RESEARCH AND EVALUATION LICENSE

MOTOROLA INC. IS WILLING TO LICENSE THIS SPECIFICATION TO YOU ONLY UNDER THE TERMS CONTAINED IN THIS LICENSE (“LICENSE”). PLEASE READ THE TERMS AND CONDITIONS OF THIS LICENSE CAREFULLY. BY ACCESSING OR USING THE SPECIFICATION YOU WILL BE BOUND BY THE TERMS OF THIS LICENSE.

JSR 271 Mobile Information Device Profile Specification ("Specification")

Version: 3.0

Status: Final Release

Specification Lead: Motorola Inc. ("Specification Lead")

Release: November 3, 2009

Copyright 2007-2009 Motorola Inc. All Rights Reserved.

1. NOTICE; LIMITED LICENSE GRANTS

1.1 The Specification Lead hereby grants You a non-exclusive, non-transferable, non-assignable, worldwide, royalty-free, fully paid-up, limited license (without the right to sublicense) solely under intellectual property rights licensable by the Specification Lead to analyze and to use the Specification for research, evaluation, optimization and development purposes. In addition You may make a reasonable number of verbatim copies of this Specification in its entirety for Your private or internal use, as applicable, in accordance with the terms and conditions of this License.

1.2 No rights are granted under this License for internal deployment, the creation and/or distribution of implementations of the Specification for direct or indirect (including strategic) gain or advantage, the modification of the Specification (other than to the extent of Your fair use rights) or the distribution of the Specification or making the Specification available for 3rd parties.

1.3 Except as expressly set forth in this license, You acquire no right, title or interest in or to Specification or any other intellectual property licensable by the Specification Lead and no other rights are granted by implication, estoppel or otherwise. The Specification may only be used in accordance with the license terms set forth herein. This License will terminate immediately without notice from Specification Lead if You fail to comply with any provision of this License.

2. TRADEMARKS

2.1 Motorola® is a registered trademark of Motorola, Inc. Motorola, Inc's product names are either trademarks or registered trademarks of Motorola, Inc. Your access to this Specification should not be construed as granting, by implication, estoppel or otherwise, any license or right to use any marks appearing in the Specification without the prior written consent of Motorola Inc. or their respective licensors, as applicable. No right, title, or interest in or to any trademarks, service marks, or trade names of any third parties, is granted hereunder.

2.2 You shall not be allowed to remove any of the copyright statements or disclaimers or other proprietary notices contained in the Specification and You are obliged to include the copyright statement and the disclaimers, if any, in any copies of the Specification You make.
3. DISCLAIMER OF WARRANTIES

3.1 SUBJECT TO ANY STATUTORY WARRANTIES OR CONDITIONS WHICH CAN NOT BE EXCLUDED, THE SPECIFICATION IS PROVIDED #AS IS# WITHOUT WARRANTY OR CONDITION OF ANY KIND EITHER EXPRESS, IMPLIED, OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, AND STATUTORY ARE HEREBY DISCLAIMED. THE ENTIRE RISK ARISING OUT OF OR RELATING TO THE USE OR PERFORMANCE OF THE SPECIFICATION REMAINS WITH YOU.

3.2 THE SPECIFICATION MAY 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. SPECIFICATION LEAD 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.

4. LIMITATION OF LIABILITY

4.1 TO THE FULLEST EXTENT PERMITTED BY LAW, IN NO EVENT WILL THE SPECIFICATION LEAD OR ITS SUPPLIERS BE LIABLE FOR ANY LOST PROFITS, LOST SAVINGS, LOST REVENUE, LOST DATA, PROCUREMENT OF SUBSTITUTE GOODS, OR FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, EVEN IF THE SPECIFICATION LEAD OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSSES OR DAMAGES. IN ADDITION THE SPECIFICATION LEAD AND ITS SUPPLIERS WILL NOT BE LIABLE FOR ANY DAMAGES CLAIMED BY YOU BASED ON ANY THIRD PARTY CLAIM.

4.2 Some jurisdictions do not allow the exclusion of implied warranties, or the limitation for consequential damages, so Section 4.1 may not apply to You in whole, but in such case Section 4.1 will apply to You to the maximum extent permitted by applicable law.

5. EXPORT CONTROL

5.1 You shall follow all export control laws and regulations relating to Specification.

6. RESTRICTED RIGHTS LEGEND

6.1 Note to U.S. Government Users. The Specification is a "Commercial Items", as that term is defined at 48 C.F.R. 2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation", as such terms are used in 48 C.F.R. 12.212 or 48 C.F.R. 227.7202, as applicable. Consistent with 48 C.F.R. 12.212 or 48 C.F.R. 227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software Documentation are being licensed to U.S. Government end users a) only as Commercial Items and b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States.
## Contents

1 Mobile Information Device Profile, v3.0 (JSR 271) .................. 1
  1.1 Preface ........................................................................... 1
  1.2 Revision History ............................................................. 1
  1.3 Who Should Use This Specification ................................. 2
  1.4 How This Specification Is Organized ............................... 2
  1.5 Report and Contact ....................................................... 3
  1.6 Document Conventions .................................................. 3
  1.7 Glossary ....................................................................... 3
  1.8 Contributors ................................................................... 6
  1.9 Introduction ................................................................. 7
  1.10 New Features ............................................................... 8
  1.11 Scope ........................................................................... 8
  1.12 Architecture .................................................................. 9
  1.13 Device Requirements .................................................. 10
    1.13.1 Hardware ............................................................... 10
    1.13.2 Software ............................................................... 10
  1.14 Specification Requirements ....................................... 11
  1.15 References ................................................................... 12

2 MIDP 3.0 Packaging ......................................................... 17
  2.1 Overview and Goals ....................................................... 17
  2.2 Functional Requirements ............................................ 17
  2.3 MIDlet Suites and LIBlets ............................................. 17
  2.4 Packaging ................................................................. 18
    2.4.1 The JAR Manifest .................................................. 18
    2.4.2 The Application Descriptor .................................... 20
    2.4.3 MIDlet Suite Packaging ........................................ 23
    2.4.4 LIBlet Packaging .................................................. 23
  2.5 Application Attributes .................................................. 24
    2.5.1 Identification Attributes ........................................ 25
    2.5.2 Provisioning Attributes ......................................... 27
    2.5.3 Operational Attributes ......................................... 30
    2.5.4 Handling Localized Attribute Versions ................. 32
6.5 Combined usage of MIDP 2.0 and MIDP 3.0 security models .................... 75

7 MIDlet Suites Trust Model Using X.509 PKI ................................. 77

7.1 Introduction .............................................................................. 77
7.1.1 Transport & Security Standards ........................................... 77
7.1.2 Definition of Terms .............................................................. 77
7.2 Signing a MIDlet Suite ............................................................. 78
7.2.1 Creating the Signing Certificate .......................................... 78
7.2.2 Inserting Certificates into the Application Descriptor ............ 78
7.2.3 Creating the RSA SHA-1 signature of the JAR ................. 79
7.3 Authenticating a MIDlet Suite .................................................. 79
7.3.1 Verifying Signing Certificate .............................................. 79
7.3.2 Verifying the MIDlet Suite JAR .......................................... 80
7.3.3 Summary of MIDlet Suite Source Verification Results .......... 80
7.3.4 Caching of Authentication and Authorization Results ........ 81
7.4 Security of Alternative Application Representation Formats .......... 81
7.5 MIDP X.509 Certificate Profile for Trusted MIDlet Suites ........... 81
7.5.1 Certificate Processing for Provisioning ................................... 81
7.5.2 Certificate Expiration and Revocation .................................. 82
7.6 Scenarios of MIDlet Suite Signing ............................................ 82
7.6.1 Scenario 1a ........................................................................ 82
7.6.2 Scenario 1b ........................................................................ 82
7.6.3 Scenario 2 .......................................................................... 82
7.6.4 Scenario 3 .......................................................................... 83

8 MIDP 3.0 Security Policy .............................................................. 84
8.1 Introduction .............................................................................. 84
8.2 Protection Domains ................................................................. 84
8.2.1 Manufacturer and Operator Protection Domains .................. 85
8.2.1.1 Manufacturer Protection Domain ...................................... 85
8.2.1.2 Operator Protection Domain ............................................ 86
8.2.2 Third Party Protection Domains ........................................... 89
8.2.2.1 Identified Third Party Protection Domain ....................... 89
8.2.2.2 Unidentified Third Party Protection Domain ................... 90
10.2.5 microedition.platform ..................................................109
10.2.6 microedition.comports ..................................................110
10.2.7 microedition.hostname ..................................................110
10.3 Application Resource Files ..............................................110
11 javax.microedition.io .......................................................112
11.1 HTTP Networking ..........................................................112
  11.1.1 HTTP Network Connection ...........................................112
  11.1.2 HTTP Request Headers ..............................................114
  11.1.3 User-Agent and Accept-Language Request Headers .............114
  11.1.4 System Properties Used for User-Agent and Accept-Language Request Headers 114
    11.1.4.1 HTTP Request Header Example ..................................114
  11.2 StreamConnection Behavior ..........................................114
11.3 Secure Networking .......................................................115
11.4 Low Level IP Networking ...............................................115
11.5 Push Applications ........................................................116
11.6 Serial Port Communications ............................................116
11.7 IP Version Support .......................................................116
11.8 Inter-MIDlet Communications (IMC) ....................................116
11.9 Security of Networking Functions .....................................117
11.10 Security of PushRegistry .................................................117
Classes .................................................................119
  CommConnection ..........................................................120
  HttpConnection ............................................................124
  HttpsConnection ...........................................................147
  IMCConnection .............................................................151
  IMCServerConnection ......................................................154
  PushRegistry ...............................................................157
  PushRegistryPermission ....................................................169
  SecureConnection ..........................................................172
  SecurityInfo ..............................................................175
  ServerSocketConnection ..................................................177
  SocketConnection ..........................................................180
12  javax.microedition.lcdui  .........................................................187

12.1 User Interface .................................................................187
12.2 Structure of the MIDP UI API ...........................................187
  12.2.1 Class Hierarchy .........................................................188
  12.2.2 Class Overview ..........................................................189
  12.2.3 Interplay with Application Manager ..............................189
12.3 Event Handling ...............................................................190
  12.3.1 Abstract Commands .....................................................191
  12.3.2 Device-Provided Operations ........................................192
  12.3.3 Jog Dial Interaction .....................................................193
  12.3.4 High-Level API for Events ..........................................194
  12.3.5 Low-Level API for Events ..........................................194
  12.3.6 Interplay of High-Level Commands and the Low-Level API ....196
12.4 Graphics and Text in Low-Level API ...................................196
  12.4.1 The Redrawing Scheme .................................................196
  12.4.2 Drawing Model ..........................................................197
  12.4.3 Coordinate System ......................................................198
  12.4.4 Font Support .............................................................198
    12.4.4.1 Downloadable Fonts ...............................................198
12.5 Threading Model ............................................................199
12.6 Text Truncation in UI Components .....................................200
12.7 Activation of Idle Screen MIDlets .....................................200
12.8 Implementation Notes ......................................................201

Classes .................................................................202
  Alert ...............................................................206
  AlertType ...........................................................215
  AnimatedImage ..........................................................218
  Canvas .............................................................223
  CanvasItem ..........................................................246
  Choice ...........................................................250
  ChoiceGroup .........................................................262
<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>271</td>
</tr>
<tr>
<td>CommandLayoutPolicy</td>
<td>282</td>
</tr>
<tr>
<td>CommandListener</td>
<td>285</td>
</tr>
<tr>
<td>CustomItem</td>
<td>286</td>
</tr>
<tr>
<td>DateField</td>
<td>304</td>
</tr>
<tr>
<td>Display</td>
<td>308</td>
</tr>
<tr>
<td>Displayable</td>
<td>344</td>
</tr>
<tr>
<td>DisplayCapabilityException</td>
<td>352</td>
</tr>
<tr>
<td>DisplayListener</td>
<td>354</td>
</tr>
<tr>
<td>FileSelector</td>
<td>356</td>
</tr>
<tr>
<td>Font</td>
<td>367</td>
</tr>
<tr>
<td>FontFormatException</td>
<td>386</td>
</tr>
<tr>
<td>Form</td>
<td>388</td>
</tr>
<tr>
<td>FormLayoutPolicy</td>
<td>398</td>
</tr>
<tr>
<td>Gauge</td>
<td>405</td>
</tr>
<tr>
<td>Graphics</td>
<td>413</td>
</tr>
<tr>
<td>IdleItem</td>
<td>445</td>
</tr>
<tr>
<td>Image</td>
<td>450</td>
</tr>
<tr>
<td>ImageItem</td>
<td>468</td>
</tr>
<tr>
<td>Item</td>
<td>474</td>
</tr>
<tr>
<td>ItemCommandListener</td>
<td>489</td>
</tr>
<tr>
<td>ItemLayoutHint</td>
<td>490</td>
</tr>
<tr>
<td>ItemStateListener</td>
<td>491</td>
</tr>
<tr>
<td>ItemTraversalListener</td>
<td>492</td>
</tr>
<tr>
<td>KeyListener</td>
<td>494</td>
</tr>
<tr>
<td>List</td>
<td>497</td>
</tr>
<tr>
<td>Menu</td>
<td>509</td>
</tr>
<tr>
<td>Notification</td>
<td>519</td>
</tr>
<tr>
<td>NotificationException</td>
<td>526</td>
</tr>
<tr>
<td>NotificationListener</td>
<td>527</td>
</tr>
<tr>
<td>NotificationType</td>
<td>529</td>
</tr>
<tr>
<td>ScalableImage</td>
<td>532</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>15.5.1</td>
<td>Screen Saver Identification ........................................</td>
</tr>
<tr>
<td>15.5.2</td>
<td>Screen Saver Provisioning ...........................................</td>
</tr>
<tr>
<td>15.5.3</td>
<td>Screen Saver MIDlet Execution ......................................</td>
</tr>
<tr>
<td>15.5.4</td>
<td>Screen Saver Deactivation ...........................................</td>
</tr>
<tr>
<td>15.5.5</td>
<td>Screen Saver Security ..............................................</td>
</tr>
<tr>
<td>15.6</td>
<td>Idle Screen MIDlets ..................................................</td>
</tr>
<tr>
<td>15.7</td>
<td>Auto Start MIDlets ..................................................</td>
</tr>
<tr>
<td>15.8</td>
<td>Controlling User actions on a MIDlet Suite .......................</td>
</tr>
<tr>
<td>15.8.1</td>
<td>Persistent MIDlet Suites ...........................................</td>
</tr>
<tr>
<td>15.9</td>
<td>Version Numbering ...................................................</td>
</tr>
<tr>
<td>15.10</td>
<td>MIDlet Classes ......................................................</td>
</tr>
<tr>
<td>15.11</td>
<td>Application Lifecycle ................................................</td>
</tr>
<tr>
<td>15.12</td>
<td>MIDlet Lifecycle .....................................................</td>
</tr>
<tr>
<td>15.12.1</td>
<td>MIDlet Lifecycle Definitions ........................................</td>
</tr>
<tr>
<td>15.13</td>
<td>MIDlet States ..........................................................</td>
</tr>
<tr>
<td>15.14</td>
<td>MIDlet Lifecycle Model ...............................................</td>
</tr>
<tr>
<td>15.15</td>
<td>Application Implementation Notes ...................................</td>
</tr>
<tr>
<td>15.16</td>
<td>Example MIDlet Application .........................................</td>
</tr>
<tr>
<td>15.17</td>
<td>Splash Screens ........................................................</td>
</tr>
<tr>
<td>Classes</td>
<td>................................................................ .............</td>
</tr>
<tr>
<td></td>
<td>ActionsDeniedPermission ...............................................</td>
</tr>
<tr>
<td></td>
<td>AutoStartPermission ....................................................</td>
</tr>
<tr>
<td></td>
<td>MIDlet .................................................................</td>
</tr>
<tr>
<td></td>
<td>MIDletIdentity .........................................................</td>
</tr>
<tr>
<td></td>
<td>MIDletStateException ....................................................</td>
</tr>
<tr>
<td>16</td>
<td>javax.microedition.pki ................................................</td>
</tr>
<tr>
<td>16.1</td>
<td>MIDP X.509 Certificate Profile .......................................</td>
</tr>
<tr>
<td>16.1.1</td>
<td>Certificate Extensions ...............................................</td>
</tr>
<tr>
<td>16.1.2</td>
<td>Certificate Size .......................................................</td>
</tr>
<tr>
<td>16.1.3</td>
<td>Algorithm Support ......................................................</td>
</tr>
<tr>
<td>16.1.4</td>
<td>Certificate Processing for HTTPS ...................................</td>
</tr>
<tr>
<td>Classes</td>
<td>................................................................ .............</td>
</tr>
</tbody>
</table>
List of Tables

Table 1-1 : MIDP 3.0 Glossary ......................................................... .3
Table 2-a : MIDlet Identification Attributes .............................................. .25
Table 2-b : Identification Attributes for LIBlets ............................................ .26
Table 2-c : Provisioning Attributes for MIDlet Suites and MIDlets .................. .27
Table 2-d : Provisioning Attributes for LIBlets ............................................ .29
Table 2-e : Operational Attributes for MIDlet Suites and MIDlets .................. .30
Table 2-f : Operational Attributes for LIBlets ............................................ .32
Table 2-3 : Dependency Names For Standard APIs ......................................... .37
Table 2-4 : Informational Attributes for LIBlets ............................................ .38
Table 3-2 : Install Status Codes and Messages ............................................. .50
Table 3-4 : MIDP RDF Elements ........................................................... .52
Table 6-1 : Definition Of Security Terms .................................................... .63
Table 6-2 : Determining Application Credentials .......................................... .72
Table 6-3 : Application Attributes Required for Application Level Access Authorization ........ .72
Table 7-1 : Definition Of PKI Security Terms .............................................. .77
Table 7-2 : Actions Upon Completion of Signing Certificate Verification ................ .79
Table 7-3 : Summary of MIDlet Suite Source Verification ................................ .80
Table 8-1 : Packages Allowed Without Confirmation to Unidentified Third Party Protection Domain .91
Table 8-2 : Packages Allowed With Confirmation to Unidentified Third Party Protection Domain .91
Table 8-3 : Function Groups & User Settings for Third Party Protection Domains ........ .93
Table 8-4 : Mapping Permissions to Function Groups ..................................... .95
Table 8-5 : Permissions Not Mapped To Function Groups ................................ .96
Table 9-1 : Named Permission to Class Permission Mapping ............................. .103
Table 9-2 : Application Attribute Compatibility .......................................... .105
Table 10-1 : MIDP 3.0 System Properties ................................................... .107
Table 11-1 : System Properties Used for User-Agent and Accept-Language Request Headers .114
Table 15-1 : MIDlet States ................................................................. .668
Table 15-2 : MIDlet Lifecycle Sequence .................................................... .670
Table 18-1 : RMS Interchange File Format .................................................. .706
Table A-1 : Application Descriptor Attributes ............................................. .794
List of Figures

Figure 2-1 : Navigation MIDlet Dependencies ...........................................35
Figure 3-1 : LIBlet Provisioning Overview ..................................................41
Figure 5-1 : Data Isolation Between Concurrent MIDlets .................................60
Figure 5-2 : Concurrent MIDlet Class and Data Separation ...............................60
Figure 6-1 : Verification Steps for Granting & Denying Application Level Access ..........74
Figure 8-1 : Assigning Root Certificates to Operator & Identified Third Party Protection Domains ...88
Figure 8-2 : Recommended Security Prompt Icons ...........................................90
Figure 11-1 : HTTP Network Connection Over Various Stacks ..........................112
Figure 15-2 : Idle Screen MIDlet Example ......................................................665
Figure 15-1 : MIDlet Lifecycle State Diagram ...............................................669
Figure C-1 : Application Level Access Authorization Example 1 .......................800
Figure C-2 : Application Level Access Authorization Example 2 .......................802
Figure C-3 : Application Level Access Authorization Example 3 .......................803
Figure C-4 : Application Level Access Authorization Example 4 .......................804
1.1 Preface

This document defines the Proposed Final Draft of the Mobile Information Device Profile (MIDP) v3.0 for the Java Platform, Micro Edition (Java ME™). This version of the specification represents the version of the specification that will be used as the basis for the RI and TCK. The JSR 271 Expert Group will use feedback gathered during review of the Proposed Final Draft of the specification to revise and refine this specification.

