.Display 215
COLOR_HIGHLIGHTED_BORDER
of javax.microedition.lcdui.Display 215
COLOR_HIGHLIGHTED_FOREGROUND
of javax.microedition.lcdui.Display 215
Command
of javax.microedition.lcdui 183
Command 183
Command(String, int, int)
of javax.microedition.lcdui 188
Command(String, String, int, int)
of javax.microedition.lcdui 189
commandAction(Command, Displayable)
of javax.microedition.lcdui.CommandListener 190
commandAction(Command, Item)
of javax.microedition.lcdui.ItemCommandListener 301
CommandListener 190
CommandListener
of javax.microedition.lcdui 190
CommConnection
of javax.microedition.io 62
CommConnection 62
compare(byte[], byte[])
of javax.microedition.rms.RecordComparator 467
CONFIRMATION
of javax.microedition.lcdui.AlertType 147
Connector 66
Connector
of javax.microedition.io 66
CONSTRAINT_MASK
of javax.microedition.lcdui.TextField 334
CONTINUOUS_IDLE
of javax.microedition.lcdui.Gauge 245
CONTINUOUS_RUNNING
of javax.microedition.lcdui.Gauge 245
Control
of javax.microedition.media 391
Control 391
Controllable
of javax.microedition.media 392
Controllable 392
copyArea(int, int, int, int, int, int)
of javax.microedition.lcdui.Graphics 258
createAnimatedTile(int)
of javax.microedition.lcdui.game.TiledLayer 382
createImage(byte[], int, int)
of javax.microedition.lcdui.Image 274
createImage(Image)
of javax.microedition.lcdui.Image 275
createImage(Image, int, int, int, int)
of javax.microedition.lcdui.Image 275
createImage(InputStream)
of javax.microedition.lcdui.Image 276
createImage(int, int)
of javax.microedition.lcdui.Image 277
createImage(String)
of javax.microedition.lcdui.Image 277
createPlayer(InputStream, String)
of javax.microedition.media.Manager 396
createPlayer(String)
of javax.microedition.lcdui.Choice 170
createRGBImage(int[], int, int, boolean)
of javax.microedition.lcdui.Image 277
CustomItem 191
CustomItem
of javax.microedition.lcdui 191
CustomItem(String)
of javax.microedition.lcdui 198
D
DATE
of javax.microedition.lcdui.DateField 208
DATE_TIME
of javax.microedition.lcdui.DateField 208
DateField 207
DateField
of javax.microedition.lcdui 207
DateField(String, int)
of javax.microedition.lcdui 209
DateField(String, int, TimeZone)
of javax.microedition.lcdui 209
deallocate()
of javax.microedition.media.Player 405
DECIMAL
of javax.microedition.lcdui.TextField 334
defineCollisionRectangle(int, int, int, int)
of javax.microedition.lcdui.game.Sprite 373
defineReferencePixel(int, int)
of javax.microedition.lcdui.game.Sprite 373
DELAY
of javax.microedition.io.SocketConnection 119
delete(int)
of javax.microedition.lcdui.ChoiceGroup 178
docele(int)
of javax.microedition.lcdui.Form 240
docele(int)
of javax.microedition.lcdui.List 307
docele(int)
of javax.microedition.lcdui.Choice 170
dele(int, int)
of javax.microedition.lcdui.TextBox 323
doceleAll()
of javax.microedition.lcdui.TextField 338
of javax.microedition.lcdui.ChoiceGroup 178
of javax.microedition.lcdui.Form 241
of javax.microedition.lcdui.List 308
deleteRecord(int)
of javax.microedition.rms.RecordStore 478
deleteRecordStore(String)
of javax.microedition.rms.RecordStore 479
destroy()
of javax.microedition.rms.RecordEnumeration 469
destroyApp(boolean)
of javax.microedition.midlet.MIDlet 441
DEVICE_AVAILABLE
of javax.microedition.media.PlayerListener 411
DEVICE_UNAVAILABLE
of javax.microedition.media.PlayerListener 411
DISMISS_COMMAND
of javax.microedition.lcdui.Alert 141
Display 211
Display
of javax.microedition.lcdui 211
Displayable
of javax.microedition.lcdui 223
Displayable 223
DOTTED
of javax.microedition.lcdui.Graphics 256
DOWN
of javax.microedition.lcdui.Canvas 155
DOWN_PRESSED
of javax.microedition.lcdui.game.GameCanvas 351
drawArc(int, int, int, int, int)
of javax.microedition.lcdui.Graphics 259
drawChar(char, int, int, int)
of javax.microedition.lcdui.Graphics 259
drawChars(char[], int, int, int, int, int)
of javax.microedition.lcdui.Graphics 260
drawImage(Image, int, int, int)
of javax.microedition.lcdui.Graphics 260
drawLine(int, int, int, int)
of javax.microedition.lcdui.Graphics 260

drawRect(int, int, int, int)
of javax.microedition.lcdui.Graphics 261
drawRegion(Image, int, int, int, int, int, int, int, int)
of javax.microedition.lcdui.Graphics 261
drawRGB(int[], int, int, int, int, int, int, boolean)
of javax.microedition.lcdui.Graphics 263
drawRoundRect(int, int, int, int, int)
of javax.microedition.lcdui.Graphics 264
drawString(String, int, int, int)
of javax.microedition.lcdui.Graphics 264
drawSubstring(String, int, int, int, int, int)
of javax.microedition.lcdui.Graphics 264
DURATION_UPDATED
of javax.microedition.media.PlayerListener 411