This document and all associated documents are subject to the terms of the JCP and associated agreements (JSPA).

A profile of Java ME 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.

1.2 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>27-September-2002</td>
<td>MIDP 2.0, Final Specification</td>
<td>Final MIDP 2.0 specification</td>
</tr>
<tr>
<td>05-June-2006</td>
<td>MIDP 2.1, Maintenance Release Specification</td>
<td>Final MIDP 2.1 specification - includes clarifications developed by JSR 248</td>
</tr>
<tr>
<td>22-December-2006</td>
<td>MIDP 3.0, Early Draft Review</td>
<td>Incorporated initial MIDP3 components, including Platform, LiBlets, Concurrency, and LCDUI changes</td>
</tr>
<tr>
<td>21-November-2007</td>
<td>MIDP 3.0, Public Review Draft</td>
<td>Incorporated new features and further refinements:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• System and application events</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• refined dependency expression and LiBlets</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• added scalable icon attributes and attributes for localizable display names</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• RMS provisioning and interchange format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MIDlet Categories (Screen Savers, Idle Screen MIDlets, Auto Start MIDlets)</td>
</tr>
</tbody>
</table>
### JSR 271 Overview

- separate chapters for security framework, X.509 PKI trust model, and security policy
- multiply signed MIDlet suites
- application level access authorization
- LCDUI updates for multiple displays, new widgets (TabbedPane), TextInput/TextInputListener
- LCDUI updates for NotificationManager and Menu classes
- clarified layout policy behaviors
- generalized allowing the merging the permissions of similar function groups in security policy

| 14-May-2009 | MIDP 3.0, Proposed Final Draft | Incorporated new features and further refinements:
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Added CommandLayoutPolicy, SecureRecordStoreException, RecordStoreInfo, NotificationType, ScalableImage, and CanvasItem classes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Replaced TextInput class with TextEditor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Renamed UserAllowedPermission class to ActionsDeniedPermission</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Removed NotificationManager Class</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added additional capabilities to Display</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified handling of APPLICATION_RELAUNCH events</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified components required to uniquely identify a LIBlet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified matching of standard API dependency</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified behavior and access to LIBlet record store and resources</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Created separate attributes for LIBlets and MIDlets</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified attribute handling for legacy MIDP 1.0/2.x MIDlet suites</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reformatted attribute table in Appendix A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added information on platform default encoding</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Standardized Displayable.getWidth and Displayable.getHeight definition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added new key codes in Canvas class</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added KeyListener for Canvas and CustomItem</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified FileSelector modes (LOAD, SAVE, DIRECTORY_SELECT, DIRECTORY_CREATE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified Section 8.3.1 and Table 8-3 for JSR 249 alignment</td>
</tr>
</tbody>
</table>

### 1.3 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 MIDP 3.0
- MIDlet application developers targeting MIDP 3.0
- Network operators deploying infrastructure to support MIDP 3.0 devices

### 1.4 How This Specification Is Organized

The JSR 271 Specification includes the following important components:

- MIDP Packaging
- MIDP Provisioning
- MIDP Platform
- MIDP Concurrency
- MIDP Security Framework
1.5 Report and Contact

Your comments on this specification are welcome and appreciated. However, given the phase of this specification, please restrict your comments to issues that would hinder the development of the RI and TCK.

The preferred forum for comments will be the JSR 271 project on opensource.motorola.com. Commenters can request an account, and can post comments and join in the JSR 271 Public Comments discussion forum. Commenters will need an opensource.motorola.com account in order to post comments and participate in the Public Comments discussion forum. The process to request an account and get added to the JSR 271 project on opensource.motorola.com project are as follows:

- Go to opensource.motorola.com and select 'Create an Account' in the New Users box along the left side of the page. On the page that follows, click on 'create account' again.
- For your user name, choose something of the form firstname_lastname (no handles, please) so that others can easily identify you.
- For the 'reason for access' dropdown, select 'member/collaborator for JSR/MIDP work'.
- In the explanation field, you must indicate your intention to provide JSR 271 Public comments so that the request gets routed and handled promptly.
- Finally, in the 'Project' field be sure to select the JSR 271 project to request that you be added to that project.
- IMPORTANT: Once your account is created you will receive a verification link that you need to follow to finalize the process.

Alternatively, comments may be sent to jsr-271-comments@jcp.org; comments received via this mailing list will be cross posted to the discussion forum on opensource.motorola.com.

1.6 Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Code examples are formatted as follows:

```java
List list = new List("Email", List.IMPLICIT, headers);
readCommand = new Command("Read", Command.ITEM, 1);
```

1.7 Glossary

<table>
<thead>
<tr>
<th>Table 1-1 : MIDP 3.0 Glossary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Attributes</td>
</tr>
<tr>
<td>Application Level Access Authorization</td>
</tr>
<tr>
<td><strong>Application Management Software (AMS)</strong></td>
</tr>
<tr>
<td><strong>Auto Start MIDlet</strong></td>
</tr>
<tr>
<td><strong>Auxiliary Display</strong></td>
</tr>
<tr>
<td><strong>Built-In Display</strong></td>
</tr>
<tr>
<td><strong>Certificate (Public Key Certificate)</strong></td>
</tr>
<tr>
<td><strong>Certificate Authority</strong></td>
</tr>
<tr>
<td><strong>MIDP 3.0 MIDlet Concurrency</strong></td>
</tr>
<tr>
<td><strong>Dependency Declaration Chain</strong></td>
</tr>
<tr>
<td><strong>Dependency Expression</strong></td>
</tr>
<tr>
<td><strong>Discovery Application</strong></td>
</tr>
<tr>
<td><strong>Display Capabilities</strong></td>
</tr>
<tr>
<td><strong>Display State</strong></td>
</tr>
<tr>
<td><strong>Display Hardware State</strong></td>
</tr>
<tr>
<td><strong>Idle Screen MIDlet</strong></td>
</tr>
<tr>
<td><strong>JAD (Java Application Descriptor)</strong></td>
</tr>
</tbody>
</table>
### JSR 271 Overview

<table>
<thead>
<tr>
<th><strong>Item</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>JAR (Java Archive)</td>
<td>An archive used to distribute a MIDlet suite or a LIBlet, containing compiled Java classes and associated resources and metadata that constitute MIDP applications or LIBlets.</td>
</tr>
<tr>
<td>LIBlet</td>
<td>A shareable software component that one or more MIDlet suites may use at runtime.</td>
</tr>
<tr>
<td>Licensee Open Classes</td>
<td>Also known as <strong>OEM Specific Classes</strong>, proprietary packages that do not logically fall into the category of a Java ME configuration, profile, or optional package.</td>
</tr>
<tr>
<td>Manifest</td>
<td>Contains information that describes the contents of a JAR. Located at <code>/META-INF/MANIFEST.MF</code> within a JAR; same syntax as the JAD and may share the same attributes.</td>
</tr>
<tr>
<td>MIDlet Lifecycle</td>
<td>Defines the protocol between a MIDlet and its execution environment through:</td>
</tr>
<tr>
<td></td>
<td>• a simple, well defined state machine</td>
</tr>
<tr>
<td></td>
<td>• a concise definition of the MIDlet's states</td>
</tr>
<tr>
<td></td>
<td>• APIs to signal changes between MIDlet states</td>
</tr>
<tr>
<td>MIDlet Suite</td>
<td>The fundamental MID application packaging component, consisting of a single JAR containing one or more MIDlets.</td>
</tr>
<tr>
<td>MIDlet Suite Lifecycle</td>
<td>Defines the protocol between a MIDlet Suite and the Application Management Software, including:</td>
</tr>
<tr>
<td></td>
<td>• discovery</td>
</tr>
<tr>
<td></td>
<td>• installation</td>
</tr>
<tr>
<td></td>
<td>• update</td>
</tr>
<tr>
<td></td>
<td>• invocation</td>
</tr>
<tr>
<td></td>
<td>• removal</td>
</tr>
<tr>
<td>MIDP Execution Environment</td>
<td>Each MIDlet is executed in a separate environment that includes the MIDlet suite's classes, its shared libraries and the requested configuration, profile and optional APIs.</td>
</tr>
<tr>
<td>Packaging</td>
<td>Collectively, the techniques and components used to prepare MIDlet suites for distribution, including Java archive (JAR), Manifest, Java Application Descriptor (JAD), and resources.</td>
</tr>
<tr>
<td>Permission</td>
<td>Represents a specific access to a particular resource, such as an API or function, and the authorized use of that resource.</td>
</tr>
<tr>
<td>Persistent MIDlet Suite</td>
<td>A MIDlet Suite that cannot be deleted by the user. See <code>javax.microedition.midlet</code> for more details.</td>
</tr>
<tr>
<td>Provisioning</td>
<td>The process of discovering, deploying, and maintaining content between a server and a client device.</td>
</tr>
<tr>
<td>RMS</td>
<td>Defined in <code>javax.microedition.rms</code>, Record Management System allows MID applications to persistently store and later retrieve data.</td>
</tr>
<tr>
<td>RMS Record Tag</td>
<td>An integer tag associated with a record in an RMS record store, specified at runtime by a MIDlet to either assign the tag to a record or to be used as a filter while searching a record store.</td>
</tr>
<tr>
<td>Root Certificate</td>
<td>A certificate or equivalent entity occupying the top-most position in a certificate chain and forming the root trust level that is inherited by all certificates below it in the certificate chain.</td>
</tr>
<tr>
<td>Screen Saver MIDlet</td>
<td>A MIDlet that can be launched automatically when the device has been in an idle state for a predetermined amount of time.</td>
</tr>
</tbody>
</table>
JSR 271 Overview

<table>
<thead>
<tr>
<th>Security Policy</th>
<th>Defines the mapping for a set of permissions that are available to applications running within each of the individual security protection domains for a device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Protection Domain</td>
<td>Associates a MIDlet suite with a defined set of permissions and related interaction modes that allow access to the functionality protected by the domain, based on the permissions granted.</td>
</tr>
<tr>
<td>Versioning</td>
<td>A scheme to uniquely identify a MIDlet suite, LIBlet, or API by major, minor, and micro version, to enable identification for provisioning and dependency expression.</td>
</tr>
<tr>
<td>X.509 Public Key Infrastructure</td>
<td>A cryptography standard for that specifies standard formats for public key certificates and a certificate chain validation algorithm.</td>
</tr>
</tbody>
</table>

1.8 Contributors

This specification was produced by the JSR 271 Expert Group, as a part of the Java Community Process. The following companies and individuals, listed in alphabetical order, are members of the Expert Group:

- Companies:
  - Access Co., Ltd
  - Alticast Corporation
  - Aplix Corporation
  - ARM Limited
  - Aromasoft Corporation
  - Beijing ZRRT Communications Technology Co., Ltd.
  - BenQ Mobile
  - Cellon France SAS
  - China Mobile Communications Corporation
  - Cingular Wireless
  - Cisco Systems
  - Communology
  - Danger, Inc.
  - Digital Chocolate
  - Ericsson AB
  - Esmertec AG
  - Fathammer
  - Gemplus Corporation
  - Google
  - Hutchison 3G UK
  - Hybrid Graphics
  - IBM
  - Infospace
  - Institute for Information Industry
  - Intel Corporation
  - JAMDAT Mobile Inc.
  - LG Electronics
  - LG Soft India
  - Magmic Games
  - MobileScope
  - Monotype Imaging
  - Motorola, Inc.
  - Mr. Goodliving Ltd.
  - Musiwave
JSR 271 Overview

- Nokia Corporation
- NTT DoCoMo, Inc.
- Orange France SA
- OZ Communication Inc.
- Philips
- ProSyst Software GmbH
- Research in Motion
- Samsung Electronics Corporation
- SavaJe Technologies Inc.
- Seven Networks, Inc.
- Sharp Corporation
- Siemens AG
- SK Telecom
- Sony Ericsson Mobile Communications
- Sorrent, Inc.
- Sprint Nextel
- Sun Microsystems, Inc.
- Symbian
- Telecom Italia
- TELUS Mobility
- Tira Wireless
- T-Mobile
- Veloxsoft, Inc.
- Vodafone
- WideRay, Inc.
- Wisegram, Inc.
- XCE Co., Ltd.

- Individuals:
  - Luiz Carlos Bentes dos Anjos
  - Paul Golding
  - Jesper Zuschlag
  - Neil Katin
  - C. Enrique Ortiz
  - Marco Tessarotto
  - Daniel Wandarti
  - Wei-Meng Chee
  - Herval Freire
  - Marion Luz
  - Stefano Andreani

1.9 Introduction

This document, produced as a result of Java Specification Request (JSR) 271, defines the Mobile Information Device Profile (MIDP) v3.0 for the Java Platform, Micro Edition (Java ME™). 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 3.0 specification is based on the MIDP 2.1 specification, and provides backward compatibility with MIDP 2.1 so that MIDlets written for MIDP 2.1 can execute in MIDP 3.0 environments.
JSR 271 Overview

MIDP 3.0 is designed to operate on top of the Connected Limited Device Configuration (CLDC), which is described in CLDC 1.1 (JSR 139), but will also work on top of CDC 1.1 (JSR 218). It is anticipated that most MIDP 3.0 implementations will be based on CLDC.

1.10 New Features

MIDP 3.0 introduces a number of new features that build upon the previous success of MIDP. These new features are intended to enable a wide range of applications and services, as well as to improve the stability, and predictability of MIDP implementations. These new features are:

- Support for both CLDC and CDC
- System and Application Events
- Concurrency
- InterMIDlet Communication
- RMS Data Provisioning & RMS Interchange File Format
- Shared Components (LIBlets)
- MIDlet types (Screen Savers, Idle Screen MIDlets, Auto Start MIDlets)
- Multiple MIDlet Suite Signers
- LCDUI enhancements: Support for multiple displays, FileSelector, tabs
- Notifications
- Java SE-like model for permissions
- Support for preinstalled, packaged and downloaded OpenType fonts

1.11 Scope

Mobile Information Devices (MIDs) span a potentially wide set of capabilities. Rather than attempt to address all such capabilities, previous MIDP Experts Groups (JSR 037 and JSR 118) agreed to limit the set of APIs specified, addressing only those functional areas that were considered absolute requirements needed to achieve broad portability and successful deployments. The MIDP 3.0 Expert Group continued and extended this philosophy to expand upon the existing MIDP functionality in all areas, improve interoperability across implementations, and build upon the success of MIDP 2.1 by enhancing the profile in the following areas:

- Platform: enable and specify proper MIDlet behavior on CLDC and CDC
- Shared Components: enable statically shared libraries
- Application Lifecycle (i.e., defining the semantics of a MIDP application and how it is controlled): clarify and extend to specify proper behavior of concurrently executing MIDlets
- Security Framework and Security Policy
- Networking: connection preferences
- Persistent storage: define a secure binary interchange format for the provisioning of record stores
- 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, beginning in Chapter 10.

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 relies on the Configuration for the integrity of bytecode verification and execution.
• Service oriented architecture: The MIDP requires no specific service oriented architecture, in which resources (local or remote) are made available via independent services. An inter-MIDlet communications architecture is specified, but the underlying implementation can be as simple or complex as the native platform allows.

1.12 Architecture

This section addresses issues that both implementers and developers will encounter when working with MIDP. While not comprehensive, this section does reflect the most important issues raised during deliberations of the MIDP 3.0 Expert Group.

As previously stated, the goal of the MIDP is to create an open third-party application development environment for MIDs. Ideally, this specification would only have to address functionality defined by the MIDP specification itself. In reality, most devices that implement the MIDP specification will incorporate features and components that exist in today's mobile devices market. The High Level Architecture diagram (Figure 1-1) provides an overview of how MIDP 3.0 fits into a mobile information device software architecture. Note that not all devices that implement the MIDP 3.0 specification will incorporate all of the elements depicted in Figure 1-1, nor will every device necessarily implement a software stack precisely as depicted in this figure.

Figure 1-1 - MIDP High Level Architecture

In the MIDP High Level Architecture, the lowest level block (labeled Mobile Information Device) represents the device hardware. Logically above this hardware is the native device system software; this layer includes the operating system and runtime libraries used by the device.

Logically above the native system software is the Java Virtual Machine and configuration layer. This block represents the Java Virtual Machine and associated libraries defined by the CLDC and/or CDC specifications. This block provides the underlying Java functionality upon which higher level Java APIs may be built, such as MIDP and optional packages.
Licensee Open Classes, sometimes also called OEM Specific Classes, are not JSRs, and are generally proprietary packages that do not logically fall into the category of a Java ME configuration, profile, or optional package.

OEM Specific Applications may make use of non-standard packages and APIs, and may be MIDlets. Native Applications are non-Java applications that make use of the device system software, but may also interact and share data with MIDP implementations.

1.13 Device Requirements

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

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

1.13.1 Hardware

As previously stated, the main goal of the MIDP is to establish an open, third-party application development environment for MIDs. To achieve this goal, the MIDP 3.0 Expert group has defined a MID to be a device that SHOULD have the following minimum characteristics:

- Display:
  - Screen-size: 176x220
  - Display depth: 16-bit
  - Pixel shape (aspect ratio): approximately 1:1
- Input:
  - One or more of the following user-input mechanisms: one-handed keyboard, two-handed keyboard, touch screen, or scroll wheel
- Memory:
  - 1 megabyte of non-volatile memory for the MIDP implementation, beyond what's required for CLDC or CDC.
  - 512 kilobytes of non-volatile memory for application-created persistent data
  - 1 megabyte of volatile memory for the Java runtime (e.g., the Java heap)
- Networking:
  - One or more logical network interfaces
  - Two-way, wireless connections, 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, wireless-enabled personal digital assistants (PDAs), and handheld multimedia devices.

1.13.2 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
JSR 271 Overview

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.

1.14 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 3.0 implementations:

- MUST support MIDP 1.0, MIDP 2.0, and MIDP 2.1 MIDlets and MIDlet suites.
- MUST include all packages, classes, and interfaces described in this specification.
- MUST implement the MIDP 3.0 Provisioning specification.
- MAY incorporate zero or more supported protocols for PushRegistry.
- 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 via gateway services.
- MUST provide support for secure HTTP connections either directly, or via gateway services.
- SHOULD provide support for datagram 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 SVG Tiny version 1.1 as a scalable image format, and MAY additionally support later, compatible versions.
- 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 MIDlet Suites Trust Model Using X.509 PKI to recognize signed MIDlet suites as trusted unless PKI is not used by the device for signing applications.
JSR 271 Overview

- 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 defined in [RFC2279] 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 and any resulting copies will be copy protected in a manner consistent with the implementation's policy.

1.15 References

[CDC] Connected Device Configuration (JSR 218) 1.1, Sun Microsystems, Inc. Available at: http://jcp.org/jsr/detail/218.jsp

[CLDC] Connected, Limited Device Configuration 1.1 (JSR 139), Sun Microsystems, Inc. Available at: http://jcp.org/jsr/detail/139.jsp


[JSR75] PDA Optional Packages for the J2ME™ Platform, IBM, Inc. and PalmSource, Inc. Available at: http://jcp.org/jsr/detail/75.jsp

[JSR82] Java™ APIs for Bluetooth, Motorola, Inc. Available at: http://jcp.org/jsr/detail/82.jsp

[JSR113] Java™ Speech API 2.0, Conversational Computing Corp. Available at: http://jcp.org/jsr/detail/113.jsp

[JSR135] Mobile Media API (JSR 135) 1.2, Nokia Corp. Available at: http://jcp.org/jsr/detail/135.jsp

[JSR172] J2ME™ Web Services Specification (JSR 172), Sun Microsystems, Inc. Available at: http://jcp.org/jsr/detail/172.jsp

[JSR177] Security and Trust Services API for J2ME™ (JSR 177), Sun Microsystems, Inc. Available at: http://jcp.org/jsr/detail/177.jsp

[JSR179] Location API for J2ME™ (JSR 179), Nokia Corp. Available at: http://jcp.org/jsr/detail/179.jsp

[JSR180] SIP API for J2ME™ (JSR 184), Nokia Corp. Available at: http://jcp.org/jsr/detail/180.jsp


[JSR205] Wireless Messaging API (JSR 205) 2.0, Siemens AG. Available at: http://jcp.org/jsr/detail/205.jsp

[JSR211] Content Handler API (JSR 211), Sun Microsystems, Inc. Available at: http://jcp.org/jsr/detail/211.jsp
JSR 271 Overview


[JSR234] Advanced Multimedia Supplements (JSR 234), Nokia Corp. Available at: http://jcp.org/jsr/detail/234.jsp

[JSR238] Mobile Internationalization API (JSR 238), Nokia Corp. Available at: http://jcp.org/jsr/detail/238.jsp

[JSR239] Java™ Binding for the OpenGL® ES API (JSR 239), Sun Microsystems, Inc. Available at: http://jcp.org/jsr/detail/239.jsp

[JSR248] Mobile Service Architecture (JSR 248), Nokia Corp. and Vodafone Group Services Ltd. Available at: http://jcp.org/jsr/detail/248.jsp

[JSR253] Mobile Telephony API (MTA) (JSR 253), Motorola, Inc. Available at: http://jcp.org/jsr/detail/253.jsp

[JSR256] Mobile Sensor API (JSR 256), Nokia Corp. Available at: http://jcp.org/jsr/detail/256.jsp

[JSR257] Contactless Communication API (JSR 257), Nokia Corp. Available at: http://jcp.org/jsr/detail/257.jsp

[JSR258] Mobile User Interface Customization API (JSR 258), Nokia Corp. Available at: http://jcp.org/jsr/detail/258.jsp

[JSR272] Mobile Broadcast Service API for Handheld Terminals (JSR 272), Nokia Corp. and Motorola, Inc. Available at: http://jcp.org/jsr/detail/272.jsp

[JSR280] XML API for Java™ ME (JSR 280), Sun Microsystems, Inc. and Nokia Corp. Available at: http://jcp.org/jsr/detail/280.jsp

[JSR290] Java™ Language & XML User Interface Markup Integration (JSR 290), Sun Microsystems, Inc. Available at: http://jcp.org/jsr/detail/290.jsp

[JSR293] Location API 2.0 (JSR 293), Nokia Corp. Available at: http://jcp.org/jsr/detail/293.jsp


[MIDP1] Mobile Information Device Profile (JSR 37) 1.0, Motorola, Inc. Available at: http://jcp.org/jsr/detail/37.jsp

[MIDP2] Mobile Information Device Profile (JSR 118) 2.1, Motorola, Inc. Available at: http://jcp.org/jsr/detail/118.jsp


[RFC2068]
JSR 271 Overview


[USIM] Jacques SEIF, Characteristics of the Universal Subscriber Identity Module (USIM) application, 3GPP TS 31.102, 3rd Generation Partnership Project, Technical
JSR 271 Overview


## Package Summary

<table>
<thead>
<tr>
<th>Package Summary</th>
<th>javax.microedition.lcdui</th>
<th>The APIs in the LCDUI package provide a set of features for implementing user interfaces in MIDP applications.</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Interface Package</td>
<td><strong>javax.microedition.lcdui.game</strong></td>
<td>The Game API package provides a series of classes that enable the development of rich gaming content for wireless devices.</td>
</tr>
<tr>
<td>Game Package</td>
<td><strong>javax.microedition.midlet</strong></td>
<td>The MIDlet package defines Mobile Information Device Profile (MIDP) applications and the interactions between the application and the environment in which the application runs.</td>
</tr>
<tr>
<td>Application Lifecycle Package</td>
<td><strong>javax.microedition.io</strong></td>
<td>MID Profile includes networking support based on the Generic Connection framework from the Connected, Limited Device Configuration.</td>
</tr>
<tr>
<td>Networking Package</td>
<td><strong>javax.microedition.pki</strong></td>
<td>Certificates are used to authenticate information for secure Connections.</td>
</tr>
<tr>
<td>Public Key Package</td>
<td><strong>javax.microedition.media</strong></td>
<td>MIDP 3.0 requires the support of the JSR 135 Mobile Media API v1.1 specification.</td>
</tr>
<tr>
<td>Sound</td>
<td><strong>javax.microedition.rms</strong></td>
<td>The Mobile Information Device Profile provides a mechanism for MIDlets to persistently store data and later retrieve it.</td>
</tr>
<tr>
<td>Persistence Package</td>
<td><strong>java.lang</strong></td>
<td>Descriptions of modified and extended features included from Java Platform Micro Edition Configurations (CLDC and CDC).</td>
</tr>
<tr>
<td>Core Packages</td>
<td><strong>javax.microedition.event</strong></td>
<td>Package to provide events for application to application communication and provide access to and events for changes in system states.</td>
</tr>
</tbody>
</table>
MIDP 3.0 Packaging

2.1 Overview and Goals

This chapter provides a high level overview of how MIDlet suites and LIBlets (or shared libraries) are packaged; that is, the logical packaging formats, the required and optional files, and the resources, content, and attributes that all MIDlet suites and LIBlets must support as a requirement of MIDP 3.0 Packaging. Adherence to these requirements and definitions will ensure content interoperability among MIDP 3.0 implementations, and will also provide guidance to MIDP 3.0 content developers and tools providers.

2.2 Functional Requirements

A MIDP 3.0 compliant device MUST be capable of:

- Correctly recognizing and reacting to all mandatory MIDP 3.0 Application Attributes.
- Correctly recognizing and reacting to all optional Application Attributes that may be presented.

2.3 MIDlet Suites and LIBlets

A MIDlet suite is the fundamental MID application packaging component. It contains one or more MIDlets that the user can select to be executed.

The elements of a MIDlet suite are:

- MIDP 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 delete MIDlets. The term Application Management Software (AMS) is used to refer collectively to these device specific functions. The AMS provides an environment in which the MIDlet is installed, started, stopped, and deleted. 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.

A LIBlet is a shareable software component that one or more MIDlets MAY use at runtime. Support for multiple applications to share a common component enables great flexibility and scalability to a platform. LIBlets save static footprint size by enabling multiple MIDlet suites to share the same common code without packaging them redundantly. LIBlets effectively reduce download times for applications that declare dependencies on shared components.

LIBlets are inert code fragments that have no execution context of their own. Classes contained within a LIBlet MUST NOT execute outside the context of the MIDP execution environment that exists for a MIDlet. Each LIBlet exposes a set of classes and resources to applications for their use,
just as if those classes and resources were originally packaged within the MIDlet JAR. The support for LIBlets is achieved by the presence of LIBlet specific attributes in the JAD and JAR. No runtime specific APIs exist to support LIBlets. For more information about binding LIBlets, see the Execution Environment section.

2.4 Packaging

A single JAR is used to package either a MIDlet suite or a LIBlet. The contents of this JAR include:

- A manifest describing the contents of the MIDlet suite or LIBlet
- Java classes for the MIDlet(s) and classes shared by the MIDlets, or Java classes representing the contents of the LIBlet
- Resource files used by the MIDlet(s) or LIBlet
- Persistent data in the MIDP 3.0 RMS Interchange Format

The developer is responsible for creating and distributing the components of the JAR 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.

2.4.1 The JAR Manifest

The JAR manifest provides information about the contents of the JAR and 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 list of attributes and their allowed values can be found in Application Attributes.

Information about the JAR format and the manifest syntax is available in the JDK documentation at http://java.sun.com/j2se/1.5.0/docs/guide/jar/index.html. MIDP implementations MUST implement handling of lines longer than 72 bytes (not characters) as defined in the manifest specification.

Manifest attributes are passed to the MIDlet when requested using the MIDlet.getAppProperty method. An attribute MAY appear in both the manifest and in the application descriptor, but they MUST have identical values. There is no ambiguity in the value of attributes; each appears either in the manifest or in the application descriptor exclusively, or has exactly the same value in both.

Any attribute that is exactly duplicated within the manifest, irrespective of the attribute value, MUST be treated as an invalid packaging, and MUST result in a failed installation with an installation status report of Status Code 906 (Invalid Descriptor) in the status report.

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-Name-<locale> for each desired locale
MIDP 3.0 Packaging

- MIDlet-<n>-<locale> for each MIDlet and each desired locale
- MIDlet-<n>-Category for each MIDlet
- MIDlet-Operator-Allowed
- MIDlet-Dependency-<n> for each LIBlet the MIDlet depends on
- MIDlet-Dependency-Jar-SHA1-<n> for each LIBlet the MIDlet depends on
- MIDlet-Description
- MIDlet-Description-<locale> for each desired locale
- MIDlet-Icon
- MIDlet-Icon-<locale> for each desired locale
- MIDlet-Font
- MIDlet-Info-URL
- MIDlet-Data-Size
- MIDlet-Permissions
- MIDlet-Permissions-Opt
- MIDlet-Permission-<n> for each permission that the MIDlet needs
- MIDlet-Permission-Opt-<n> for each permission that the MIDlet needs
- MIDlet-Push-<n> for each push registration
- MIDlet-Install-Notify
- MIDlet-Delete-Notify
- MIDlet-Delete-Confirm
- MIDlet-Delete-Confirm-<locale> for each desired locale
- MIDlet-<n>-Type for each MIDlet
- MIDlet-Minimum-Canvas-Size
- MIDlet-Maximum-Canvas-Size
- MIDlet-Splash-Screen-Image
- MIDlet-<n>-UserDenied for each MIDlet
- MIDlet-UserDenied
- MIDlet-Update-URL
- MIDlet-Required-IP-Version
- MIDlet-Scalable-Icon
- MIDlet-Scalable-Icon-<locale> for each desired locale
- MIDlet-<n>-Scalable-Icon for each MIDlet
- MIDlet-<n>-Scalable-Icon-<locale> for each MIDlet and each desired locale
- MIDlet-Event-Launch-<n> for each static launch event
- MIDlet-Persistent-Data-URL-<n> for each Persistent Data file
- MIDlet-Access-Auth-Type-<n> for each restricted access
- MIDlet-Access-Auth-Cert-<n> for each restricted access
- LIBlet-Description
- LIBlet-Description-<locale> for each desired locale
- LIBlet-NonShared-Data-Size
- LIBlet-Shared-Data-Size
- LIBlet-Dependency-<n> for each LIBlet that the LIBlet depends on
- LIBlet-Install-Notify
- LIBlet-Persistent-Data-URL-<n> for each Persistent Data file
- LIBlet-Icon
- LIBlet-Delete-Notify
- LIBlet-Font
- LIBlet-Permissions
- LIBlet-Permissions-Opt
- LIBlet-Permission<n> for each permission that the LIBlet needs
- LIBlet-Permission-Opt<n> for each permission that the LIBlet needs
2.4.2 The Application Descriptor

Each JAR 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 of the MIDlet suite. It also allows configuration-specific attributes (parameters) to be supplied to the MIDlet(s) without modifying the JAR.

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 application descriptor are made available to the MIDlet(s). The list of attributes and their allowed values can be found in Application Attributes.

Application developers and implementers MAY define attributes not beginning with MIDlet-, MicroEdition-, or LIBlet- for application-specific purposes. Vendor and application specific attributes SHOULD have a prefix that identifies the company or other entity. Attribute names are case-sensitive and MUST match exactly. An attribute MUST NOT appear more than once within the application descriptor. Any attribute that is exactly duplicated within the application descriptor, irrespective of the attribute value, MUST be treated as an invalid packaging, and MUST result in a failed installation with an installation status report of **Status Code 906 (Invalid Descriptor)** in the status report. The MIDlet can retrieve an attribute 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**
MIDP 3.0 Packaging

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

- MicroEdition-Profile
- MicroEdition-Configuration
- MIDlet-Permission-<n>
- MIDlet-Permission-Opt-<n>
- MIDlet-Minimum-Canvas-Size
- MIDlet-Maximum-Canvas-Size

The application descriptor MAY additionally contain:

- MIDlet-<n> for each MIDlet
- MIDlet-<n>-<locale> for each MIDlet and each desired locale
- MicroEdition-Profile
- MicroEdition-Configuration
- MIDlet-Description
- MIDlet-Description-<locale> for each desired locale
- MIDlet-Icon
- MIDlet-Icon-<locale> for each desired locale
- MIDlet-Font
- MIDlet-Info-URL
- MIDlet-Data-Size
- MIDlet-Permissions
- MIDlet-Permissions-Opt
- MIDlet-Permission-<n> for each permission that the MIDlet needs
- MIDlet-Permission-Opt-<n> for each permission that the MIDlet needs
- MIDlet-Push-<n> for each push registration
- MIDlet-Install-Notify
- MIDlet-Delete-Notify
- MIDlet-Delete-Confirm
- MIDlet-Delete-Confirm-<locale> for each desired locale
- MIDlet-<n>-Type for each MIDlet
- MIDlet-Splash-Screen-Image
- MIDlet-<n>-UserDenied< for each MIDlet/li>
- MIDlet-UserDenied
- MIDlet-Update-URL
- MIDlet-Required-IP-Version
- MIDlet-<n>-Type for each MIDlet
- MIDlet-<n>-Category for each MIDlet
- MIDlet-Operator-Allowed
- MIDlet-Dependency-<n> for each LIBlet the MIDlet depends on
- MIDlet-Dependency-JAD-URL-<n> for each LIBlet the MIDlet depends on
- MIDlet-Dependency-Jar-SHA1-<n> for each LIBlet the MIDlet depends on
- MIDlet-Scalable-Icon
- MIDlet-Scalable-Icon-<locale> for each desired locale
- MIDlet-<n>-Scalable-Icon for each MIDlet
- MIDlet-<n>-Scalable-Icon-<locale> for each MIDlet and each desired locale
- MIDlet-Event-Launch-<n> for each static launch event
- MIDlet-Persistent-Data-URL-<n> for each Persistent Data file
- MIDlet-Access-Auth-Type-<n> for each restricted access
MIDP 3.0 Packaging

- MIDlet-Access-Auth-Cert-<n> for each restricted access
- MIDlet-Certificate-<n>-<m> for each certificate in the certificate chain and for each certificate chain
- MIDlet-Jar-RSA-SHA1-<n> for each certificate chain
- MIDlet-Jar-RSA-SHA1
- MIDlet-Profile-Request
- MIDlet-Name-<locale> for each desired locale
- LIBlet-Persistent-Data-URL-<n> for each Persistent Data file
- LIBlet-Icon
- LIBlet-Description
- LIBlet-Description-<locale> for each desired locale
- LIBlet-Install-Notify
- LIBlet-Delete-Notify
- LIBlet-NonShared-Data-Size
- LIBlet-Shared-Data-Size
- LIBlet-Jar-SHA1
- LIBlet-Dependency-<n> for each LIBlet that the LIBlet depends on
- LIBlet-Dependency-JAD-URL-<n> for each LIBlet that the LIBlet depends on
- LIBlet-Dependency-Jar-SHA1-<n> for each LIBlet that the LIBlet depends on
- LIBlet-Font
- LIBlet-Permissions
- LIBlet-Permissions-Opt
- LIBlet-Permission-<n> for each permission that the LIBlet needs
- LIBlet-Permission-Opt-<n> for each permission that the LIBlet needs
- Any application-specific attributes that do not begin with MIDlet-, LIBlet- 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.

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.

For example, an application descriptor for a hypothetical MIDlet suite of card games would look like the following example:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDlet-Name:</td>
<td>CardGames</td>
</tr>
<tr>
<td>MIDlet-Version:</td>
<td>1.1.9</td>
</tr>
<tr>
<td>MIDlet-Vendor:</td>
<td>CardsRUS</td>
</tr>
<tr>
<td>MIDlet-1: Solitaire,</td>
<td>/Solitare.png, com.cardsrus.org.Solitaire</td>
</tr>
<tr>
<td>MicroEdition-Profile:</td>
<td>MIDP-3.0</td>
</tr>
<tr>
<td>MicroEdition-Configuration:</td>
<td>CLDC-1.1</td>
</tr>
<tr>
<td>MIDlet-Description:</td>
<td>Really cool card games</td>
</tr>
<tr>
<td>MIDlet-Jar-URL:</td>
<td><a href="http://www.cardsrus.com/games/cardgames.jar">http://www.cardsrus.com/games/cardgames.jar</a></td>
</tr>
<tr>
<td>MIDlet-Jar-Size:</td>
<td>7378</td>
</tr>
<tr>
<td>MIDlet-Data-Size:</td>
<td>256</td>
</tr>
<tr>
<td>MIDlet-Minimum-Canvas-Size:</td>
<td>120, 120</td>
</tr>
</tbody>
</table>

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
MIDP 3.0 Packaging

need to be read through the equivalent of java.io.InputStreamReader with the appropriate
encoding. The UTF-8 character encoding is the default character encoding for application
descriptors, unless otherwise specified. Descriptors retrieved via HTTP, if that is supported,
SHOULD use the standard HTTP content negotiation mechanisms, such as the ContentEncoding header and the charset parameter of the Content-Type header, to correctly convert
the stream to Unicode.
Application descriptors, especially those with many localized versions of the attributes, may contain
characters used in many different languages and writing systems. It is RECOMMENDED that UTF-8
be used as the character encoding of descriptors, and that the encoding is correctly indicated in the
HTTP headers.
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

2.4.3 MIDlet Suite Packaging
A MIDlet suite is packaged as a single JAR which MUST contain one or more MIDlets. The JAR
contains class files, resources, and a manifest describing the JAR contents. A MIDlet suite MAY be
accompanied by an associated JAD. All MIDlet and user defined application attributes present in
both the MIDlet JAD and Manifest MUST match exactly.
The following attributes MUST NOT be present in the MIDlet Manifest, and MUST be present only in
the MIDlet JAD :
• MIDlet-Jar-URL
• MIDlet-Jar-Size
• MIDlet-Dependency-JAD-URL-<n>

2.4.4 LIBlet Packaging
A LIBlet is packaged in a JAR, in a similar way to a MIDlet suite. A LIBlet JAR contains class files,
associated resources, and a Manifest describing the contents. A LIBlet JAR MUST contain exactly
one LIBlet. A LIBlet MUST have an associated JAD. All LIBlet and user defined application attributes
MUST be present in both the LIBlet JAD and LIBlet Manifest, and they MUST match exactly, with the
following exceptions that MUST NOT be present in the LIBlet Manifest, and MUST be present only in
the LIBlet JAD :
• LIBlet-Jar-URL
• LIBlet-Jar-Size
• LIBlet-Jar-SHA1

Mobile Information Device Profile v3.0 - JSR 271
© Copyright Motorola, Inc. 2002-2009. All Rights Reserved.

Java Community Process - Final Release
Page 23 of 891


MIDP suites SHOULD NOT modify any attributes in nor add any attributes to the JAD or Manifest of LIBlets they depend on, since doing so will result in an Application Integrity Failure due to a hash value for the LIBlet that differs from the one generated by the LIBlet developer. MIDlet suites SHOULD locate any additional MIDlet attributes in the MIDlet suite's manifest and JAD.