E
EMAILADDR
of javax.microedition.lcdui.TextField 335
END_OF_MEDIA
of javax.microedition.media.PlayerListener 412
enumerateRecords(RecordFilter, RecordComparator, boolean)
of javax.microedition.rms.RecordStore 479
EQUIVALENT
of javax.microedition.rms.RecordComparator 466
ERROR
of javax.microedition.media.PlayerListener 412
of javax.microedition.lcdui.AlertType 147
EXCLUSIVE
of javax.microedition.lcdui.Choice 168
EXIT
of javax.microedition.lcdui.Command 187
EXPIRED
of javax.microedition.pki.CertificateException 455

F
FACE_MONOSPACE
of javax.microedition.lcdui.Font 229
FACE_PROPORTIONAL
of javax.microedition.lcdui.Font 229
FACE_SYSTEM
of javax.microedition.lcdui.Font 229
fillArc(int, int, int, int, int)
of javax.microedition.lcdui.Graphics 265
fillCells(int, int, int, int)
of javax.microedition.lcdui.game.TiledLayer 383
fillRect(int, int, int, int)
of javax.microedition.lcdui.Graphics 265
fillRoundRect(int, int, int, int, int)
of javax.microedition.lcdui.Graphics 266
fillTriangle(int, int, int, int, int, int)
of javax.microedition.lcdui.Graphics 266
FIRE
of javax.microedition.lcdui.Canvas 155
FIRE_PRESSED
of javax.microedition.lcdui.game.GameCanvas 351
flashBacklight(int)
of javax.microedition.lcdui.Display 218
flushGraphics()
of javax.microedition.lcdui.game.GameCanvas 352
flushGraphics(int, int, int)
of javax.microedition.lcdui.game.GameCanvas 352
FOLLOWS
of javax.microedition.rms.RecordComparator 467
Font 227
Font
of javax.microedition.lcdui 227
FONT_INPUT_TEXT
of javax.microedition.lcdui.Font 230
FONT_STATIC_TEXT
of javax.microedition.lcdui.Font 230
FOREVER
of javax.microedition.lcdui.Alert 141
Form 235
Form
of javax.microedition.lcdui.Form 235
Form(String)
of javax.microedition.lcdui 239
Form(String, Item[])
of javax.microedition.lcdui 239
G
GAME_A
of javax.microedition.lcdui.Canvas 155
GAME_A_PRESSED
of javax.microedition.lcdui.game.GameCanvas 351
GAME_B
of javax.microedition.lcdui.Canvas 155
GAME_B_PRESSED
of javax.microedition.lcdui.game.GameCanvas 351
GAME_C
of javax.microedition.lcdui.Canvas 155
GAME_C_PRESSED
of javax.microedition.lcdui.game.GameCanvas 351
GAME_D
of javax.microedition.lcdui.Canvas 155
GAME_D_PRESSED
of javax.microedition.lcdui.game.GameCanvas 351
GameCanvas
of javax.microedition.lcdui.game 348
GameCanvas 348
GameCanvas(boolean)
of javax.microedition.lcdui.game 352
Gauge 243
Gauge
of javax.microedition.lcdui 243
Gauge(String, boolean, int, int)
of javax.microedition.lcdui 246
GET
of javax.microedition.io.HttpConnection 82
get(int)
of javax.microedition.lcdui.Form 241
-548-
getAddress() of javax.microedition.io.SocketConnection
getAltText() of javax.microedition.lcdui.ImageItem
getAnimatedTile(int) of javax.microedition.lcdui.game.TiledLayer
getAppearanceMode() of javax.microedition.lcdui.ImageItem
getAppProperty(String) of javax.microedition.midlet.MIDlet
getBaselinePosition() of javax.microedition.lcdui.Font
getBaudRate() of javax.microedition.io.CommConnection
getBestImageHeight(int) of javax.microedition.lcdui.Display
getBestImageWidth(int) of javax.microedition.lcdui.Display
getBlueComponent() of javax.microedition.lcdui.Graphics
getBorderStyle(boolean) of javax.microedition.lcdui.Display
getCaretPosition() of javax.microedition.lcdui.TextField
getCell(int, int) of javax.microedition.lcdui.game.TiledLayer
getCellHeight() of javax.microedition.lcdui.game.TiledLayer
getCellWidth() of javax.microedition.lcdui.game.TiledLayer
getCertificate() of javax.microedition.pki.CertificateException
getChars(char[]) of javax.microedition.lcdui.TextField
getCipherSuite() of javax.microedition.io.SecurityInfo
getClipHeight() of javax.microedition.lcdui.Graphics
getClipWidth() of javax.microedition.lcdui.Graphics
getClipX() of javax.microedition.lcdui.Graphics
getClipY() of javax.microedition.lcdui.Graphics
getColor() of javax.microedition.lcdui.Graphics
getColor(int) of javax.microedition.lcdui.Graphics
getColumns() of javax.microedition.lcdui.game.TiledLayer
getCommandType() of javax.microedition.lcdui.Command
getConstraints() of javax.microedition.lcdui.TextField
getContentType() of javax.microedition.media.Player
getControl(String) of javax.microedition.media.Controllable
getControls() of javax.microedition.media.Controllable
getCurrent() of javax.microedition.lcdui.Display
getDate() of javax.microedition.io.HttpConnection
getDefaultFont() of javax.microedition.lcdui.Font
getDefaultTimeout() of javax.microedition.lcdui.Alert
getDisplay(MIDlet) of javax.microedition.lcdui.Display
getDisplayColor(int) of javax.microedition.lcdui.Display
of javax.microedition.lcdui.Graphics 267
getDuration()
of javax.microedition.media.Player 406
getExpiration()
of javax.microedition.io.HttpConnection 88
getAddress()
of javax.microedition.lcdui.Font 232
getFile()
of javax.microedition.io.HttpConnection 88
getFilter(String)
of javax.microedition.io.PushRegistry 106
getFitPolicy()
of javax.microedition.lcdui.GameCanvas 353
of javax.microedition.lcdui.GameCanvas 353
g.getFont()
of javax.microedition.lcdui.StringItem 319
of javax.microedition.lcdui.Graphics 268
g.getFont(int)
of javax.microedition.lcdui.ChoiceGroup 179
g.getFont(int, int, int)
of javax.microedition.lcdui.BooleanChoice 171
g.getFont()
of javax.microedition.lcdui.GameCanvas 353
of javax.microedition.lcdui.GameCanvas 353
g.getFrame()
of javax.microedition.lcdui.Sprite 374
g.getFrameSequenceLength()
of javax.microedition.lcdui.GameCanvas 374
g.getGameAction(int)
of javax.microedition.lcdui.Canvas 158
g/Graphics()
of javax.microedition.lcdui.Image 278
g.getGraphics()
of javax.microedition.lcdui.GameCanvas 353
g.getGrayScale()
of javax.microedition.lcdui.Graphics 268
g.getGreenComponent()
of javax.microedition.lcdui.Graphics 268
getHeaderField(int)
of javax.microedition.io.HttpConnection 88
getHeaderField(String)
of javax.microedition.io.HttpConnection 88
getHeaderFieldDate(String, long)
of javax.microedition.io.HttpConnection 89
getHeaderFieldInt(String, int)
of javax.microedition.io.HttpConnection 89
getHeaderFieldKey(int)
of javax.microedition.io.HttpConnection 89
getHost()
of javax.microedition.lcdui.GameCanvas 353
of javax.microedition.lcdui.GameCanvas 353
getImage()
of javax.microedition.lcdui.Alert 142
g.getImage(int)
of javax.microedition.lcdui.GameCanvas 353
g.getInteractionModes()
of javax.microedition.lcdui.GameCanvas 353
g.getKeyCode(int)
of javax.microedition.lcdui.Image 279
g.getKeyName(int)
of javax.microedition.lcdui.GameCanvas 353
g.getKeyStates()
of javax.microedition.lcdui.GameCanvas 353
g.getLabel()
of javax.microedition.lcdui.Item 297
g getLastModified()  
of javax.microedition.rms.RecordStore 480  
of javax.microedition.io.HttpConnection 90
g getLayerAt(int)  
of javax.microedition.lcdui.game.LayerManager 360
g getLayout()  
of javax.microedition.lcdui.Item 297  
of javax.microedition.lcdui.ImageItem 286
g getLevel()  
of javax.microedition.media.control.VolumeControl 424
g getLocalAddress()  
of javax.microedition.io.UDPDatagramConnection 123  
of javax.microedition.io.SocketConnection 120  
of javax.microedition.io.ServerSocketConnection 116
g getLocalPort()  
of javax.microedition.io.ServerSocketConnection 116  
of javax.microedition.io.UDPDatagramConnection 123  
of javax.microedition.io.SocketConnection 120
g getLongLabel()  
of javax.microedition.lcdui.Command 189
g getMaxSize()  
of javax.microedition.lcdui.TextBox 324  
of javax.microedition.lcdui.TextField 339
g getMaxValue()  
of javax.microedition.lcdui.Gauge 247
g getMediaTime()  
of javax.microedition.media.Player 406
g getMIDlet(String)  
of javax.microedition.io.PushRegistry 106
g getMinContentHeight()  
of javax.microedition.lcdui.CustomItem 199
g getMinContentWidth()  
of javax.microedition.lcdui.CustomItem 198
g getMinimumHeight()  
of javax.microedition.lcdui.Item 297
g getMinimumWidth()  
of javax.microedition.lcdui.Item 297
g getName()  
of javax.microedition.rms.RecordStore 480
g getNextRecordID()  
of javax.microedition.rms.RecordStore 480
g getNextRecord()  
of javax.microedition.pki.Certificate 451
g getNotBefore()  
of javax.microedition.pki.Certificate 451
g getNumRecords()  
of javax.microedition.rms.RecordStore 480
g getPort()  
of javax.microedition.io.HttpConnection 90  
of javax.microedition.io.SocketConnection 120  
of javax.microedition.io.HttpsConnection 96
g getPrefContentHeight(int)  
of javax.microedition.lcdui.Item 297
g getPrefContentWidth(int)  
of javax.microedition.lcdui.Item 298
g getPreferredHeight()  
of javax.microedition.lcdui.Item 297
g getPreferredWidth()  
of javax.microedition.lcdui.Item 298
g getPriority()  
of javax.microedition.lcdui.Command 189
g getProtocol()  
of javax.microedition.io.HttpConnection 90
g getProtocolName()  
of javax.microedition.io.SecurityInfo 112
g getProtocolVersion()  
of javax.microedition.io.SecurityInfo 113
g getQuery()  
of javax.microedition.io.HttpConnection 90
g getRawFrameCount()  
of javax.microedition.lcdui.game.Sprite 374
g getReason()  
of javax.microedition.pki.CertificateException 457
getRecord(int)
of javax.microedition.rms.RecordStore 481
getRecord(int, byte[], int)
of javax.microedition.rms.RecordStore 481
getRecordSize(int)
of javax.microedition.rms.RecordStore 481
getRedComponent()
of javax.microedition.lcdui.Graphics 268
getRef()
of javax.microedition.io.HttpConnection 90
getRefPixelX()
of javax.microedition.lcdui.game.Sprite 374
getRefPixelY()
of javax.microedition.lcdui.game.Sprite 375
getRequestMethod()
of javax.microedition.io.HttpConnection 91
getRequestProperty(String)
of javax.microedition.io.HttpConnection 91
getResponseCode()
of javax.microedition.io.HttpConnection 91
getResponseMessage()
of javax.microedition.io.HttpConnection 91
getRGB(int[], int, int, int, int, int, int)
of javax.microedition.lcdui.Image 279
getRows()
of javax.microedition.lcdui.game.TiledLayer 384
getSecurityInfo()
of javax.microedition.io.HttpsConnection 96
of javax.microedition.io.SecureConnection 111
getSelectedFlags(boolean[])
of javax.microedition.lcdui.game.ChoiceGroup 179
of javax.microedition.lcdui.game.Choice 171
of javax.microedition.lcdui.List 308
getSelectedIndex()
of javax.microedition.lcdui.List 309
of javax.microedition.lcdui.game.Choice 172
of javax.microedition.lcdui.game.ChoiceGroup 179
getSerialNumber()
of javax.microedition.pki.Certificate 452
getServerCertificate()
of javax.microedition.io.SecurityInfo 113
getSigAlgName()
of javax.microedition.pki.Certificate 452
getSize()
of javax.microedition.lcdui.game.LayerManager 360
of javax.microedition.lcdui.Font 233
of javax.microedition.rms.RecordStore 482
getSizeAvailable()
of javax.microedition.rms.RecordStore 482
getSocketOption(byte)
of javax.microedition.rms.RecordStore 482
getState()
of javax.microedition.media.Player 406
getString()
of javax.microedition.lcdui.game.TextBox 324
of javax.microedition.lcdui.game.TextField 339
of javax.microedition.lcdui.game.Alert 143
of javax.microedition.lcdui.game.Ticker 344
getString(int)
of javax.microedition.lcdui.game.Choice 172
of javax.microedition.lcdui.game.ChoiceGroup 180
of javax.microedition.lcdui.game.List 309
getStrokeStyle()
of javax.microedition.lcdui.Graphics 268
getStyle()
of javax.microedition.lcdui.Font 233
getSubject()
of javax.microedition.pki.Certificate 452
getSupportedContentTypes(String)
of javax.microedition.media.Manager 397
getSupportedProtocols(String)
of javax.microedition.media.Manager 397
getText()
of javax.microedition.lcdui.StringItem 319
getTicker()
of javax.microedition.lcdui.Displayable 224
getTimeout()
of javax.microedition.lcdui.game.Alert 143
title()
of javax.microedition.lcdui.game.Displayable 225
getTranslateX()
of javax.microedition.lcdui.Graphics 268
getTranslateY()
of javax.microedition.lcdui.Graphics 269
ggetType()
of javax.microedition.pki.Certificate 452
of javax.microedition.lcdui.Alert 143
getURL()
of javax.microedition.io.HttpConnection 92
getValue()
of javax.microedition.lcdui.Gauge 247
getVersion()
of javax.microedition.pki.Certificate 452
of javax.microedition.rms.RecordStore 482
getWidth()
of javax.microedition.lcdui.game.Layer 356
of javax.microedition.lcdui.Displayable 225
of javax.microedition.lcdui.Form 241
of javax.microedition.lcdui.Image 281
of javax.microedition.lcdui.Canvas 159
defX()
of javax.microedition.lcdui.game.Layer 356
defY()
of javax.microedition.lcdui.game.Layer 356