At installation time, when binding MIDlet suites together with independently developed LIBlets, there is potential for namespace conflicts if the MIDlet suite and LIBlets implement classes with the same name in the same packages. Implementations MUST ensure there are no namespace conflicts at install time. The MIDlet suite and the LIBlets it depends on MUST be packaged in non-overlapping namespaces, with package level granularity. Implementations MUST also ensure that resources with the same name do not exist in the MIDlet suite-LIBlet binding. It is RECOMMENDED that resources are placed in packages alongside the classes that use them to prevent resource level namespace collisions. Implementations MUST enumerate all the MIDlet suite JARs and LIBlet JARs to ensure that none of the described collisions exist. The values of attributes defined for a LIBlet MUST be retrieved with the LIBlet name and vendor; thus no namespace collision issue exists across LIBlets.

### 2.5 Application Attributes

This section describes the application attributes predefined by MIDP, and their allowed values. Refer to Appendix A - MIDP 3.0 Application Attributes for details on how application attributes can be included in the JAR manifest and the application descriptor.

Among other things, attributes allow the application management software to identify, retrieve, install, and invoke the MIDlet.

Any attribute that is exactly duplicated within the manifest, irrespective of the attribute value, MUST be treated as an invalid packaging, and MUST result in a failed installation with an installation status report of Status Code 906 (Invalid Descriptor) in the status report.

MIDlets must not add any attributes to the MIDlet suite JAR manifest or the Application Descriptor that begin with MIDlet- or MicroEdition- other than those defined in the relevant Configuration and Profile (e.g. CLDC and MIDP) specifications. In addition, MIDlets must not add any attributes that begin with LIBlet-. Unrecognized attributes that begin with MIDlet- or MicroEdition- and all attributes that begin with LIBlet- MUST be ignored by the AMS, MUST be discarded, and their values MUST NOT be made available to a running MIDlet.

LIBlets must not add any attributes to the LIBlet JAR manifest or the LIBlet JAD file that begin with LIBlet- or MicroEdition- other than those defined in this specification. In addition, LIBlets must not add any attributes that begin with MIDlet-. Unrecognized attributes that begin with LIBlet- or MicroEdition- and all attributes that begin with MIDlet- MUST be ignored by the AMS, MUST be discarded, and their values MUST NOT be made available to a running MIDlet.

For ease of reference, the attributes are grouped by usage as follows:

- MIDlet Identification Attributes
- Identification Attributes for LIBlets
- Provisioning Attributes for MIDlet Suites and MIDlets
- Provisioning Attributes for LIBlets
- Operational Attributes for MIDlet Suites and MIDlets
- Operational Attributes for LIBlets
In various attributes, \(<n>\) (and \(<m>\)) represent values of the integer sequence \(n = \{1, 2, \ldots\}\). The first non-consecutive value of \(n\) terminates the list, and any additional values MUST be ignored.

\(<\text{locale}>\) represents a valid value of the \texttt{microedition.locale} system property. See the \textit{Handling Localized Attribute Versions} section for more information.

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 for each value.

### 2.5.1 Identification Attributes

Identification attributes are used by the AMS in various phases of the MIDlet lifecycle.

The following table defines the identification attributes used for MIDlet suites and individual MIDlets.

<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-Name-&lt;locale&gt;</td>
<td>Optional, localized version of the MIDlet suite name for (&lt;\text{locale}&gt;). This attribute MUST NOT be used instead of the original MIDlet-Name for verification. It is only intended for display purposes.</td>
</tr>
<tr>
<td>MIDlet-Version</td>
<td>The version number of the MIDlet suite. Version numbers are formatted so that they can be used by the application management software for install and update purposes, 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 an image file within the JAR, used to represent the MIDlet suite. The image file MUST be in the PNG format. It SHOULD be used when the Application Management Software displays an icon to identify the suite.</td>
</tr>
<tr>
<td>MIDlet-Icon-&lt;locale&gt;</td>
<td>Optional localized version of the MIDlet suite icon for (&lt;\text{locale}&gt;). The image file MUST be in the PNG format. If an icon is not available for the current locale, the value of the MIDlet-Icon attribute MUST be used.</td>
</tr>
<tr>
<td>MIDlet-Scalable-Icon</td>
<td>Optional, representative icon for the MIDlet suite, in SVG format. The value indicates a filename inside the JAR, like MIDlet-Icon.</td>
</tr>
<tr>
<td>MIDlet-Scalable-Icon-&lt;locale&gt;</td>
<td>Localized version of MIDlet-Scalable-Icon. Points to a dedicated scalable icon file for (&lt;\text{locale}&gt;).</td>
</tr>
<tr>
<td>MIDlet-Description</td>
<td>The description of the MIDlet suite.</td>
</tr>
<tr>
<td>MIDlet-Description-&lt;locale&gt;</td>
<td>Optional localized version of the MIDlet suite description for (&lt;\text{locale}&gt;). If a description is not available for the current locale, the value of the MIDlet-Description attribute MUST be used.</td>
</tr>
</tbody>
</table>
| MIDlet-<n>           | \(<\text{name}>\), \(<\text{icon}>\), \(<\text{class}>\)
The name, icon, and class of the \( n \)th MIDlet in the JAR, separated by a comma.

- \(<name>\) is used to identify this MIDlet to the user. The name MUST be present and be non-null.
- \(<icon>\) is the case-sensitive absolute path name of an image within the JAR for the icon of the \( n \)th MIDlet. The icon MAY be omitted. The image file MUST be in the PNG format.

If an implementation displays MIDlet icons, the following requirements apply for bitmap image formats:
  - The application manager MUST always be able to display an icon of 16 by 16 pixels in the application list.
  - The application manager SHOULD always be able to display an icon of 24 by 24 pixels in the application list.
  - An icon larger than 24 by 24 MAY be displayed properly in full size in the application list.
- \(<class>\) is the name of the class extending the javax.microedition.midlet.MIDlet class for the \( n \)th 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 MUST be treated as case sensitive.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDlet-(&lt;n&gt;)-locale</td>
<td>Localized version of the MIDlet-(&lt;n&gt;) attribute. The values are the same as in the MIDlet-(&lt;n&gt;) attribute, except that all the parts of the value - name, icon, and class - are OPTIONAL. If some part is missing, the corresponding part of the MIDlet-(&lt;n&gt;) value MUST be used. If there is no MIDlet-(&lt;n&gt;)-locale attribute present for the current locale, the value of the MIDlet-(&lt;n&gt;) attribute MUST be used.</td>
</tr>
<tr>
<td>MIDlet-(&lt;n&gt;)-Scalable-Icon</td>
<td>Optional, representative scalable icon for the ( n )th MIDlet in the suite. The value is a relative pathname of an SVG Tiny file inside the JAR.</td>
</tr>
<tr>
<td>MIDlet-(&lt;n&gt;)-Scalable-Icon-locale</td>
<td>Optional, representative scalable icon for the ( n )th MIDlet in the suite, for (&lt;locale&gt;). The value is a relative pathname of an SVG Tiny file inside the JAR.</td>
</tr>
</tbody>
</table>

The following table defines the identification attributes used for LIBlets.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBlet-Name</td>
<td>The name of the LIBlet. This attribute identifies that the JAD/JAR entity represents a LIBlet, not a MIDlet. If the JAD/Manifest</td>
</tr>
</tbody>
</table>
contains both LIBlet-Name and MIDlet-Name, then the installation of such a JAD/JAR MUST fail.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBlet-Version</td>
<td>The version number of the LIBlet. Version numbers are <em>formatted</em> so that they can be used by the application management software for install and update purposes, as well as communication with the user.</td>
</tr>
<tr>
<td>LIBlet-Vendor</td>
<td>The vendor of the LIBlet.</td>
</tr>
<tr>
<td>LIBlet-Description</td>
<td>Description of the LIBlet.</td>
</tr>
<tr>
<td>LIBlet-Description-&lt;locale&gt;</td>
<td>Description of the LIBlet for &lt;locale&gt;.</td>
</tr>
<tr>
<td>LIBlet-Icon</td>
<td>An optional image icon associated with this LIBlet. It may be used by the AMS to display LIBlet usage and dependencies for information and troubleshooting on the device. It may also be used by IDE or deployment tools. The image file MUST be in one of the mandatory supported image formats.</td>
</tr>
</tbody>
</table>

### 2.5.2 Provisioning Attributes

Provisioning attributes are used by the AMS when discovering, installing, updating and deleting MIDlet suites and related LIBlets.

The following table lists the MIDlet suite attributes related to provisioning.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDlet-Info-URL</td>
<td>A URL for information further describing the MIDlet suite. The syntax and meaning MUST conform to [RFC3986] and the RFCs that define each scheme.</td>
</tr>
<tr>
<td>MIDlet-Jar-URL</td>
<td>The URL from which the JAR can be loaded. The syntax and meaning MUST conform to [RFC3986] 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.</td>
</tr>
<tr>
<td>MIDlet-Update-URL</td>
<td>The absolute URL to use for automatic updates of the MIDlet suite. The syntax and meaning MUST conform to [RFC3986] and RFCs that define each scheme.</td>
</tr>
<tr>
<td></td>
<td>If the value contains a valid URL, the automatic update of the MIDlet suite MUST be requested from this URL. Note that this overrides the priority rules used to decide which URL is used in updating. In this case other URLs MUST NOT be used, and other rules related to the update MUST remain in effect.</td>
</tr>
<tr>
<td></td>
<td>If the URL is empty, user MUST NOT be able to activate the automatic update feature of the AMS.</td>
</tr>
<tr>
<td>MIDlet-Jar-Size</td>
<td>The number of bytes in the JAR.</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. This MUST include the data requirement of any Record Stores being provisioned using the MIDlet-Persistent-Record-Size-&lt;n&gt; attribute.</td>
</tr>
</tbody>
</table>
### MIDlet-Install-Notify

<url>[;ack]

- `<url>`: The URL to which an HTTP 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 (Invalid Descriptor) in the status report.

- `ack`: Optional value to indicate that installation MUST NOT be considered successful and the MIDlet suite MUST NOT be made available for use if no acknowledgement is received after the installation status report has been sent.

### MIDlet-Delete-Notify

The URL to which a POST request is sent to report the deletion of this MIDlet suite. The device MUST use this URL unmodified. The URL MUST be no longer than 256 UTF-8 encoded characters. If the device receives a URL longer than 256 UTF-8 encoded characters, it MUST reject the installation and return Status Code 906 (Invalid Descriptor) in the status report.

### MIDlet-Delete-Confirm

Text message to be provided to the user when prompted to confirm deletion of this MIDlet suite.

### MIDlet-Delete-Confirm-<locale>

Localized version of the MIDlet-Delete-Confirm attribute for `<locale>`. If there is no version for `<locale>`, the normal value MUST be used.

### MIDlet-Persistent-Data-URL-<n>

The URL from which MIDlet RMS data can be loaded, along with processing instructions.

```
<dataURL> ["overwrite"] ["encryptLocally]
```

- `<dataURL>`: an absolute or relative URL that points to an RMS data file. The value MUST conform to [RFC3986]. If the URL is relative, the base MUST be the root of the MIDlet suite JAR file.

- `overwrite`: if present, data from the RMS file replaces any existing record store of the same name

- `encryptLocally`: if present, the record store is to be encrypted locally

Refer to RMS Data Provisioning for details.

### MIDlet-Certificate-<n>-<m>

The Base64 encoding of a security certificate.

- `<n>`: a number equal to 1 for first certificate chain in the descriptor or 1 greater than the previous number for additional certificate chains. This defines the sequence in which the certificates are tested to see if the corresponding Root Certificate can be found. See the Authenticating a MIDlet Suite section for details.

- `<m>`: a number equal to 1 for the signer’s certificate in a certificate chain or 1 greater than the previous number for any subsequent intermediate certificates.

### MIDlet-Jar-RSA-SHA1-<n>

The Base64 encoding of the JAR signature
### MIDP 3.0 Packaging

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDlet-&lt;n&gt;-Category</td>
<td>Possible values: games, music, video, location, shop, messaging, internet, calls, settings, camera, office, contacts, pictures. The attribute value is a comma separated list of different categories that implementations may use when choosing how to make the application available. Implementations may ignore some or all of the values, or may combine them to make an application available in several ways. Implementations may support additional values that are platform specific.</td>
</tr>
<tr>
<td>MIDlet-Profile-Request</td>
<td>Possible values: true, false. The attribute value specifies true to indicate that application descriptor MUST be downloaded by providing the client profile using UAPprof for Device Identification.</td>
</tr>
</tbody>
</table>

The following table lists the LIBlet attributes related to provisioning.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBlet-Jar-URL</td>
<td>The URL from which the LIBlet JAR can be downloaded.</td>
</tr>
<tr>
<td>LIBlet-Jar-Size</td>
<td>The size of the JAR.</td>
</tr>
<tr>
<td>LIBlet-Jar-SHA1</td>
<td>The SHA1 hash of the LIBlet JAR. If a MIDlet suite or LIBlet declares a dependency on this LIBlet, it MUST use this value in the relevant MIDlet-Dependency-Jar-SHA1-&lt;n&gt; or LIBlet-Dependency-Jar-SHA1-&lt;n&gt; attribute in the MIDlet suite or LIBlet JAD file. This value is provided in the LIBlet JAD file to help the AMS to determine if the LIBlet JAR file needs to be downloaded.</td>
</tr>
<tr>
<td>LIBlet-NonShared-Data-Size</td>
<td>The minimum number of bytes of non-shared persistent data required by this LIBlet. The device must add this value to the Data Size specified by the MIDlet during installation time to guarantee available storage space. The default is zero.</td>
</tr>
<tr>
<td>LIBlet-Shared-Data-Size</td>
<td>The data size, in bytes, of shared record stores owned by this LIBlet. This value is added to the MIDlet-Data-Size of the MIDlet Suite that caused this LIBlet to be installed. Subsequent MIDlet installations that depend on this LIBlet are not tagged with the additional data size requirement. This attribute is OPTIONAL, with the default value zero (0).</td>
</tr>
<tr>
<td>LIBlet-Install-Notify</td>
<td>The URL to which a POST request is sent to confirm successful installation of this LIBlet.</td>
</tr>
<tr>
<td>LIBlet-Delete-Notify</td>
<td>The URL to which a POST request is sent to report deletion of this LIBlet.</td>
</tr>
<tr>
<td>LIBlet-Persistent-Data-URL-&lt;n&gt;</td>
<td>The URL from which LIBlet RMS data can be loaded, along with processing instructions. &lt;dataURL&gt; [&quot; &quot; encryptLocally]</td>
</tr>
</tbody>
</table>
### MIDP 3.0 Packaging

<dataURL> = an absolute or relative URL that points to an RMS data file. The value MUST conform to [RFC3986]. If the URL is relative, the base MUST be the root of the LIBlet suite JAR file.

encryptLocally = if present, the record store is to be encrypted locally

Refer to RMS Data Provisioning for details.

### 2.5.3 Operational Attributes

Operational attributes affect the invocation and runtime behavior of MIDlets.

The following tables list the operational attributes for MIDlet suites, individual MIDlets, and LIBlets that bind to these.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MicroEdition-Profile</td>
<td>The Java ME profiles required, using the same format and value as the system property microedition.profiles (for example &quot;MIDP-3.0&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). The value of this attribute MUST NOT contain different versions of the same profile. A MIDlet suite should declare the single lowest profile version it needs to run. This attribute SHOULD be used by LIBlets and MIDlets.</td>
</tr>
<tr>
<td>MicroEdition-Configuration</td>
<td>The Java ME Configuration required, using the same format and value as the system property microedition.configuration (for example &quot;CLDC-1.1&quot;). This attribute SHOULD be used by LIBlets and MIDlets.</td>
</tr>
<tr>
<td>MIDlet-Permission-&lt;n&gt;</td>
<td>A fully qualified permission class, resource name, and actions that are critical to the function of the MIDlet suite.</td>
</tr>
<tr>
<td>MIDlet-Permission-Opt-&lt;n&gt;</td>
<td>A fully qualified permission class, resource name, and actions that are not-critical to the function of the MIDlet suite.</td>
</tr>
<tr>
<td>MIDlet-Access-Auth-Type-&lt;n&gt;</td>
<td>domain=SELF</td>
</tr>
<tr>
<td>MIDlet-Access-Auth-Cert-&lt;n&gt;</td>
<td>&lt;alias&gt; &lt;Base64 encoded signing certificate&gt; For value semantics, see Declaring Application Level Access Authorization.</td>
</tr>
<tr>
<td>MIDlet-Push-&lt;n&gt;</td>
<td>Register a MIDlet to handle inbound connections. Refer to javax.microedition.io.PushRegistry for details.</td>
</tr>
<tr>
<td>MIDlet-Event-Launch-&lt;n&gt;</td>
<td>Register a MIDlet to launch on an event. Refer to javax.microedition.event.EventManager for details.</td>
</tr>
<tr>
<td>MIDlet-&lt;n&gt;-UserDenied</td>
<td>This attribute allows a MIDlet developer to specify the actions the user is not allowed to perform for a specific MIDlet in the MIDlet Suite. The attribute value MAY contain multiple values that are comma separated.</td>
</tr>
</tbody>
</table>
### MIDP 3.0 Packaging

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MIDP 3.0 Packaging</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MIDlet-UserDenied</strong></td>
<td>This attribute allows the MIDlet developer to specify the actions the user is not allowed to perform on the MIDlet Suite once installed. The attribute value MAY contain multiple of the possible values that are comma separated. Allowed values: update, delete.</td>
</tr>
<tr>
<td><strong>update</strong></td>
<td>The user is not allowed to update this MIDlet Suite.</td>
</tr>
<tr>
<td><strong>delete</strong></td>
<td>The user is not allowed to delete this MIDlet Suite, and it is treated as a Persistent MIDlet Suite.</td>
</tr>
<tr>
<td><strong>MIDlet-Required-IP-Version</strong></td>
<td>Possible values: ipv4, ipv6, any. Specifies which IP version the MIDlets in a suite MUST use. If the device cannot support the requested version, the installation of the MIDlet suite MUST fail. If the required version is not available at runtime, a java.io.IOException MUST be thrown.</td>
</tr>
<tr>
<td><strong>MIDlet-&lt;n&gt;-Type</strong></td>
<td>Possible values: screensaver, autostart, idlescreen. The attribute value is a comma separated list of different types for which the application implements the required behaviors. It is possible for a MIDlet to implement the behaviors of several types. Here &lt;n&gt; corresponds to the n(^{th}) MIDlet in the suite identified using the MIDlet-&lt;n&gt; attribute.</td>
</tr>
<tr>
<td><strong>autostart</strong></td>
<td>The MIDlet implements the Auto Start MIDlet behavior.</td>
</tr>
<tr>
<td><strong>screensaver</strong></td>
<td>This MIDlet implements the screen saver behavior.</td>
</tr>
<tr>
<td><strong>idlescreen</strong></td>
<td>This MIDlet implements the idle screen MIDlet behavior.</td>
</tr>
<tr>
<td><strong>MIDlet-Operator-Allowed</strong></td>
<td>Specifies the allowed MCC (3 digits) and MNC (2 or 3 digits) tuples, separated by a hyphen (Unicode U+002D), to limit the execution to certain networks for any MIDlets within a MIDlet suite bound to the Operator Protection Domain. Individual tuples are separated by a blank (Unicode U+0020). An example is:</td>
</tr>
</tbody>
</table>

- **run**: If specified as the value, the user is not provided a means to launch the MIDlet.
- **stop**: If specified as the value, the user does not have the ability to stop this MIDlet when running from an AMS UI. This does not modify the exit behavior that the MIDlet logic exposes say via a Command.