Graphics
of javax.microedition.lcdui 249
Graphics 249

H
hasNextElement()
of javax.microedition.rms.RecordEnumeration 469
hasPointerEvents()
of javax.microedition.lcdui.Canvas 159
hasPointerMotionEvents()
of javax.microedition.lcdui.Canvas 159
hasPreviousElement()
of javax.microedition.rms.RecordEnumeration 469
hasRepeatEvents()
of javax.microedition.lcdui.Canvas 160

HCENTER
of javax.microedition.lcdui.Graphics 257
HEAD
of javax.microedition.io.HttpConnection 82
HELP
of javax.microedition.lcdui.Command 187
hideNotify()
of javax.microedition.lcdui.Canvas 160
of javax.microedition.lcdui.Canvas 159
HTTP_ACCEPTED
of javax.microedition.io.HttpConnection 83
HTTP_BAD_GATEWAY
of javax.microedition.io.HttpConnection 83
HTTP_BAD_METHOD
of javax.microedition.io.HttpConnection 83
HTTP_BAD_REQUEST
of javax.microedition.io.HttpConnection 83
HTTP_CONFLICT
of javax.microedition.io.HttpConnection 83
HTTP_CREATED
of javax.microedition.io.HttpConnection 83
HTTP_ENTITY_TOO_LARGE
of javax.microedition.io.HttpConnection 83
HTTP_EXPECT_FAILED
of javax.microedition.io.HttpConnection 83
HTTP_FORBIDDEN
of javax.microedition.io.HttpConnection 84
HTTP_GATEWAY_TIMEOUT
of javax.microedition.io.HttpConnection 84
HTTP_GONE
of javax.microedition.io.HttpConnection 84
HTTP_LENGTH_REQUIRED
of javax.microedition.io.HttpConnection 84
HTTP_MOVED_PERM
of javax.microedition.io.HttpConnection 84
HTTP_MOVED_TEMP
of javax.microedition.io.HttpConnection 84
INFO
of javax.microedition.lcdui.AlertType 147

INITIAL_CAPS_SENTENCE
of javax.microedition.lcdui.TextField 335

INITIAL_CAPS_WORD
of javax.microedition.lcdui.TextField 335
insert(char[], int, int, int)
of javax.microedition.lcdui.TextBox 324
of javax.microedition.lcdui.TextField 340
insert(int, Item)
of javax.microedition.lcdui.Form 241
insert(int, String, Image)
of javax.microedition.lcdui.ChoiceGroup 180
of javax.microedition.lcdui.Choice 172
of javax.microedition.lcdui.List 309
insert(Layer, int)
of javax.microedition.lcdui.game.LayerManager 360
insert(String, int)
of javax.microedition.lcdui.TextField 340
of javax.microedition.lcdui.TextBox 325
invalidate()
of javax.microedition.lcdui.CustomButton 199
InvalidRecordIDException
of javax.microedition.rms 465
InvalidRecordIDException 465
InvalidRecordIDException()
of javax.microedition.rms 465
InvalidRecordIDException(String)
of javax.microedition.rms 465
isBold()
of javax.microedition.lcdui.Font 233
isColor()
of javax.microedition.lcdui.Display 220
isDoubleBuffered()
of javax.microedition.lcdui.Canvas 160
isInteractive()
of javax.microedition.lcdui.Gauge 248
isItalic()
of javax.microedition.lcdui.Font 233
isKeptUpdated()
of javax.microedition.rms.RecordEnumeration 470
isMutable()
of javax.microedition.lcdui.Image 281
isMuted()
of javax.microedition.media.control.VolumeControl 424
isPlain()
of javax.microedition.lcdui.Font 233
isSelected(int)
of javax.microedition.lcdui.List 309
of javax.microedition.lcdui.ChoiceGroup 180
of javax.microedition.lcdui.Choice 172
isShown()
of javax.microedition.lcdui.Displayable 225
isUnderlined()
of javax.microedition.lcdui.Font 234
isVisible()
of javax.microedition.lcdui.game.Layer 356
Item 288
Item
of javax.microedition.lcdui.Command 187
of javax.microedition.lcdui 288
ItemCommandListener
of javax.microedition.lcdui 301
ItemCommandListener 301
itemStateChanged(Item)
of javax.microedition.lcdui.ItemStateListener 302
ItemStateListener
of javax.microedition.lcdui 302
ItemStateListener 302

K
KEEPALIVE
of javax.microedition.io.SocketConnection 119
keepUpdated(boolean)
of javax.microedition.rms.RecordEnumeration 470
KEY_NUM0
of javax.microedition.lcdui.Canvas 156
KEY_NUM1
of javax.microedition.lcdui.Canvas 156

KEY_NUM2
of javax.microedition.lcdui.Canvas 156

KEY_NUM3
of javax.microedition.lcdui.Canvas 156

KEY_NUM4
of javax.microedition.lcdui.Canvas 156

KEY_NUM5
of javax.microedition.lcdui.Canvas 156

KEY_NUM6
of javax.microedition.lcdui.Canvas 156

KEY_NUM7
of javax.microedition.lcdui.Canvas 157

KEY_NUM8
of javax.microedition.lcdui.Canvas 157

KEY_NUM9
of javax.microedition.lcdui.Canvas 157

KEY_POUND
of javax.microedition.lcdui.Canvas 157

KEY_PRESS
of javax.microedition.lcdui.CustomButton 196

KEY_RELEASE
of javax.microedition.lcdui.CustomButton 196

KEY_REPEAT
of javax.microedition.lcdui.CustomButton 196

KEY_STAR
of javax.microedition.lcdui.Canvas 157

keyPressed(int)
of javax.microedition.lcdui.CustomButton 200
of javax.microedition.lcdui.Canvas 160

keyReleased(int)
of javax.microedition.lcdui.CustomButton 200
of javax.microedition.lcdui.Canvas 160

keyRepeated(int)
of javax.microedition.lcdui.Canvas 161
of javax.microedition.lcdui.CustomButton 200

L
Layer 355
Layer
numRecords()
of javax.microedition.rms.RecordEnumeration 471

OK
OK
of javax.microedition.lcdui.Command 187

open(String)
of javax.microedition.io.Connector 68
open(String, int)
of javax.microedition.io.Connector 68
open(String, int, boolean)
of javax.microedition.io.Connector 68
openDataInputStream(String)
of javax.microedition.io.Connector 69
openDataOutputStream(String)
of javax.microedition.io.Connector 69
openInputStream(String)
of javax.microedition.io.Connector 69
openOutputStream(String)
of javax.microedition.io.Connector 69
openRecordStore(String, boolean)
of javax.microedition.rms.RecordStore 483
openRecordStore(String, boolean, int, boolean)
of javax.microedition.rms.RecordStore 483
openRecordStore(String, String, String)
of javax.microedition.rms.RecordStore 484