If the attribute is not present, no user actions are denied for this MIDlet. If other values than those specified above are present, the attribute is treated as invalid, and the installation MUST fail.

If <n> does not have an equivalent MIDlet-<n> attribute defined in the JAD/Manifest, the installation of the MIDlet Suite MUST fail.

Allowed values: run, stop.
MIDP 3.0 Packaging

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDlet-Operator-Allowed</td>
<td>429-01 429-02 310-150</td>
</tr>
<tr>
<td>MIDlet-Font</td>
<td>A comma separated list of case-sensitive absolute file names of OpenType or TrueType font files contained within the JAR. These fonts MUST be loaded by an implementation when MIDlet is invoked.</td>
</tr>
<tr>
<td>MIDlet-Splash-Screen-Image</td>
<td>A comma-separated list of case-sensitive absolute image file names within the JAR to be used as a splash screen. Images may be encoded using any format that is supported by the Image class, including any optional formats that are supported by the device. The images are listed in order of preference, and the first image that can be decoded and displayed on the device will be used for all MIDlets in the suite. No splash screen or a generic splash screen will be shown if this entry is missing or does not list a decodable image. Refer to javax.microedition.lcdui.Image for Image details. If the value of this attribute is set to suppress, the platform SHOULD NOT present additional user interface activity during the launch of the MIDlet, such as clearing the screen, displaying a carrier, manufacturer, or VM specific splash screen, or displaying a 'loading' message.</td>
</tr>
<tr>
<td>MIDlet-Minimum-Canvas-Size</td>
<td>The minimum width and height of a full-screen Canvas that the MIDlet suite is designed for. The width and height are expressed in terms of pixels and refer to the device's primary Display.</td>
</tr>
<tr>
<td>MIDlet-Maximum-Canvas-Size</td>
<td>The maximum width and height of a full-screen Canvas that the MIDlet suite is designed for. The width and height are expressed in terms of pixels and refer to the device's primary Display. If the MIDlet-Minimum-Canvas-Size is also specified, its dimensions must be equal to or smaller than those of the MIDlet-Maximum-Canvas-Size; the manifest is considered invalid if this condition is not met.</td>
</tr>
</tbody>
</table>

Table 2-f : Operational Attributes for LIBlets

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBlet-Font</td>
<td>A comma separated list of case-sensitive absolute file names of OpenType or TrueType font files contained within the JAR. These fonts MUST be loaded by an implementation when MIDlet is invoked to which the LIBlet is bound.</td>
</tr>
</tbody>
</table>

### 2.5.4 Handling Localized Attribute Versions

In the above, all locale identifiers in attribute names (represented by `<locale>`) MUST be interpreted as valid values of the microedition.locale system property. Attributes that pertain to user interface languages not currently present in the device MAY be discarded at installation time to save storage space. Usually one or more localized attribute versions end up being retained, but in the most degenerate case (and depending on the configuration of the device) the application installer could end up retaining only the original default attributes and none of the localized versions. Developers are encouraged to include localized attribute versions for at least those user interface languages they anticipate to be present in their target device configuration (often referred to as a "language pack").
MIDP 3.0 Packaging

The AMS SHOULD use the retained localized attribute versions to show information about the installed MIDlets to the user. First the AMS SHOULD look for a locale-specific attribute where the <locale> part matches the current microedition.locale system property value. If such an attribute version is not found, the AMS SHOULD proceed by successively removing the variant, country, and language part from <locale>, until a match is found or <locale> is exhausted. In the case of a match the localized value MUST be used. If there is no match, the default attribute value MUST be used.

Example scenario: the microedition.locale system property has the value de-DE and the application descriptor (or manifest) contains the MIDlet-Name-de attribute. In this case the AMS first looks for a MIDlet-Name-de-DE attribute, but it is not found, so the country part is removed and a new search is made. This time the MIDlet-Name-de attribute is found, and the AMS uses its value. If the MIDlet-Name-de attribute had not been present either, the AMS would have used the original MIDlet-Name attribute.

2.6 Dependency Expression

Both MIDlet suites and LIBlets MUST declare dependencies for any LIBlet, standard, or proprietary API that the MIDlet suite or LIBlet directly depends on. A MIDlet suite and each of its dependencies forms a distinct binding. Any dependencies that cannot be satisfied by an implementation MUST be treated as an invalid packaging, and the installation MUST fail with Status Code 915 (Missing Dependencies).

2.6.1 Dependency Attributes

The MIDlet-Dependency-<n> and LIBlet-Dependency-<n> attributes allow a MIDlet or a LIBlet to specify a dependency on an implementation of a standard optional API, a proprietary API, or a LIBlet. The format of the value is:

<type>; <level>; <name>; <vendor>; <version>

The components of the value are:

- **<type>** = liblet | standard | proprietary
  Indicates the type of the dependency.
  If the type is liblet, the dependency is on a LIBlet. The implementation looks for a MIDlet-Dependency-JAD-URL-<n> or LIBlet-Dependency-JAD-URL-<n> attribute and downloads the LIBlet JAD from this location.
  If the type is standard, the dependency is on a standard, such as a Java Specification Request (JSR) defined by the Java Community Process, that the device implementation supports.
  If the type is proprietary, the dependency is on a proprietary API that the device implementation supports.

- **<level>** = required | optional
  Indicates the level of the dependency.
  If the level is required, the AMS MUST fail the installation of the MIDlet or LIBlet if this required dependency cannot be resolved during installation. The installation MUST fail if:
  - the dependent LIBlet is not available on the device or cannot be installed from the dependency URL
  - the standard or proprietary API is not available on the device
If the level is optional, the AMS will attempt to resolve this optional dependency at installation time. If the dependency is not available for binding, the installation will proceed. The MIDlet or LIBlet that uses this type of a dependency is capable of running in the absence of the dependency, perhaps with reduced capability.

When a MIDlet/LIBlet specifies dependencies on JSRs and/or proprietary APIs, the AMS MAY trim the execution environment to expose only those JSRs and APIs to reduce system resource usage.

- `<name>`
  If the type is liblet, indicates the name of the LIBlet implementation.
  If the type is standard, indicates the canonical name of the standard. Each JSR will have one or more canonical names as JSR API identifiers; these names are defined in Dependency Names For Standard APIs.
  If the type is proprietary, indicates the name of the API.

- `<vendor>`
  If the type is standard or proprietary, indicates the organization that defined the standard or API. If the type is liblet, indicates the vendor of the LIBlet implementation.

- `<version>`
  If the type is standard or proprietary, indicates the version number of the standard or API. If the type is liblet, indicates the version number of the LIBlet implementation.

A MIDlet or LIBlet can only declare dependency on a specific version of a LIBlet. However, a wildcard character '*' or '+' is allowed to be used for standard and proprietary APIs. At most one wildcard character can be used at a time. The wildcard character '*' can only appear at the end of version string. The wildcard character '+' appears after a specific version and indicates a greater-than-or-equal match. Version examples:

- o * - any version
- o 2.* - any 2.x version
- o 2.1.* - any 2.1.x version
- o 2.1 - specific version
- o 2.0+ - version 2.0 or higher

When the LIBlet dependency is resolved, the matching algorithm to be used depends on the type of the dependency.

If the type is liblet, the match is based solely on the name, vendor, version, and hash fields. If the type is standard or proprietary, the implementation MUST look for a match based on the name, vendor and version fields of the dependency expression.

### 2.6.2 The LIBlet JAD Download URL Attribute

The `MIDlet-Dependency-JAD-URL-<n>` and `LIBlet-Dependency-JAD-URL-<n>` attributes indicate the URL to download the dependency LIBlet from. This URL is used to retrieve the LIBlet if it is not already available on the device. For each `MIDlet-Dependency-<n>` or `LIBlet-Dependency-<n>` attribute in a JAD or manifest that refers to a LIBlet, there MUST also be a corresponding `MIDlet-Dependency-JAD-URL-<n>` or `LIBlet-Dependency-JAD-URL-<n>` attribute in the same JAD or manifest.
2.6.3 The Dependency Hash Attribute

The `MIDlet-Dependency-Jar-SHA1-<n>` and `LIBlet-Dependency-Jar-SHA1-<n>` attributes are used to store the SHA1 hash value of the \(n\)th LIBlet that the MIDlet or LIBlet depends on.

When a MIDlet suite or a LIBlet depends on LIBlets, the SHA1 hash value of the dependent LIBlet JAR MUST be computed, and the Base64 encoded hash MUST be included in a `MIDlet-Dependency-Jar-SHA1-<n>` or `LIBlet-Dependency-Jar-SHA1-<n>` attribute in the JAD and manifest of the MIDlet suite or LIBlet that declares the dependency. For each `MIDlet-Dependency-<n>` or `LIBlet-Dependency-<n>` attribute in a JAD or manifest that is of `<type>` equal to liblet, there MUST also be a corresponding `MIDlet-Dependency-Jar-SHA1-<n>` or `LIBlet-Dependency-Jar-SHA1-<n>` attribute in the same JAD or manifest.

2.6.4 Dependency Declaration Chain

A LIBlet is uniquely identified by the quadruplet of name, vendor, version, and hash. For a given dependency declaration chain, the same LIBlet MAY appear multiple times. However, LIBlets with the same name and vendor but different version and/or hash MUST NOT exist within a single MIDlet suite’s dependency declaration chain. Also, a MIDlet suite name and vendor pair MUST NOT match a LIBlet name and vendor pair within the MIDlet suite’s dependency chain. These situations form an invalid package.

Since LIBlets can in turn depend on other LIBlets, it is possible for MIDlet suites to specify circular dependencies; such dependencies exist whenever a MIDlet suite dependency declaration chain contains a loop. The level of a dependency (optional or required) does not factor into circular dependency checking; implementations MUST enforce circular dependency checking on all dependencies found in the MIDlet suite dependency declaration chain, regardless of the specified level of any dependency in the chain; circular dependencies MUST be treated as an invalid packaging and MUST fail installation. Circular dependencies can be avoided by:

- isolating the circular portion of the two LIBlets into another (third) LIBlet, or
- combining the two LIBlets into a single LIBlet, thus making it a single deployment unit.

For a single MIDlet suite dependency declaration chain, implementations MUST allow a minimum of 16 LIBlet dependencies. As well, implementations SHOULD impose an upper limit on the number of LIBlet dependencies allowed for a single MIDlet suite. If an implementation imposes such an upper limit, and if a MIDlet suite exceeds that limit, then the implementation MUST fail installation.

If a MIDlet suite exceeds the number of LIBlet dependencies allowed by an implementation, it is recommended that the MIDlet developers reduce the number of LIBlet dependencies by combining multiple LIBlets into a single LIBlet.

2.6.5 Dependency Expression Example

As an example, consider a navigation MIDlet that depends on JSR 179 and a Maps LIBlet developed by a map vendor. The Maps LIBlet in turn has dependencies on the crypto package of JSR 177 (`microedition.satsa.crypto`) and a LIBlet that provides location based services (LBS) developed by another vendor. The location services LIBlet then has an optional dependency on the `javax.bluetooth` package of JSR 82 and a proprietary location API.

| Figure 2-1 : Navigation MIDlet Dependencies |
The JAD of the navigation MIDlet contains:

MIDlet-1: Maps-Navi, icon.png, MapsNavi
MIDlet-Name: Maps On The Go
MIDlet-Version: 1.0.0
MIDlet-Vendor: CoolNaviApps
MIDlet-Jar-Size: 12123
MIDlet-Dependency-1: liblet; required; MapsForAll; MapsGalore, Inc.; 1.1.2
MIDlet-Dependency-2: standard; required; microedition.location; JCP; 1.0+
MIDlet-Dependency-JAD-URL-1: http://www.mapsgalore.com/liblets/mapsforall.jad
MIDlet-Dependency-Jar-SHA1-1: 501A202E919B9C98343FAD6F46842412F7A0A783

The JAD of the Maps LIBlet contains:

LIBlet-Name: MapsForAll
LIBlet-Vendor: MapsGalore, Inc.
LIBlet-Version: 1.1.2
LIBlet-Jar-Size: 21544
LIBlet-Jar-URL: http://www.mapsgalore.com/liblets/mapsforall.jar
LIBlet-Dependency-1: standard; required; microedition.satsa.crypto; JCP; 1.0+
LIBlet-Dependency-2: liblet; required; LocSvcS; LocationForAll Inc.; 1.0.3
LIBlet-Dependency-JAD-URL-2: http://www.locsvcs.com/liblets/locsvcs.jad
LIBlet-Dependency-Jar-SHA1-2: 81708e4774db43b630241d124f19b1c03a5c94bb

The JAD of the LBS LIBlet contains:

LIBlet-Name: LocationServicesLIBlet
LIBlet-Version: 1.0.3
LIBlet-Vendor: LocationForAll Inc.
LIBlet-Jar-Size: 23165
LIBlet-Jar-URL: http://www.locationforall.com/liblets/locforall.jar
LIBlet-Dependency-1: standard; optional; bluetooth.api; JCP; 1.0
LIBlet-Dependency-2: proprietary; required; SuperDuper Location API; Mega Software Services Co.; 1.0
2.6.6 Dependency Names For Standard APIs

MIDlet suites and LIBlets that express dependencies on packages defined in standard APIs (JSRs) will do so via the MIDlet-Dependency-<n> or LIBlet-Dependency-<n> application attribute.

Each JSR generally defines system properties so that MIDlets can retrieve API version information. JSR standard dependency names used in the dependency expression are based on the JSR package names used in the system properties for API version retrieval. Table 2-3 defines the standard dependency names for the component JSRs required by Appendix A.1.1 of JSR 248 (Mobile Service Architecture). For JSRs that specify a system property for version retrieval, the format of the version string used when expressing a dependency MUST conform to [JPVS]; any other format for version string in this case MUST be treated as an invalid packaging, and MUST result in a failed installation with an installation status report of Status Code 906 (Invalid Descriptor) in the status report. For JSRs that do not specify a system property for version retrieval, the format of the version string used when expressing a dependency is undefined.

<table>
<thead>
<tr>
<th>Dependency Name</th>
<th>Specification Defined In</th>
<th>Version Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>microedition.io.file.FileConnection</td>
<td>[JSR75], PDA Optional Packages for the J2ME™ Platform</td>
<td>[JPVS]</td>
</tr>
<tr>
<td>microedition.pim</td>
<td>[JSR75], PDA Optional Packages for the J2ME™ Platform</td>
<td>[JPVS]</td>
</tr>
<tr>
<td>bluetooth.api</td>
<td>[JSR82], Java™ APIs for Bluetooth</td>
<td>[JPVS]</td>
</tr>
<tr>
<td>obex.api</td>
<td>[JSR82], Java™ APIs for Bluetooth</td>
<td>[JPVS]</td>
</tr>
<tr>
<td>speech.api</td>
<td>[JSR113], Java™ Speech API 2.0</td>
<td>Undefined</td>
</tr>
<tr>
<td>xml.jaxpsubset</td>
<td>[JSR172], J2ME™ Web Services Specification</td>
<td>Undefined</td>
</tr>
<tr>
<td>xml.rpcsubset</td>
<td>[JSR172], J2ME™ Web Services Specification</td>
<td>[JPVS]</td>
</tr>
<tr>
<td>microedition.satsa.apdu</td>
<td>[JSR177], Security and Trust Services API for J2ME™</td>
<td>Undefined</td>
</tr>
<tr>
<td>microedition.satsa.crypto</td>
<td>[JSR177], Security and Trust Services API for J2ME™</td>
<td>[JPVS]</td>
</tr>
<tr>
<td>microedition.satsa.jcrmi</td>
<td>[JSR177], Security and Trust Services API for J2ME™</td>
<td>[JPVS]</td>
</tr>
<tr>
<td>microedition.satsa.pki</td>
<td>[JSR177], Security and Trust Services API for J2ME™</td>
<td>[JPVS]</td>
</tr>
<tr>
<td>microedition.location</td>
<td>[JSR179], Location API for J2ME™ and [JSR293], Location API 2.0</td>
<td>[JPVS]</td>
</tr>
<tr>
<td>microedition.sip</td>
<td>[JSR180], SIP API for J2ME™</td>
<td>[JPVS]</td>
</tr>
<tr>
<td>microedition.m3g</td>
<td>[JSR184], Mobile 3D Graphics API for J2ME™</td>
<td>[JPVS]</td>
</tr>
<tr>
<td>wireless.messaging</td>
<td>[JSR205], Wireless Messaging API 2.0</td>
<td>Undefined</td>
</tr>
<tr>
<td>microedition.chapi</td>
<td>[JSR211], Content Handler API</td>
<td>[JPVS]</td>
</tr>
<tr>
<td>microedition.m2g</td>
<td>[JSR226], Scalable 2D Vector Graphics API for J2ME™</td>
<td>[JPVS]</td>
</tr>
<tr>
<td>microedition.amms</td>
<td>[JSR234], Advanced Multimedia Supplements</td>
<td>[JPVS]</td>
</tr>
<tr>
<td>microedition.global</td>
<td>[JSR238], Mobile Internationalization API</td>
<td>[JPVS]</td>
</tr>
<tr>
<td>microedition.khronos.opengles</td>
<td>[JSR239], Java™ Binding for the OpenGL® ES API</td>
<td>Undefined</td>
</tr>
</tbody>
</table>
When the dependency attribute contains a wildcard for the JSR version, an application can retrieve the exact version of the JSR that the MIDlet suite is bound to by calling System.getProperty(dependencyName + ".version"), where dependencyName is one of the dependency names found in Table 2-3 above. The same applies to proprietary APIs, provided that they define a system property with a name like proprietaryAPIName + ".version", where proprietaryAPIName is valid for the API as documented by its specification.

### 2.6.7 Informational Attributes

#### Table 2-4 : Informational Attributes for LIBlets

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBlet-Permissions-</td>
<td>Attribute contains 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. The permissions are critical to the function of the MIDlet suite.</td>
</tr>
<tr>
<td>LIBlet-Permissions-Opt-</td>
<td>Attribute contains 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. The permissions are not critical to the function of the MIDlet and it will operate correctly without them.</td>
</tr>
<tr>
<td>LIBlet-Permission-&lt;n&gt;</td>
<td>A fully qualified permission class, resource name, and actions that are critical to the function of the LIBlet.</td>
</tr>
<tr>
<td>LIBlet-Permission-Opt-&lt;n&gt;</td>
<td>A fully qualified permission class, resource name, and actions that are not-critical to the function of the LIBlet.</td>
</tr>
</tbody>
</table>
MIDP 3.0 Provisioning

3.1 Overview and Goals

This chapter describes how MIDlet suites and LIBlets are provisioned; that is, how MIDlet suites and LIBlets are deployed to client devices, and the requirements imposed upon a client device to support these deployments. Following these recommendations will help ensure interoperability between clients and servers from all manufacturers, and will also provide guidance to mobile network operators deploying MIDP 3.0 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 browser; in other cases, it could be an application written specifically to identify MIDlet suites for the user to download. Throughout this chapter, an application with this functionality will be referred to as the Discovery Application, or DA.

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

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

This chapter describes the general functional requirements of the device and the functions supporting the MIDlet suite lifecycle and LIBlet provisioning. Descriptions are included for additional Application Attributes and mechanisms that identify the device type and characteristics to servers providing MIDlet suites and LIBlets.

3.2 Functional Requirements

A MIDP 3.0 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 later as required in this chapter.
- Responding to a 401 (Unauthorized) or 407 (Proxy Authentication Required) response to an HTTP request by asking the user for a username 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 fundamental unit of transfer and installation.
3.3 MIDlet Suite Lifecycle

The lifecycle of a MIDlet suite consists of discovery, installation, update, invocation and deletion. The lifecycle of a LIBlet consists only of discovery, installation, update, and deletion.