P
paint(Graphics)
of javax.microedition.lcdui.Canvas 161
of javax.microedition.lcdui.game.TiledLayer 384
of javax.microedition.lcdui.game.Sprite 375
of javax.microedition.lcdui.game.GameCanvas 354
of javax.microedition.lcdui.game.Layer 357
paint(Graphics, int, int)
of javax.microedition.lcdui.CustomItem 200
of javax.microedition.lcdui.game.LayerManager 361
PASSWORD

of javax.microedition.lcdui.TextField 336

pauseApp()
of javax.microedition.midlet.MIDlet 443

PHONENUMBER
of javax.microedition.lcdui.TextField 337

PLAIN
of javax.microedition.lcdui.TextField 296

platformRequest(String)
of javax.microedition.midlet.MIDlet 443

PLAY_BLOCK
of javax.microedition.media.control.ToneControl 421

Player
of javax.microedition.media 400
Player 400
PlayerListener 410
PlayerListener
of javax.microedition.media 410
playerUpdate(Player, String, Object)
of javax.microedition.media.PlayerListener 413
playSound(Display)
of javax.microedition.lcdui.AlertType 147
playTone(int, int, int)
of javax.microedition.media.Manager 397

POINTER_DRAG
of javax.microedition.lcdui.CustomButton 197

POINTER_PRESS
of javax.microedition.lcdui.CustomButton 197

POINTER_RELEASE
of javax.microedition.lcdui.CustomButton 197

pointerDragged(int, int)
of javax.microedition.lcdui.CustomButton 201
of javax.microedition.lcdui.game.Layer 161

pointerPressed(int, int)
of javax.microedition.lcdui.CustomButton 201
of javax.microedition.lcdui.game.Layer 162

pointerReleased(int, int)
of javax.microedition.lcdui.CustomButton 201
of javax.microedition.lcdui.game.Layer 162

POPUP
of javax.microedition.lcdui.Choice 169
POST
  of javax.microedition.io.HttpConnection 87
PRECEDES
  of javax.microedition.rms.RecordComparator 467
prefetch()
  of javax.microedition.media.Player 407
PREFETCHED
  of javax.microedition.media.Player 404
prevFrame()
  of javax.microedition.lcdui.game.Sprite 375
previousRecord()
  of javax.microedition.rms.RecordEnumeration 471
previousRecordId()
  of javax.microedition.rms.RecordEnumeration 471
PushRegistry
  of javax.microedition.io 98
PushRegistry 98

R
RCVBUF
  of javax.microedition.io.SocketConnection 119
READ
  of javax.microedition.io.Connector 67
READ_WRITE
  of javax.microedition.io.Connector 67
realize()
  of javax.microedition.media.Player 407
REALIZED
  of javax.microedition.media.Player 404
rebuild()
  of javax.microedition.rms.RecordEnumeration 471
recordAdded(RecordStore, int)
  of javax.microedition.rms.RecordListener 473
RecordComparator 466
recordDeleted(RecordStore, int)
  of javax.microedition.rms.RecordListener 473
RecordEnumeration 468
RecordEnumeration
  of javax.microedition.rms 468
RecordFilter 472
RecordFilter
  of javax.microedition.rms 472
RecordListener
  of javax.microedition.rms 473
RecordStore 475
RecordStore
  of javax.microedition.rms 475
RecordStoreException
  of javax.microedition.rms 486
RecordStoreException
  of javax.microedition.rms 486
RecordStoreException()
  of javax.microedition.rms 486
RecordStoreException(String)
  of javax.microedition.rms 486
RecordStoreFullException 488
RecordStoreFullException
  of javax.microedition.rms 488
RecordStoreFullException()
  of javax.microedition.rms 488
RecordStoreFullException(String)
  of javax.microedition.rms 488
RecordStoreNotFoundException
  of javax.microedition.rms 489
RecordStoreNotFoundException
  of javax.microedition.rms 489
RecordStoreNotFoundException()
  of javax.microedition.rms 489
RecordStoreNotFoundException(String)
  of javax.microedition.rms 489
RecordStoreNotOpenException 490
RecordStoreNotOpenException
  of javax.microedition.rms 490
RecordStoreNotOpenException()
  of javax.microedition.rms 490
RecordStoreNotOpenException(String)
set(int, String, Image)
of javax.microedition.lcdui.Form 242
set(int, String, Image)
of javax.microedition.lcdui.List 310
set(int, String, Image)
of javax.microedition.lcdui.Choice 173
set(int, String, Image)
of javax.microedition.lcdui.ChoiceGroup 180

set(int, String, Image)
of javax.microedition.media.control.ToneControl 421

setAltText(String)
of javax.microedition.lcdui.ImageItem 286

setAnimatedTile(int, int)
of javax.microedition.lcdui.game.TiledLayer 385

setBaudRate(int)
of javax.microedition.io.CommConnection 65

setCell(int, int, int)
of javax.microedition.lcdui.game.TiledLayer 385

setChars(char[], int, int)
of javax.microedition.lcdui.TextField 340
setChars(char[], int, int)
of javax.microedition.lcdui.TextBox 325

setClip(int, int, int, int)
of javax.microedition.lcdui.Graphics 269

setColor(int)
of javax.microedition.lcdui.Graphics 269

setColor(int, int, int)
of javax.microedition.lcdui.Graphics 269

setFont(Font)
of javax.microedition.lcdui.StringItem 319

setFont(Font)
of javax.microedition.lcdui.Graphics 269

setFull BCM3eMode(boolean)
of javax.microedition.lcdui.Canvas 163

setGrayScale(int)
of javax.microedition.lcdui.Graphics 270

setImage(Image)
of javax.microedition.lcdui.ImageItem 287
setImage(Image)
of javax.microedition.lcdui.Game 143

setImage(Image, int, int)
of javax.microedition.lcdui.Sprite 376

setFrame(int)
of javax.microedition.lcdui.game.Sprite 375

setFrameSequence(int[]) of javax.microedition.lcdui.game.Sprite 376

setInitial InputMode(String)
of javax.microedition.lcdui.TextBox 341

setInputMode(int)
of javax.microedition.lcdui.TextField 341

setItemCommandListener(ItemCommandListener)
of javax.microedition.lcdui.Item 299

setItemStateListener(ItemStateListener)
of javax.microedition.lcdui.Form 242

setLabel(String)
of javax.microedition.lcdui.Item 300
<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setLayout(int)</code></td>
<td><code>Item</code></td>
<td>javax.microedition.lcdui.Item</td>
</tr>
<tr>
<td><code>setLevel(int)</code></td>
<td><code>ImageItem</code></td>
<td>javax.microedition.lcdui.ImageItem</td>
</tr>
<tr>
<td><code>setLoopCount(int)</code></td>
<td><code>Player</code></td>
<td>javax.microedition.media.media.control.VolumeControl</td>
</tr>
<tr>
<td><code>setMaxSize(int)</code></td>
<td><code>TextBox</code></td>
<td>javax.microedition.lcdui.TextBox</td>
</tr>
<tr>
<td><code>setMaxValue(int)</code></td>
<td><code>TextField</code></td>
<td>javax.microedition.lcdui.TextField</td>
</tr>
<tr>
<td><code>setMaxValue(int)</code></td>
<td><code>Gauge</code></td>
<td>javax.microedition.lcdui.Gauge</td>
</tr>
<tr>
<td><code>setMediaTime(long)</code></td>
<td><code>Player</code></td>
<td>javax.microedition.media.media.control.VolumeControl</td>
</tr>
<tr>
<td><code>setPosition(int, int)</code></td>
<td><code>Layer</code></td>
<td>javax.microedition.lcdui.game.Layer</td>
</tr>
<tr>
<td><code>setPreferredSize(int, int)</code></td>
<td><code>Item</code></td>
<td>javax.microedition.lcdui.Item</td>
</tr>
<tr>
<td><code>setRecord(int, byte[], int, int)</code></td>
<td><code>RecordStore</code></td>
<td>javax.microedition.rms.RecordStore</td>
</tr>
<tr>
<td><code>setRefPixelPosition(int, int)</code></td>
<td><code>Sprite</code></td>
<td>javax.microedition.lcdui.game.Sprite</td>
</tr>
<tr>
<td><code>setRequestMethod(String)</code></td>
<td><code>HttpConnection</code></td>
<td>javax.microedition.io.HttpConnection</td>
</tr>
<tr>
<td><code>setTimeout(int)</code></td>
<td><code>Alert</code></td>
<td>javax.microedition.lcdui.Alert</td>
</tr>
<tr>
<td><code>setTick(Ticker)</code></td>
<td><code>Displayable</code></td>
<td>javax.microedition.lcdui.Displayable</td>
</tr>
<tr>
<td><code>getTitle(String)</code></td>
<td><code>Alert</code></td>
<td>javax.microedition.lcdui.Alert</td>
</tr>
<tr>
<td><code>getViewWindow(int, int, int, int)</code></td>
<td><code>LayerManager</code></td>
<td>javax.microedition.lcdui.game.LayerManager</td>
</tr>
<tr>
<td><code>setVisible(boolean)</code></td>
<td><code>Layer</code></td>
<td>javax.microedition.lcdui.game.Layer</td>
</tr>
<tr>
<td><code>showNotify()</code></td>
<td><code>Canvas</code></td>
<td>javax.microedition.lcdui.Canvas</td>
</tr>
<tr>
<td><code>setSelectedIndex(int, boolean)</code></td>
<td><code>ChoiceGroup</code></td>
<td>javax.microedition.lcdui.ChoiceGroup</td>
</tr>
<tr>
<td><code>setSequence(byte[])</code></td>
<td><code>ToneControl</code></td>
<td>javax.microedition.media.media.control.ToneControl</td>
</tr>
<tr>
<td><code>setSocketOption(byte, int)</code></td>
<td><code>SocketConnection</code></td>
<td>javax.microedition.io.SocketConnection</td>
</tr>
<tr>
<td><code>setStaticTileSet(Image, int, int)</code></td>
<td><code>TiledLayer</code></td>
<td>javax.microedition.lcdui.game.TiledLayer</td>
</tr>
<tr>
<td><code>setString(String)</code></td>
<td><code>TextBox</code></td>
<td>javax.microedition.lcdui.TextBox</td>
</tr>
<tr>
<td><code>setText(String)</code></td>
<td><code>StringItem</code></td>
<td>javax.microedition.lcdui.StringItem</td>
</tr>
<tr>
<td><code>setTick(Ticker)</code></td>
<td><code>Displayable</code></td>
<td>javax.microedition.lcdui.Displayable</td>
</tr>
<tr>
<td><code>setTick(Ticker)</code></td>
<td><code>Displayable</code></td>
<td>javax.microedition.lcdui.Displayable</td>
</tr>
<tr>
<td><code>setType(AlertType)</code></td>
<td><code>Alert</code></td>
<td>javax.microedition.lcdui.Alert</td>
</tr>
<tr>
<td><code>setValue(int)</code></td>
<td><code>Gauge</code></td>
<td>javax.microedition.lcdui.Gauge</td>
</tr>
<tr>
<td><code>getViewWindow(int, int, int, int)</code></td>
<td><code>LayerManager</code></td>
<td>javax.microedition.lcdui.game.LayerManager</td>
</tr>
<tr>
<td><code>setVisible(boolean)</code></td>
<td><code>Canvas</code></td>
<td>javax.microedition.lcdui.Canvas</td>
</tr>
<tr>
<td><code>showNotify()</code></td>
<td><code>Canvas</code></td>
<td>javax.microedition.lcdui.Canvas</td>
</tr>
</tbody>
</table>
SILENCE
  of javax.microedition.media.control.ToneControl 421
SITENAME_MISMATCH
  of javax.microedition.pki.CertificateException 456
size()
  of javax.microedition.lcdui.Choice 174
  of javax.microedition.lcdui.ChoiceGroup 182
  of javax.microedition.lcdui.List 312
  of javax.microedition.lcdui.Form 242
  of javax.microedition.lcdui.TextField 342
  of javax.microedition.lcdui.TextBox 326
SIZE_LARGE
  of javax.microedition.lcdui.Font 230
SIZE_MEDIUM
  of javax.microedition.lcdui.Font 230
SIZE_SMALL
  of javax.microedition.lcdui.Font 230
sizeChanged(int, int)
  of javax.microedition.lcdui.Displayable 226
  of javax.microedition.lcdui.Canvas 164
  of javax.microedition.lcdui.CustomButton 202
SNDBUF
  of javax.microedition.io.SocketConnection 120
SocketConnection
  of javax.microedition.io 117
SocketConnection 117
SOLID
  of javax.microedition.lcdui.Graphics 257
Spacer
  of javax.microedition.lcdui 314
Spacer 314
Spacer(int, int)
  of javax.microedition.lcdui 315
Sprite 363
Sprite
  of javax.microedition.lcdui.game 363
Sprite(Image)
  of javax.microedition.lcdui.game 370
Sprite(Image, int, int)
  of javax.microedition.lcdui.game 370
of javax.microedition.lcdui.game 371
Sprite(Sprite)
  of javax.microedition.lcdui.game 371
start()
  of javax.microedition.media.Player 408
startApp()
  of javax.microedition.midlet.MIDlet 445
STARTED
  of javax.microedition.media.PlayerListener 412
  of javax.microedition.media.Player 405
STOP
  of javax.microedition.lcdui.Command 188
stop()
  of javax.microedition.media.Player 409
STOPPED
  of javax.microedition.media.PlayerListener 412
StringItem 317
StringItem
  of javax.microedition.lcdui 317
StringItem(String, String)
  of javax.microedition.lcdui 318
StringItem(String, String, int)
  of javax.microedition.lcdui 318
stringWidth(String)
  of javax.microedition.lcdui.Font 234
STYLE_BOLD
  of javax.microedition.lcdui.Font 230
STYLE_ITALIC
  of javax.microedition.lcdui.Font 231
STYLE_PLAIN
  of javax.microedition.lcdui.Font 231
STYLE_UNDERLINED
  of javax.microedition.lcdui.Font 231
substringWidth(String, int, int)
  of javax.microedition.lcdui.Font 234
TEXT_WRAP_DEFAULT
  of javax.microedition.media.control.ToneControl 421
ToneControl
  of javax.microedition.media.control 416