3.3.1 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.
- The provisioning server can request that a full client device profile be sent to the server with the request for the Application Descriptor.
- 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 download begins.
- If the MIDlet suite depends on one or more LIBlets, the Application Descriptors for all dependency LIBlets are downloaded prior to the MIDlet suite JAR download. The decision on whether to proceed with the MIDlet suite JAR download (and subsequent installation) depends on the verification of the JAD and JAR of both the MIDlet suite and dependency LIBlets.

3.3.2 Provisioning

When a MIDlet suite is provisioned, all files that comprise the MIDlet are downloaded to the device for installation. In addition to the MIDlet suite JAR file, one or more LIBlet JAR files could be downloaded, along with RMS data.

This section describes how the Discovery Application (DA) and the Application Management System (AMS) handle the various components of the MIDlet suite.

3.3.2.1 MIDlet Suite Provisioning

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 as described in the MIDP specification, the JAR 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-descraptor.
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. This may involve downloading of Application Descriptors of any LIBlets the MIDlet suite depends on, if any LIBlet dependencies are specified by the MIDlet suite.

If the MIDlet suite cannot be successfully installed, 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
MIDP 3.0 Provisioning

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 Application Descriptor MUST be formatted according to the syntax defined in Application Attributes. All of the application attributes designated as mandatory by Appendix A MUST be present in the Application Descriptor; if this is not the case, then the client MUST return Status Code 906 (Invalid Descriptor) in the status report. If the Application Descriptor contains any application attributes that are allowed only in the Jar manifest (see Appendix A), then the client MUST return Status Code 906 (Invalid Descriptor) 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 SHOULD be notified of them. 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.

3.3.2.2 LIBlet Provisioning

LIBlet provisioning MUST be a transparent process from the perspective of the user, requiring no additional user interactions beyond those already required as part of MIDlet provisioning. LIBlet provisioning is performed as part of MIDlet provisioning.

The MIDlet suite JAD file lists its LIBlet dependencies. Each LIBlet may in turn have other LIBlet dependencies. The JAD files involved, for both MIDlet and LIBlets, MUST be downloaded first before any MIDlet suite or LIBlet JARs are downloaded. Based on the downloaded JAD files, the device SHOULD calculate the total static size required for installation. Also, any JAD only validation steps MAY be undertaken before the JARs are downloaded.

The user SHOULD be notified of the overall size of the download, inclusive of the MIDlet suite and all needed LIBlets. That is, if the MIDlet to be installed depends on LIBlets that are already present on the device, the size of those LIBlets MUST be excluded from the notification.

Figure 3-1: LIBlet Provisioning Overview (needs to be updated to include the repeat fetch of the JAD to supply UAPerf headers)
3.3.2.3 RMS Data Provisioning

RMS data can be provisioned along with a MIDP application as files that conform to the RMS interchange file format. The implementation MUST handle RMS data thus provisioned. The RMS data files can be delivered as standalone files or embedded in the application's JAR. If multiple RMS files are provisioned with the application, each containing a different record store, the implementation MUST initialize each of the record stores as described below.

RMS data files related to the application are identified in the application's JAD file using the MIDlet-Persistent-Data-URL-<n> attribute. The attribute value contains a URL that points to an RMS data file to be installed along with the application at the time of provisioning. For standalone files the implementation MUST support the HTTP download protocol. For RMS data files embedded in the application's JAR, the URL MUST be relative to the root of the JAR.

RMS data may also be provisioned along with a LIBlet in a similar manner as application-related RMS data using analogous attributes in the LIBlet JAD file. A shared record store owned by a LIBlet MUST be created by the AMS when a LIBlet declares a LIBlet-Persistent-Data-URL-<n> attribute in its JAD and/or Manifest. For a LIBlet, specifying this attribute is the only way to create a shared record store that is owned by the LIBlet, instead of the MIDlet suites it is bound to.

The authmode field of the RMS data file provisioned along with a LIBlet MUST be set to AUTHMODE_PRIVATE or the installation will fail. For LIBlet-provisioned RMS data, the AUTHMODE_PRIVATE setting means that these record stores can only be accessed by MIDlet Suites that have declared a dependency on the owning LIBlet.

If overwrite is specified in the MIDlet-Persistent-Data-URL-<n> attribute value, data from the RMS file MUST replace any existing record store of the same name in the device. If a record store with the same name exists in the device and overwrite is not specified, the data from the RMS file MUST be discarded. Note that LIBlet-Persistent-Data-URL-<n> does not have the overwrite option since any newly installed LIBlet will bring with it a unique record store namespace, unrelated to any other LIBlet that may exist on the device. If encryptLocally is
specified in the MIDlet-Persistent-Data-URL-<n> or LIBlet-Persistent-Data-URL-<n> attribute value, the record store MUST be encrypted locally.

For example, the following JAD attributes set the minimum data size and establish two record stores:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDlet-Data-Size</td>
<td>3000</td>
</tr>
<tr>
<td>MIDlet-Persistent-Data-URL-1</td>
<td>data.rms overwrite encryptLocally</td>
</tr>
<tr>
<td>MIDlet-Persistent-Data-URL-2</td>
<td><a href="http://vendor.com/gamedata/coolgame.rms">http://vendor.com/gamedata/coolgame.rms</a></td>
</tr>
</tbody>
</table>

The implementation MUST initialize the record store with the data obtained from the location provided in the MIDlet-Persistent-Data-URL-<n> and LIBlet-Persistent-Data-URL-<n> attributes. The installation of the application MUST fail with Status Code 913 (RMS Initialization failure) if any of the following conditions occur:

- The RMS data cannot be obtained from the specified URL
- The RMS data file does not conform to the RMS interchange file format specification; see RMS Interchange File Format for details.
- A record store with the same name id provisioned more than once.
- The authmode field of the RMS Data file for a LIBlet is not set to AUTHMODE_PRIVATE.

Each attribute MUST result in a unique record store name/attribute combination. Record store overwrites MUST NOT be allowed from within a single installation. Record store overwrites MUST account for provisioning atomicity, only replacing an existing record store once installation is known to have been successful.

For example, the following JAD attributes would result in a failed installation.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDlet-Persistent-Data-URL-1</td>
<td><a href="http://foo.com/data.rms">http://foo.com/data.rms</a></td>
</tr>
<tr>
<td>MIDlet-Persistent-Data-URL-2</td>
<td><a href="http://foo.com/data.rms">http://foo.com/data.rms</a> overwrite encryptLocally</td>
</tr>
</tbody>
</table>

As another example, if data1.rms and data2.rms both have the same value for Record Store Name field, the following installation would result in an RMS provisioning failure.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDlet-Persistent-Data-URL-1</td>
<td><a href="http://foo.com/data1.rms">http://foo.com/data1.rms</a></td>
</tr>
<tr>
<td>MIDlet-Persistent-Data-URL-2</td>
<td><a href="http://foo.com/data2.rms">http://foo.com/data2.rms</a></td>
</tr>
</tbody>
</table>

The record store data from an RMS data file MUST be initialized as specified in the documentation of the importRecordStore method of the javax.microedition.rms.RecordStore class. If the RMS data is encrypted, the AMS MUST prompt the user for the password to be used to decrypt the data. If the record store is to be encrypted locally, the AMS MUST prompt the user for the record store password.

### 3.3.3 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. With the exception of Persistent MIDlet Suites, the user MUST retain control of the resources used by MIDlet suites on the device and MUST be allowed to install or delete MIDlet suites.

The device MUST make the MIDlet(s) in the MIDlet suite available for invocation by the user. If the MIDlet suite contains more than one MIDlet, the user SHOULD be made aware of it. After a successful installation, the device MUST provide the user with an option to invoke a MIDlet from the downloaded MIDlet suite immediately.
MIDP 3.0 Provisioning

If the MIDlet suite depends on one or more LIBlets, implementation MAY make the user aware of all LIBlets the MIDlet suite depends on. The means by which MIDlet dependencies are presented to the user are implementation specific.

LIBlet installation is done as part of the MIDlet suite installation process. Since other applications could depend on other versions of a LIBlet already installed in the device, a LIBlet is not simply replaced with a newer version. Instead, another (but typically newer) version of the LIBlet is installed. Any existing version of the LIBlet MUST NOT be deleted if there are other MIDlets or LIBlets depending on it. It follows that multiple versions of the same LIBlet MAY reside on the device at the same time.

At installation of a signed MIDlet suite, the implementation MUST present to the user all security information related to the MIDlet suite, as guided in Presenting Security Related Information.

During installation, the user SHOULD be informed of progress and MUST be given an opportunity to cancel the process. Interrupting the 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 the Update section 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. Every download of an application descriptor by the AMS for installation or update MUST supply the headers needed by UAProf for Device Identification. If an Application Descriptor was first downloaded as described in the Discovery section either without the UAProf headers or if it is unknown whether the headers were sent and the descriptor includes the MIDlet-Profile-Request: true attribute then the download of the descriptor must be repeated using the URL of the descriptor and including the headers required by UAProf. The new descriptor supersedes the original descriptor in the processing that follows. The requests for downloading descriptors and JARs in 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 username 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:
   a. If there is insufficient memory to store the MIDlet suite on the device, the device MUST return Status Code 901 (Insufficient Memory) in the Status Report. In case a LIBlet dependency is specified, the device MUST first verify there is sufficient persistent storage space by summing the JAR size specified in the JADs of the MIDlet suite and all of the LIBlets that the MIDlet suite declares dependency on, directly or indirectly, using the following formula:

\[
\text{MIDlet-Jar-Size} + \sum \text{(LIBlet-Jar-Size for distinct LIBlets)}
\]

The associated JARs will only be downloaded if there is sufficient persistent storage space. If a LIBlet already exists on the device, its size will be omitted from this total size.

b. If the JAR is not available at the MIDlet-Jar-URL attribute in the descriptor, the device MUST return Status Code 907 (Invalid JAR) in the Status Report. Similarly, if a dependent
**LIBlet is not already installed on the device and the JAR is not available at the LIBlet-Jar-URL, the device MUST return Status Code 907 (Invalid JAR) in the Status Report.**

c. If the received JAR size does not match the size specified in the Application Descriptor, the device MUST return Status Code 904 (JAR size mismatch) in the Status Report.

d. If the manifest or any other file cannot be extracted from the JAR, the device MUST return Status Code 907 (Invalid JAR) in the Status Report.

e. If the JAR manifest is not in the correct syntax, or if any of the required application attributes are missing in the JAR manifest, or if the JAR manifest contains any application attributes that are allowed only in the JAD (see Appendix A), then the device MUST return Status Code 907 (Invalid JAR) in the Status Report.

f. 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 (Attribute mismatch) in the Status Report. Similarly, if the mandatory attributes in the LIBlet descriptor LIBlet-Name, LIBlet-Version, and LIBlet-Vendor do not match those in the LIBlet JAR manifest, the device MUST return Status Code 905 (Attribute mismatch) in the Status Report.

g. The value of each attribute that appears in both the Manifest and Application Descriptor MUST have the identical value, with the exception of MIDlet suites with the MicroEdition-Profile attribute equal to "MIDP-1.0", "MIDP-2.0", or "MIDP-2.1". If not identical, the device MUST return Status Code 905 (Attribute mismatch) in the Status Report. For MIDP 1.0, MIDP 2.0, and MIDP 2.1 MIDlet suites, refer to Attribute Overrides in Application Descriptor.

h. If a MIDlet Application Descriptor or Manifest contains any instances of LIBlet-Dependency-<n> attributes, or if any LIBlet Application Descriptor or Manifest associated with the MIDlet suite contains any instances of MIDlet-Dependency-<n> attributes, then the device MUST return Status Code 906 (Invalid Descriptor) in the Status Report.

i. At installation time, the device MUST first verify there is sufficient disk space for persistent data using the data size specified in the JADs of the MIDlet suite and the LIBlets, computed as:

\[
\text{MIDlet-Data-Size} + \sum (\text{LIBlet-NonShared-Data-Size} \text{ for distinct LIBlets}) + \\
\sum (\text{LIBlet-Shared-Data-Size} \text{ for distinct LIBlets if it is the first binding for the LIBlet on the device})
\]

The associated JARs will only be downloaded if there is sufficient persistent storage space. In case of insufficient memory, the device MUST return Status Code 901 (Insufficient Memory) in the Status Report.

j. If the application failed to be authenticated, the device MUST return Status Code 909 (Application authentication failure) in the Status Report.

k. If circular dependencies are detected in the LIBlets that a MIDlet suite depends on, the device MUST reject the installation of such a binding and MUST return Status Code 916 (Circular LIBlet dependency) in the Status Report.

l. If a device imposes an upper limit on the number of LIBlet dependencies allowed for a single MIDlet suite, and if that upper limit is exceeded, the device MUST reject the installation of the MIDlet suite and MUST return Status Code 918 (LIBlet dependencies limit exceeded) in the Status Report.

m. If the computed hash of downloaded LIBlet JAR does not match the hash from the attribute MIDlet-Dependency-Jar-SHA1-<n> or LIBlet-Dependency-Jar-SHA1-<n> of the dependent's MIDlet suite or LIBlet, or if the computed hash of downloaded LIBlet JAR does not match the hash specified in the LIBlet JAD attribute LIBlet-Jar-SHA1, then the implementation MUST reject the downloaded LIBlet and MUST return Status Code 914 (Application Integrity Failure) in the Status Report. If the
MIDlet suite installation cannot proceed because of faults with the dependency LIBlets, the user SHOULD be presented with a clear explanation of the cause of the installation failure.

n. If the dependent LIBlet specified in the dependency expression of a MIDlet suite/LIBlet matches the attribute values of name, vendor, and version of a LIBlet that already exists on the device, but the hash value specified in the existing LIBlet JAD attribute \texttt{LIBlet-Jar-SHA1} does not match the hash value specified in the \texttt{MIDlet-Dependency-Jar-SHA1-<n>} or \texttt{LIBlet-Dependency-Jar-SHA1-<n>} attribute, then the implementation MUST download the new LIBlet JAR specified by the \texttt{LIBlet-JAR-URL} attribute. Implementations MUST install the downloaded LIBlet JAR as a unique LIBlet if the specified hash matches the value of the \texttt{MIDlet-Dependency-Jar-SHA1-<n>} or \texttt{LIBlet-Dependency-Jar-SHA1-<n>} attribute.

o. If the application is not authorized for a permission as defined by the Requesting Permissions for a MIDlet Suites section then installation must fail and the device MUST return Status Code 910 (Application authorization failure) in the Status Report. The JARs of the MIDlet suite and the LIBlets it depends on MUST NOT be downloaded.

p. If a static push registration fails for a reason other than not being authorized, the device MUST return Status Code 911 (Push registration failure) in the Status Report.

q. If the network service is lost during installation, Status Code 903 (Loss of Service) SHOULD be used in a Status Report if possible (it may be impossible to deliver the status report due to the network-service outage).

r. If the device detects a namespace collision in the MIDlet suite and the LIBlets it depends on, the device MUST fail the installation of such a binding and return Status Code 917 (LIBlet namespace collision) in the Status Report.

s. If either of the \texttt{MIDlet-Minimum-Canvas-Size} or \texttt{MIDlet-Maximum-Canvas-Size} attributes are specified, and the primary display of the device is incapable of meeting the specified size requirements, the device MUST fail the installation and return Status Code 905 (Attribute mismatch) in the Status Report. If \texttt{MIDlet-Maximum-Canvas-Size} is specified, its dimensions must be equal to or smaller than those of \texttt{MIDlet-Minimum-Canvas-Size} (if specified); if this condition is not met, the device MUST fail the installation and return Status Code 905 (Attribute mismatch) in the Status Report.

t. If installation of a valid MIDlet suite fails due to an unexpected internal error in the device, or if installation of a MIDlet suite fails due an error that cannot be reflected by one of the defined Status Codes, then the device MUST fail the installation and return Status Code 919 (General failure) in the Status Report.

u. If the user is presented with a security prompt at MIDlet installation time and the user denies the action, such case SHOULD be treated as if the MIDlet is not authorized for a permission. In this case the installation SHOULD fail and the device SHOULD return Status Code 910 (Application authorization failure) in the Status Report. The JARs of the MIDlet suite and the LIBlets it depends on SHOULD NOT be downloaded.

4. If any \texttt{MIDlet-Persistent-Data-URL-<n>} or \texttt{LIBlet-Persistent-Data-URL-<n>} attributes are specified, the associated RMS data file(s) are obtained from these locations, and the RMS record stores are created and initialized with corresponding data. If the RMS data files cannot be obtained or the record stores cannot be created, as defined in RMS Data Provisioning the installation MUST return Status Code 913 (RMS Initialization failure) in the Status Report.

5. If the \texttt{MIDlet-Required-IP-Version} attribute is specified, and that IP version is not supported by the implementation, the installation MUST fail and return Status Code 905 (Attribute mismatch) in the Status Report.

6. If the MIDlet is protected by OMA DRM and the device can not parse it, then Status Code 953 (Non-Acceptable Content) MUST be sent as described in [OMADMIDP].
7. 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.

8. The 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 the Status Reports section.

### 3.3.4 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 the software are installed. See Device Identification and Request Headers for the attributes that apply to updates.

If a MIDlet suite update does not result in binding to the same security domain as the original MIDlet Suite, then installation MUST fail and the device MUST return Status Code 910 (Application authorization failure) in the Status Report. Additionally, if a MIDlet Suite update and the already installed MIDlet Suite do not have at least one common signer, then installation MUST fail and the device MUST return Status Code 910 (Application authorization failure) in the Status Report. Common signer is defined as matching the Organization field within the Subject field of the signing certificate of MIDlet Suite update and the signing certificate of the original MIDlet Suite, where the signing certificates are validated against the same Protection Domain Root Certificate.

If the provisioned MIDlet suite has the MIDlet-Update-URL attribute in the JAD file or the JAR manifest, the implementation MUST use it to configure the automatic update feature. The value of this attribute MUST either be empty or contain a valid URL. For details, see the description of the MIDlet-Update-URL attribute in Application Attributes.

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.

If some MIDlet from the MIDlet suite being updated is running when the update is started, the implementation SHOULD prompt the user to shut down the running MIDlet so that it can be updated. After the MIDlet suite has been updated, the implementation SHOULD offer to start again the MIDlet that was running when the update of the suite was started. Note that these actions do not apply for autostart MIDlets, which involve no user interaction.

If the MIDlet suite dependencies change, the developer of the MIDlet suite needs to provide an updated JAD file containing the new LIBlet dependencies to be provisioned to the device. These dependencies MUST be resolved like in the initial installation of the MIDlet suite.

When updating a dependency declaration chain, the MIDlet suite declaring the dependency MUST increase its own version number, regardless of whether or not any code changes occurred to the MIDlet suite or LIBlet itself declaring the dependency. Additionally, for all LIBlets in a dependency chain, whenever a LIBlet version changes due to code change and/or dependency declaration change, the attribute LIBlet-Dependency-Jar-SHA1-<n> MUST contain the new hash value of the updated LIBlet JAR.

For the case where a MIDlet suite is being updated only because of new LIBlet dependencies, the MIDlet suite developer will need to check that the required Permissions for new LIBlet dependencies are reflected in MIDlet manifest and/or JAD. The MIDlet suite will also need to be correctly signed.
For a signed MIDlet suite update, the RMS record stores MUST be retained. For an unsigned MIDlet suite update, the RMS record stores MUST be managed as follows:

- 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 none of the conditions above are met, 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 the case where the RMS data of the original MIDlet suite is retained and additional RMS data is provisioned via the `MIDlet-Persistent-Data-URL-<n>` attribute, the update MUST be handled as follows: the new record store data will replace the RecordStore if the `overwrite` attribute is set; otherwise the original record is unmodified and the new record data is discarded.

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.