ToneControl
  of javax.microedition.lcdui.Graphics 257

TRANS_MIRROR
  of javax.microedition.lcdui.game.Sprite 369

TRANS_MIRROR_ROT180
  of javax.microedition.lcdui.game.Sprite 370

TRANS_MIRROR_ROT270
  of javax.microedition.lcdui.game.Sprite 370

TRANS_MIRROR_ROT90
  of javax.microedition.lcdui.game.Sprite 370

TRANS_NONE
  of javax.microedition.lcdui.game.Sprite 370

TRANS_ROT180
  of javax.microedition.lcdui.game.Sprite 370

TRANS_ROT270
  of javax.microedition.lcdui.game.Sprite 370

TRANS_ROT90
  of javax.microedition.lcdui.game.Sprite 370

translate(int, int)
  of javax.microedition.lcdui.Graphics 270

traverse(int, int, int, int[])
  of javax.microedition.lcdui.CustomItem 203

TRAVERSE_HORIZONTAL
  of javax.microedition.lcdui.CustomItem 197

TRAVERSE_VERTICAL
  of javax.microedition.lcdui.CustomItem 197

traverseOut()
  of javax.microedition.lcdui.CustomItem 206
UNREALIZED
of javax.microedition.media.Player 405

UNRECOGNIZED_ISSUER
of javax.microedition.pki.CertificateException 456

unregisterConnection(String)
of javax.microedition.io.PushRegistry 107

UNSUPPORTED_PUBLIC_KEY_TYPE
of javax.microedition.pki.CertificateException 456

UNSUPPORTED_SIGALG
of javax.microedition.pki.CertificateException 456

UP
of javax.microedition.lcdui.Canvas 158

UP_PRESSED
of javax.microedition.lcdui.game.GameCanvas 352

URL
of javax.microedition.lcdui.TextField 338

V
VCENTER
of javax.microedition.lcdui.Graphics 257

VERIFICATION_FAILED
of javax.microedition.pki.CertificateException 456

VERSION
of javax.microedition.media.control.ToneControl 422

dvibrate(int)
of javax.microedition.lcdui.Display 222

VOLUME_CHANGED
of javax.microedition.media.PlayerListener 413

VolumeControl
of javax.microedition.media.control 423

VolumeControl 423

W
WARNING
of javax.microedition.lcdui.AlertType 147

WRITE