3.3.5 Invocation

When the user selects a MIDlet to be run, the device MUST invoke the MIDlet with the MIDP Execution Environment. If multiple MIDlets are present, the AMS user interface MUST allow the user to select each one individually for invocation. If a MIDlet is already running in the MIDP execution environment, the AMS user interface MUST indicate this to the user. In addition, the AMS user interface MUST be able to bring the already running MIDlet to the foreground (if the MIDlet has a Displayable that has been bound to a display).

In case of user-initiated 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 found in the JAD file or JAR manifest), the 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 MIDlet 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.

3.3.6 Deletion

3.3.6.1 MIDlet Suite Deletion

Devices MUST allow users to delete MIDlet suites, with the exception of Persistent MIDlets (see MIDlet Package Overview for details). When a MIDlet suite is to be deleted from the device, the user SHOULD be prompted to confirm that the MIDlet suite can be deleted. 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. The device SHOULD warn the user of any other special circumstances that arise during the deletion of the MIDlet suite.

If the Application Descriptor includes the `MIDlet-Delete-Confirm` attribute, 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 deleted.
3.3.6.2 LIBlet Deletion

LIBlet deletion is usually done as part of the MIDlet suite deletion process, if a LIBlet has no other MIDlet suites depending on it. An implementation that has sufficient persistent storage MAY choose to retain LIBlets for possible later use, even after all MIDlet suites depending on the LIBlet have been deleted.

A LIBlet becomes unreferenced if there are no MIDlet suites installed on the device that specify a dependency on that particular LIBlet. It is up to the implementation to determine when (if ever) it is appropriate to delete unreferenced LIBlets. For implementations that do retain unreferenced LIBlets, the AMS MAY provide a means to delete such LIBlets.

The user MUST NOT be able to delete individual LIBlets so long as there exist any installed dependent MIDlet suites on the device. However, the AMS SHOULD provide the user the ability to delete unreferenced LIBlets. Some implementations MAY, according to policy, never allow unreferenced LIBlet deletion, either automatically or by the user, even if the LIBet's reference count drops to zero.

The AMS MAY choose to display MIDlet suite dependency information to the user. The AMS MAY also provide a way to present the LIBlets on the system, their size and usage information to the user. If the AMS provides a means for a user to determine the MIDlet suite size, it MAY in addition provide information about its dependent components. For example, it MAY provide information that resembles the following:

| Name: Ringtones App                  |
| Size: 100KB                          |
| Size of shared components not used by other apps: 1MB |
| Size of shared components used also by other apps: 4MB |

This enables the user to determine the space that would be freed up if the MIDlet suite was deleted.

3.4 Status Reports

The success or failure of the installation, update, or deletion of a MIDlet suite is of interest to the service providing the MIDlet suite. The Application Descriptor MAY include relevant application attributes for provisioning that specify URLs for status reporting. Those URLs MUST be used for reporting as specified below.

The operation status is reported by means of an HTTP POST request to the relevant URL specified. The server is expected to respond with a valid HTTP response. For an example of a status report, see Example: Install Status via HTTP Post Request. Status reports are only sent for MIDlet suites downloaded Over The Air (OTA).

The body of the POST request MUST include a status code and status message on its first line. See Provisioning Status Codes and Messages for a list of valid status codes and suggested status messages. The messages are informational only.

The implementation MUST transcode the request body into UTF-8, and MUST set the Content-Type entity-header field of the request as follows:

```
Content-Type: text/plain; charset=utf-8
```

If the server receives the status report successfully, it MUST reply with a 2xx response code (typically 200 OK) and SHOULD NOT include a message body. Any content in the message body MUST be ignored by the implementation.
Since the body of the status report is short and simple, the implementation SHOULD NOT set the Expect: 100-continue header in the request, and MAY ignore any 100 Continue response sent by the server.

For other typical server responses, the implementation MUST maintain a retry count and attempt to retry the sending of the report as follows:

- If the server response is 301 Moved Permanently, 302 Found, 303 See Other or 307 Temporary Redirect, retry the request using the URI supplied in the response.
- If the server response is 4xx, increase the retry count by one. The request SHOULD be examined for errors before retrying.
- If the server response is 5xx, increase the retry count by one, and defer the retrying by an implementation-defined amount of time.

Unless an acknowledgement is requested with a MIDlet-Install-Notify attribute whose value includes the ack tag, the sending of any status report MUST be retried until the retry count reaches at least five (5), or it can be determined that there is no server response (for example, due to the loss of network connectivity or a network timeout). The implementation MAY continue to retry the sending of the report, but SHOULD limit the number of attempts, as each one could result in data costs to the user.

The installed MIDlet suite MUST be made available for use even if the sending of the installation status report fails, unless an acknowledgement is requested with the ack tag in the MIDlet-Install-Notify attribute. If the ack tag is present, the installation of the MIDlet suite MUST fail if a server response other than 2xx or 3xx is received. When a MIDlet suite is updated, any ack tag SHOULD be ignored, and the updated MIDlet suite be made available immediately.

If the MIDlet suite being installed has dependencies on some LIBlets, an installation status report MUST be sent to any LIBlet-Install-Notify URL specified by the LIBlet vendor.

For deletion status reports, the notification is sent when a MIDlet suite or LIBlet is deleted; Status Code 912 (Deletion Notification) MUST be sent, notifying that the deletion occurred. The attempt to send the deletion status report SHOULD 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.

The deletion of a MIDlet suite or LIBlet MUST be completed even if the sending of the deletion status report fails. Note that LIBlets are not deleted along with the MIDlet suite if there are other MIDlet suites that depend on them.

The following table contains the provisioning status codes and suggested status messages.

<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>
</tbody>
</table>
### Incompatible Configuration or Profile

### Application authentication failure

### Application authorization failure

### Push registration failure

### Deletion Notification

### RMS Initialization failure

### Application Integrity Failure

### One or more missing LIBlets

### Circular LIBlet dependency

### LIBlet namespace collision

### LIBlet dependencies limit exceeded

### General failure

### Non-Acceptable Content

### 3.5 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.

#### 3.5.1 UAProf for Device Identification

Provisioning servers require detailed information about the configuration of the device hardware, software, and installed middleware. However, the Discovery Agent, normally a web browser is not expected to provide detailed information because it does not have access to the detailed configuration of the Java Runtime.

The Open Mobile Alliance (OMA) has developed the UAProf Specification to communicate device and software configuration. See OMA User Agent Profile V2.0 for detailed specifications and schemas. UAProf is based on the Composite Capability/preference Profiles (CC/PP). The configuration schemas are extensible and the processing of profile instances is defined to allow static and difference profile instances to be merged to yield a complete description. The static configuration information is presented to the network using the URL of a profile that describes the device including hardware details and the software provisioned with the device. The client device can supply both the URL of a profile (by reference) and a difference profile (by value in a XML stream) in http headers using the UAProf specification.

The difference profile includes information that has changed relative to the static configuration due to installation of software, changes in the SIM card, user settings, etc.

The UAProf specification defines the additional HTTP headers required to deliver configuration information to the server. The headers include `x-wap-profile`, and `x-wap-profile-diff` headers. These HTTP Request Headers MUST be supplied on the initial request by the AMS to retrieve a JAD.
3.5.2 MIDP Configuration Profile for UAProf

The predefined configuration schema is not adequate to describe the features of the MIDP runtime and the optional packages and shared libraries. The additional schema defined below addresses configuration of the MIDP Execution environment. The MIDP Configuration Profile builds on the established mobile profile. The MIDP specific RDF schema elements are defined below and are used incrementally as extensions to the existing schema.

The RDF elements are described Table 3-4 below.

<table>
<thead>
<tr>
<th>RDF Element</th>
<th>Domain of Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDSharedLibrary</td>
<td>MIDProfile</td>
<td>A MIDP Shared Library property contains values for Name, Vendor, Version and JAR hash. The LIBlet Name, Vendor and Version are defined by LIBlet attributes.</td>
</tr>
<tr>
<td>SecurityDomain</td>
<td>MIDProfile</td>
<td>SecurityDomain provides values to describe the domain name, certificate subject and certificate hash (Base64 encoded) for each domain.</td>
</tr>
<tr>
<td>Certificate</td>
<td>MIDProfile</td>
<td>For each certificate used for application authentication, the subject name as the key and the hash of the public key (Base64 encoded).</td>
</tr>
<tr>
<td>SystemProperty</td>
<td>MIDProfile</td>
<td>An RDF Bag containing, for each system property the name and the value.</td>
</tr>
<tr>
<td>PushProtocols</td>
<td>MIDProfile</td>
<td>An RDF Bag containing each of protocols supported for Push.</td>
</tr>
</tbody>
</table>

The RDF schema for the MIDP specific elements is:

```xml
<rdf:RDF
   xmlns      = '&ns-rdf;'
   xmlns:rdf  = '&ns-rdf;'
   xmlns:rdfs = '&ns-rdfs;'
   xmlns:prf  = '&ns-prf;'
   xmlns:xsd  = 'http://www.w3.org/2001/XMLSchema#'
   xmlns:midf = 'http://opensource.motorola.com/midp3/midprofile#'>
   <rdf:Description rdf:ID='MIDProfile'>
     <rdf:type rdf:resource='&ns-rdfs;Class'/>
     <rdfs:subClassOf rdf:resource='&ns-prf;Component'/>
     <rdfs:label>MIDProfile</rdfs:label>
     <rdfs:comment>description: A platform based on MIDP.</rdfs:comment>
   </rdf:Description>
   <rdf:Description rdf:ID='MIDSharedLibrary'>
     <rdfs:label>MIDSharedLibrary</rdfs:label>
     <rdfs:comment>MIDP Shared Library property contains values for Name, Vendor, Version and JAR hash. The LIBlet Name, Vendor and Version are defined by LIBlet attributes.</rdfs:comment>
   </rdf:Description>
   <rdf:Description rdf:ID='SecurityDomain'>
     <rdfs:label>SecurityDomain</rdfs:label>
     <rdfs:comment>SecurityDomain provides values to describe the domain name, certificate subject and certificate hash (Base64 encoded) for each domain.</rdfs:comment>
   </rdf:Description>
   <rdf:Description rdf:ID='Certificate'>
     <rdfs:label>Certificate</rdfs:label>
     <rdfs:comment>For each certificate used for application authentication, the subject name as the key and the hash of the public key (Base64 encoded).</rdfs:comment>
   </rdf:Description>
   <rdf:Description rdf:ID='SystemProperty'>
     <rdfs:label>SystemProperty</rdfs:label>
     <rdfs:comment>An RDF Bag containing, for each system property the name and the value.</rdfs:comment>
   </rdf:Description>
   <rdf:Description rdf:ID='PushProtocols'>
     <rdfs:label>PushProtocols</rdfs:label>
     <rdfs:comment>An RDF Bag containing each of protocols supported for Push.</rdfs:comment>
   </rdf:Description>
</rdf:RDF>
```
MIDP 3.0 Provisioning

<rdf:type rdf:resource='&ns-rdf;Property'/>
<rdf:type rdf:resource='&ns-rdf;Bag'/>
<rdfs:domain rdf:resource='#MIDProfile'/>
<rdfs:label>MIDShareLibrary</rdfs:label>
<rdfs:comment>
Description: A MIDP Shared Library described by name, vendor, version,
and JAR hash. The values are separated by ";" (semi-colon).
Type: Literal (Bag) String with multiple fields.
Resolution: Append
Examples: "QSort; BigCo; 1.1"
</rdfs:comment>
</rdf:Description>

<rdf:Description rdf:ID='MIDSystemProperty'>
<rdf:type rdf:resource='&ns-rdf;Property'/>
<rdf:type rdf:resource='&ns-rdf;Bag'/>
<rdfs:domain rdf:resource='#MIDProfile'/>
<rdfs:label>MIDSystemProperty</rdfs:label>
<rdfs:comment>
Description: A System property defined by the platform.
Type: Literal (Bag) String String
Resolution: Append
Examples: "microedition.locale fr-FR", "microedition.profiles MIDP-3.0"
</rdfs:comment>
</rdf:Description>

<rdf:Description rdf:ID='SecurityDomain'>
<rdf:type rdf:resource='&ns-rdf;Property'/>
<rdf:type rdf:resource='&ns-rdf;Bag'/>
<rdfs:domain rdf:resource='#MIDProfile'/>
<rdfs:label>SecurityDomain</rdfs:label>
<rdfs:comment>
Description: A security domain is a property with values for
Domain Name, Certificate Subject and Certificate hash.
For each root certificate, a separate entry is required.
Type: Literal (Bag) String String String
Resolution: Append
Examples: "Operator XYZCorp,Inc. a23438fd8e8a8a8a8a8aaaaaaa"
</rdfs:comment>
</rdf:Description>

<rdf:Description rdf:ID='Certificate'>
<rdf:type rdf:resource='&ns-rdf;Property'/>
<rdf:type rdf:resource='&ns-rdf;Bag'/>
<rdfs:domain rdf:resource='#MIDProfile'/>
<rdfs:label>Certificate</rdfs:label>
<rdfs:comment>
Description: A certificate identified by an alias and its hash
used for access authorization.
For each certificate, a separate entry is required.
Type: Literal (Bag) String String
Resolution: Append
Examples: "XYZCorp a23248fd8e8a8a929a8aaaaa"
</rdfs:comment>
</rdf:Description>

<rdf:Description rdf:ID='PushProtocols'>
<rdf:type rdf:resource='&ns-rdf;Property'/>
<rdf:type rdf:resource='&ns-rdf;Bag'/>
<rdfs:domain rdf:resource='#MIDProfile'/>
<rdfs:label>PushProtocols</rdfs:label>
<rdfs:comment>
</rdfs:comment>
</rdf:Description>
MIDP 3.0 Provisioning

<rdf:Description rdf:ID="Profile">
  <prf:component>
    <rdf:Description rdf:ID="MIDProfile">
      <!-- System properties combined from all APIs -->
      <midf:SystemProperty>
        <rdf:Bag>
          <rdf:li>microedition.media.version 1.2</rdf:li>
          <rdf:li>audio.encodings encoding=pcm rate=11025&bits=16&channels=1</rdf:li>
          <rdf:li>microedition.deviceid imei:490154203237518</rdf:li>
          <rdf:li>microedition.subscriberid imsi:310150123456789</rdf:li>
          <rdf:li>microedition.profiles MIDP-3.0</rdf:li>
        </rdf:Bag>
      </midf:SystemProperty>
      <!-- Shared Libraries present on the device. -->
      <midf:MIDSharedLibrary>
        <rdf:Bag>
          <rdf:li>QSort MWV 1.1 hash8</rdf:li>
        </rdf:Bag>
      </midf:MIDSharedLibrary>
      <!-- Supported Push Protocols -->
      <midf:PushProtocols>
        <rdf:Bag>
          <rdf:li>sms</rdf:li>
          <rdf:li>socket</rdf:li>
          <rdf:li>datagram</rdf:li>
        </rdf:Bag>
      </midf:PushProtocols>
      <!-- Security Domains and their root certificates -->
      <midf:SecurityDomain>
        <rdf:Bag>
          <rdf:li>Operator Alias1 hash1</rdf:li>
          <rdf:li>Manufacturer Alias2 hash2</rdf:li>
          <rdf:li>Identified Alias3 hash3</rdf:li>
          <rdf:li>Identified Alias4 hash4</rdf:li>
        </rdf:Bag>
      </midf:SecurityDomain>
      <!-- Certificates used for access authorization -->
      <midf:Certificates>
      </midf:Certificates>
    </rdf:Description>
  </prf:component>
</rdf:Description>

A partial example might look like:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:prf="http://www.openmobilealliance.org/tech/profiles/UAPROF/ccpschema-20021212#"
  <rdf:Description rdf:ID="Profile">
    <prf:component>
      <rdf:Description rdf:ID="MIDProfile">
        <!-- System properties combined from all APIs -->
        <midf:SystemProperty>
          <rdf:Bag>
            <rdf:li>microedition.media.version 1.2</rdf:li>
            <rdf:li>audio.encodings encoding=pcm rate=11025&bits=16&channels=1</rdf:li>
            <rdf:li>microedition.deviceid imei:490154203237518</rdf:li>
            <rdf:li>microedition.subscriberid imsi:310150123456789</rdf:li>
            <rdf:li>microedition.profiles MIDP-3.0</rdf:li>
          </rdf:Bag>
        </midf:SystemProperty>
        <!-- Shared Libraries present on the device. -->
        <midf:MIDSharedLibrary>
          <rdf:Bag>
            <rdf:li>QSort MWV 1.1 hash8</rdf:li>
          </rdf:Bag>
        </midf:MIDSharedLibrary>
        <!-- Supported Push Protocols -->
        <midf:PushProtocols>
          <rdf:Bag>
            <rdf:li>sms</rdf:li>
            <rdf:li>socket</rdf:li>
            <rdf:li>datagram</rdf:li>
          </rdf:Bag>
        </midf:PushProtocols>
        <!-- Security Domains and their root certificates -->
        <midf:SecurityDomain>
          <rdf:Bag>
            <rdf:li>Operator Alias1 hash1</rdf:li>
            <rdf:li>Manufacturer Alias2 hash2</rdf:li>
            <rdf:li>Identified Alias3 hash3</rdf:li>
            <rdf:li>Identified Alias4 hash4</rdf:li>
          </rdf:Bag>
        </midf:SecurityDomain>
        <!-- Certificates used for access authorization -->
        <midf:Certificates>
        </midf:Certificates>
      </rdf:Description>
    </prf:component>
  </rdf:Description>
</rdf:RDF>
```
3.5.3 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 in the Networking portion of this 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.

3.5.4 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.

3.5.5 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.

3.5.6 Example: HTTP Request for Application Descriptor

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

```
GET http://host.foo.bar/app-dir/game.jad HTTP/1.1
Host: host.foo.bar
Accept: text/vnd.sun.j2me.app-descriptor
User-Agent: CoolPhone/1.4 Profile/MIDP-3.0 Configuration/CLDC-1.1
Accept-Language: en-US, fi, fr
Accept-Charset: utf-8
```

The response headers from the server might look as follows:

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

3.5.7 Example: HTTP Request to Install/Update a MIDlet Suite

When requesting the download of a MIDlet suite JAR, the request headers might look as follows:

```
GET http://host.foo.bar/app-dir/game.jar 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: 25432
Content-Type: application/java-archive
```

3.5.8 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-Type: text/plain; charset=utf8
Content-Length: 13
900 Success
```

The response from the server might be:

```
HTTP/1.1 200 OK
Server: CoolServer/1.3.12
```
MIDP 3.0 Platform

Historically, the MIDP Specification was designed to be used in conjunction with the Java ME Connected Limited Device Configuration (CLDC). For MIDP 1.0, the only available underlying Java ME configuration was CLDC 1.0, whereas the MIDP 2.0 and 2.1 Specifications were designed to be used with CLDC 1.0 and 1.1. MIDP 3.0 extends this architecture by requiring CLDC 1.1 as the minimum configuration, while also allowing for implementations to be based on CDC 1.1 (Connected Device Configuration).

Because device capabilities have increased significantly in the past few years, it has become feasible to run MIDlets not only on top of the CLDC configuration, but also on the more capable CDC configuration. Many device manufacturers are already building mobile information devices for which the underlying Java ME configuration is CDC. This has created a need to specify the behavior of MIDP 3.0 APIs and functions for execution environments above and beyond CLDC 1.1, including the Connected Device Configuration (CDC).

4.1 General Guidelines for MIDlet Behavior on Different Execution Environments

The MIDP application model, including packaging, provisioning, lifecycle, and APIs, MUST be uniform regardless of the underlying execution environment. The essential aspects of the MIDP application model, such as application packaging (including the definition of JAD, JAR and manifest formats), provisioning semantics, installation and removal, and the overall application lifecycle MUST remain constant.

Implementation independence. The behavior of MIDP 3.0 APIs and functions MUST be independent of the implementation details of the underlying configuration or platform. The MIDlet itself specifies the execution environment that it requires. The specification of the MIDP execution environment is defined in the package documentation for javax.microedition.midlet.MIDlet.

4.2 Developer Recommendations for Dealing With Overlapping Configuration APIs

For historical reasons, the MIDP Specification contains some APIs that overlap with the functionality offered by the CDC Specification:

1) Overlapping network library choices. In CLDC-based systems, the only way to access network libraries is to use the Generic Connection Framework (GCF). In CDC, the preferred way is to use the java.net.* libraries.

2) Overlapping file system API choices. In CLDC based systems, the only way to access the file system of a device is via the JSR 75 FileConnection interface. In CDC, the preferred way is to use the File* classes provided in package java.io.

3) Overlapping localization API choices. In CLDC-based systems, the only way to support certain internationalization and localization features such as formatting of numbers, percentages and dates
MIDP 3.0 Platform

is via the JSR 238 Mobile Internationalization API. In CDC, the preferred way is to use the facilities provided in package `java.text`.

**Developer Recommendation:** As a general guideline and recommendation, developers should continue to use those APIs included in the Mobile Service Architecture (MSA) specification that are in widespread use in MIDlet development today. For example, to access networking functionality, the use of the Generic Connection Framework is recommended. Similarly, to improve portability of applications, the file system should be accessed primarily via the JSR 75 `FileConnection` interface.

Developers who have a compelling need for CDC APIs should be aware that the APIs they choose will limit the market for their applications to those MIDP 3.0 devices that support the CDC APIs. Continuing to limit their API choices to CLDC 1.1.1 and MIDP 3.0 APIs ensures that applications will run on the maximum number of devices.

### 4.3 Platform Default Character Encoding

The CLDC specification defines the `microedition.encoding` system property, used to determine the character encoding to be used when none is explicitly specified in various input stream and string operations that depend on interpreting bytes as characters. This is known as the "platform default character encoding". The CLDC specification defines "ISO-8859-1" as the default value. This value has been used in many implementations, but it has proven to be too restrictive for real-world use.

In MIDP 3.0, the platform default character encoding is uniformly defined to be UTF-8 (see [RFC2279]). Therefore, MIDP 3.0 implementations MUST redefine the value of the `microedition.encoding` system property to be the string "UTF-8" always. UTF-8 was determined to be the most feasible character encoding that covers all the required characters for international applications.

Support for UTF-8 has been mandatory in MIDP since version 2.0. The change in platform default character encoding affects the following API methods and classes:

- `java.lang.String(byte[] bytes)` - constructor that takes an array of bytes and interprets it in the platform default encoding
- `java.io.InputStreamReader(InputStream is)` - constructor that uses the platform default encoding to interpret the data that is read
- `java.io.OutputStreamWriter(OutputStream os)` - constructor that uses the platform default encoding to interpret the data that is written
- `java.io.PrintStream` - to print representations of data values
- `java.io.ByteArrayOutputStream` - using the `toString` method to translate bytes into characters

In MIDP 3.0 implementations, these APIs continue to work based on the value of the `microedition.encoding`, but the end result is determined by the UTF-8 encoding. Method variants that take an explicit character encoding are not affected, so that a particular character encoding can still be requested by the application, provided that it is supported by the implementation.

For MIDP 2.0 and MIDP 1.0 applications (identified as such by the value of the `MicroEdition-Profile` attribute), the value of the `microedition.encoding` system property MUST continue to be whatever the implementation used earlier, as it varies between existing implementations. Accordingly, the APIs mentioned above MUST work as determined by the system property values. This guarantees backward compatibility for existing applications.
MIDP 3.0 MIDlet Concurrency

5.1 Overview

As mobile devices become more capable and MIDlets are written to tackle a wider array of functionality, there is an increasing need for the ability to run MIDlets concurrently. The MIDP 1.0 and 2.0 specifications did not prohibit running MIDlets concurrently. However, they were not explicit in detailing the expected behavior of MIDlets running concurrently. These previous specifications also did not provide the necessary mechanisms to allow MIDlets to detect and handle state changes.

Devices implementing the MIDP 3.0 specification MUST support the concurrent execution of two or more MIDlets. The MIDP 3.0 specification will define the expected behaviors of basic concurrency issues so that implementations behave as consistently as possible without dictating implementation decisions.

5.2 Runtime Behavior Of Concurrent MIDlets

The primary usability rule for defining concurrency behavior is that the MIDlets in question should behave as though they were running independently. In order to achieve this, data sets between MIDlet should be isolated, resource contention should be handled as transparently as possible, non-fatal errors should be quietly handled, and scheduling should be take into account the overall user experience.

There are many concurrency use cases which can be solved by implementing a solution based on stopping the execution of an application while a different application is running. In cases where the application in question is one which does no processing while the user is not interacting with it, this approach mimics the behavior of having concurrency. For instance, in the case of a game application, having the application stop or pause when switched away from is the desired behavior. The same might be true for a simple PIM application.

However, there are certain other use cases in which the desired behavior is for the application to continue running while not in focus. For instance, a media player application might wish to continue playing music while some other application is in focus.

A platform which is capable of supporting the behavior where an application not in focus runs simultaneously with the one in focus is also capable of supporting the model where the application not in focus is paused. However, the reverse is not true. Therefore, in order to standardize the general behavior of concurrency across platforms, MIDP 3.0 implementations must provide the ability to run applications simultaneously. It will be up to each application to use the mechanisms provided to determine its proper behavior when not in focus.
5.2.1 Data Isolation

In order to achieve an environment where each MIDlet behaves in the same manner regardless of what other MIDlets are running, it must be ensured that the data accessible by each application must be isolated from the data visible from any other application from a different scope.

Different MIDlets may also contain classes that have the same name but are different classes. Therefore the class space for each application context must also be separated. For instance, in Figure 5-11 below, Application 1 contains a class named “B”. Application 2 also has a class named “B”. Since Application 1 and Application 2 are in different contexts, they might refer to two completely different classes named “B”. Therefore, Application 1 would need to load the class it required into its class space while Application 2 would load a different version.

![Figure 5-1 : Data Isolation Between Concurrent MIDlets](image)

Every MIDlet, regardless of the value set in the microedition-profile attribute, MUST be launched within its own execution environment. Classes and static data MUST NOT be shared among MIDlets, even those within the same MIDlet suite. However, all other factors which currently make up the definition of a MIDlet suite such as permissions, RMS access, and resources remain per suite.

In this model, although the classes loaded by MIDlets in the same suite are identical because they are from the same JAR, the loading of the class and maintaining of the class state is handled on a per MIDlet basis, not per MIDlet suite. This includes the static variable data for all classes. For instance, if a system class A has a static variable x, changing the value of x in MIDlet 1 would not affect the value of x in MIDlet 2, even if the MIDlets were in the same suite. Data between MIDlet suites is maintained separately per suite.

![Figure 5-2 : Concurrent MIDlet Class and Data Separation](image)
Associating an execution environment to a MIDlet rather than a MIDlet suite is more consistent with the application models found in CDC and Java SE. This is highly desirable in order to prepare for the possible paths of future development of the MIDP specification.

5.2.2 Resource Contention
There is no specified resource contention policy between concurrently executing MIDlets. Implementations are free to implement any policy that best suits the platform. For instance, a platform could have a policy where a particular native resource, such as sound channels, are virtualized among running applications, or could instantiate a policy where the resources are allocated on a first-come-first-served basis.

5.2.3 Error Handling
When only running a single application, there is much more flexibility in the handling of errors than when running multiple applications simultaneously. In the case of a single MIDlet, most errors can be handled by the shutdown or re-initialization of the virtual machine. However, when multiple MIDlets are being executed simultaneously, an error affecting one MIDlet may not require that the second MIDlet be stopped. For instance, a NoClassDefFound Error may be fatal for the MIDlet involved but should not affect any other MIDlet which is also running.

5.2.4 Scheduling
The fine-grain details and algorithms used in determining thread and application scheduling will always be a platform-specific implementation detail. Due to the differences in implementations and platforms, it is impractical to mandate a specific policy for all devices. However, the individual policies used on all devices should share some similar characteristics.
MIDP 3.0 MIDlet Concurrency

MIDP 3.0 implementations SHOULD implement a scheduling policy which avoids starvation. No MIDlet running should ever be completely starved of execution time regardless of priority level or visibility.

Implementations should also attempt to ensure that MIDlets the user is currently interacting with are reasonably responsive. Assuming that the application currently in focus is the one that impacts the user the most, the policy should attempt to give threads of that MIDlet a higher priority and a proportionally higher percentage of processing resources than those of other MIDlets.

5.2.5 Execution Of More Than One Instance Of A MIDlet

The AMS MUST NOT launch a second instance of a MIDlet. If the AMS should receive a request to launch a MIDlet that is already running, either through user interaction or from an event, the AMS MUST instead post a relaunch system event to the MIDlet. The relaunch system event name is unique to the target MIDlet and is defined by APPLICATION_RELAUNCH_PREFIX. The event value MUST be true. MIDlets that intend to handle a second invocation request should register an EventDataListener for the concatenated system event name.

If the launch request is user-initiated from the AMS, the AMS MUST bring the requested MIDlet to the foreground if the MIDlet has a Displayable.
Security for MIDP Applications

6.1 Introduction

MIDP 3.0 builds on the domain based trust model introduced in MIDP 2.0 and adopts the class based permissions from the rest of the Java platform. The domain model is used to assign applications to protection domains.

MIDP 3.0 leverages the Permission classes used in the other Java™ Platform Editions to provide extensible Permissions, for example to be able to provide access to a single property, resource or function. In contrast, the MIDP 2.0 permission model provided only boolean permissions. Refer to the backward compatibility section for MIDP 2.0 Security.

The use of Permission classes allows the APIs in this specification and in separate optional package specifications to have a single consistent Permission definition on all configurations and profiles. Currently, optional packages that must be defined for CLDC, or CDC and Java SE must have different permission definitions depending on the configuration they are used with. The unification of Permissions simplifies specification, implementation, developer use and conformance testing.

Sensitive APIs and functions use permissions to restrict access. Each API defines the Permissions needed to allow access to the API. Any API or function of this specification which is not security sensitive, having no permissions defined for them, are implicitly accessible by all MIDlet suites.

6.2 MIDlet Suite Trust Model

Security for 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. A MIDlet Suite is bound to a domain during installation and MUST remain bound to the same domain until it is deleted. 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. MIDlet suites that are identified as untrusted must execute in a secure domain defined as a restricted environment where access to protected APIs or functions is either not allowed or is allowed with explicit user permission. The Security Policy Chapter defines the policy for the protection domain(s).

MIDlet Suites Trust Model Using X.509 PKI describes a mechanism for identifying trusted MIDlet suites through signing and verification. Implementations of this specification MUST recognize MIDlet suites signed using PKI as trusted MIDlet suites and they MUST be verified according to the formats and processes specified in MIDlet Suites Trust Model Using X.509 PKI.

Table 6-1: Definition Of Security Terms
### Security for MIDP Applications

#### Security for MIDP Applications

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protection Domain</td>
<td>A set of <em>Allowed</em> and <em>User</em> Permissions that may be granted to a MIDlet suite</td>
</tr>
<tr>
<td>Permission</td>
<td>A Permission class defined for an API or function to prevent it from being used without authorization</td>
</tr>
<tr>
<td>Function group</td>
<td>A collection of Permissions protecting similar resources or functions. Primarily used for optimising user experience.</td>
</tr>
<tr>
<td>Trusted MIDlet Suite</td>
<td>A MIDlet suite that is a) allowed to be installed, and b) for which the authenticity and the integrity of the JAR can be verified by the device, and c) bound to a protection domain. For an implementation of this specification that recognizes MIDlet suites signed using PKI, a trusted MIDlet suite is a MIDlet suite that was signed and verified as specified in <a href="https://example.com">MIDlet Suits Trust Model Using X.509 PKI</a></td>
</tr>
<tr>
<td>Untrusted MIDlet Suite</td>
<td>A MIDlet suite that is a) allowed to be installed, and b) for which the authenticity and the integrity of the JAR can not be verified by the device and c) bound to a designated protection domain.</td>
</tr>
</tbody>
</table>

#### 6.3 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-Permission-<n>, MIDlet-Permission-Opt-<n>, MIDlet-Permissions, and MIDlet-Permissions-Opt` attributes
- The user who may be asked to grant permission

#### 6.3.1 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 may be 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 modification except by authorized parties.
- 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`.

#### 6.3.2 Protection Domain

A protection domain defines a set of Permissions and related interaction modes. A protection domain consists of Permissions that are checked against those requested by the MIDlet suite:

- *Allowed* Permissions — 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 MUST NOT require any user interaction.
Security for MIDP Applications

- **User Permissions** — 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.

Within a protection domain, each Permission MUST be exactly one of *User* or *Allowed*.

When performing a Permission check against a Permission in the policy, as required of the java.security.AccessController.checkPermission method the *User* Permissions MUST be checked first, then the *Allowed* Permissions. This ensures that checking Permissions against the policy has the same behavior including prompts and initial values even if two or more Permissions in the Policy could grant access.

### 6.3.3 Permissions

Permissions are the means to protect access to a single property, resource or function which require explicit authorization before being executed. Whenever such protection is needed, each API must define the Permission classes and parameters of each protected function. Permissions are subclassed from java.security.Permission. Each API may define new Permission subclasses or may use existing Permission classes such as java.security.BasicPermission. New Permission classes should be defined in the package they protect. Permission checks are performed by calling java.security.AccessController.checkPermission. Note that for *User* permissions the method must block if a response is needed from the user. Permissions MUST be checked by the implementation during the execution of the protected function. Since by calling java.security.AccessController.checkPermission a MIDlet is able to check if a certain permission is allowed by the security policy or not, MIDlet developers must be aware that for permissions set to *User* in the policy, such a check will trigger unnecessary and confusing prompts to the user. Therefore it is advised that MIDlets do not check permissions unless absolutely necessary.

Permissions for MIDP APIs use and extend the Permission classes defined in the Configuration ([CLDC] or [CDC]). Applications check if a Permission is allowed by the security policy using the checkPermission method of java.security.AccessController. When a Permission is not granted the java.security.AccessControlException, a subclass of java.lang.SecurityException, is thrown.

For more detail please refer to the Change Log of the CLDC 1.1 Maintenance Release.

#### 6.3.3.1 Recommendation for Permissions for Optional Packages

Each API in this specification that provides access to a protected function defines the needed Permissions. For APIs defined outside of MIDP, there must be a single document that specifies the permissions and the behavior of the API when it is implemented with MIDP. New API specifications and maintenance releases of existing APIs targeted at CLDC or CDC configurations MUST define the permissions and require those permissions to be used if the underlying configuration includes the java.security.Permission class. In order for MIDP 3.0 to work with existing versions of APIs that have not yet defined Permission classes however, MIDP 3.0 allows simultaneous use of both named permissions and Permission classes.

### 6.3.4 Requesting Permissions for a MIDlet Suite

All MIDlet suites that require access to protected APIs or functions MUST request the corresponding Permissions with the MIDlet-Permission-<n>, MIDlet-Permission-Opt-<n>, MIDlet-Permissions, and MIDlet-Permissions-Opt attributes. Permissions requested using zero or more MIDlet-Permission-<n> and MIDlet-Permissions attributes are critical to the function of the MIDlet suite and it will not operate correctly without them. In some cases, it is not possible
Security for MIDP Applications

to request permissions for specific resource names in the attributes; for example, the file paths corresponding the MIDlet suite's private data may not be known. In other cases, the protection domain policy may vary from device to device and access to a particular resource may not be consistently allowed; for example, access to the property containing the device or user identification. In these cases, the MIDlet must defer until runtime the identification of the resource and be prepared to handle its absence or the lack of access due to device policy. If the MIDlet suite can function correctly without access to the resources requested by the Permission(s), it should request them using the \texttt{MIDlet-Permission-Opt-<n>} and \texttt{MIDlet-Permissions-Opt} attributes. The MIDlet suite is granted only as much access as allowed by the protection domain policy, which will in some cases be less than was requested. The MIDlet suite must be able to operate correctly without access to the resources requested in the permissions. All requested permissions are checked against the policy as specified in \textit{Granting Permissions to MIDlet Suites}.

Each \texttt{MIDlet-Permissions} and \texttt{MIDlet-Permissions-Opt} attribute contains 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+0099) are ignored.

Each \texttt{MIDlet-Permission-<n>} and \texttt{MIDlet-Permission-Opt-<n>} attribute contains a single Permission request and SHOULD be used over their \texttt{MIDlet-Permissions} and \texttt{MIDlet-Permissions-Opt} counterparts if the requested Permission class exists. The lowest value of \texttt{<n>} MUST be 1 and consecutive ordinals MUST be used. The first missing attribute terminates the list. Any additional attributes MUST be ignored. The value of the attribute has the following fields separated by whitespace:

- \texttt{Classname} – the fully qualified name of a subclass of \texttt{java.security.Permission}. The \texttt{Classname} is required. For example, \texttt{javax.microedition.io.HttpProtocolPermission}.
- \texttt{Name} – the name of the protected resource. The \texttt{Name} is required if the \texttt{Classname} constructor requires one or more arguments. For example, \texttt{http://server/index.html}.
- \texttt{Action} – the action(s) requested. The \texttt{Action} field is required if the \texttt{Classname} constructor requires two or more arguments. The \texttt{Action} value must contain one or more actions as specified by second argument of the constructor of \texttt{Classname}. For example, in a \texttt{java.util.PropertyPermission}, the action read.

Some examples of class-based MIDlet Permissions are:

| MIDlet-Permission-1: java.microedition.io.HttpProtocolPermission | http://*:* |
| MIDlet-Permission-2: java.microedition.io.SocketProtocolPermission | socket://4321 |
| MIDlet-Permission-Opt-1: java.lang.PropertyPermission | ** | read |

Authorization of MIDlet suites uses protection domain information and critical and non-critical Permissions requested in the MIDlet suite. Verification of requested Permissions results in failed installation of a MIDlet suite and the \textit{Status Code 910 (Application authorization failure)} returned in the status report if any of the following errors occur:

- For each \texttt{MIDlet-Permission-Opt-<n>} requested that is not critical to the MIDlet suite, if either the Permission is not in the domain or the Permission class can not be found, then the attribute is ignored. For each remaining \texttt{MIDlet-Permission-<n>}, if a constructor with the appropriate number of arguments cannot be found or is inaccessible or the arguments supplied would fail the static requirements of the constructor, then the installation MUST fail. If the Permission class has only a one argument constructor, then it is an error to supply the \texttt{Action} field.
Security for MIDP Applications

- If any of the requested critical Permissions are not contained in the protection domain (Allowed or User Permission sets then the MIDlet suite does not have sufficient authorization and the installation MUST fail.
- If any of the requested non-critical Permissions are not contained in the protection domain (Allowed or User) Permission sets then the MIDlet suite MUST be installed unless there are other errors.

6.3.4.1 LIBlet Permissions

LIBlets MUST declare their required access to protected APIs or functions by requesting the corresponding permissions. Critical permissions are declared using zero or more LIBlet-Permission-<n> and LIBlet-Permissions attributes. Non-critical permissions are declared using zero or more LIBlet-Permission-Opt-<n> and LIBlet-Permissions-Opt attributes. Each LIBlet-Permission-<n> and LIBlet-Permission-Opt-<n> attribute contains a single permission request and SHOULD be used over their LIBlet-Permissions and LIBlet-Permissions-Opt counterparts if the requested Permissions class exists. For the value of the attribute, see Requesting Permissions for a MIDlet suite.

The LIBlet by itself will not be bound to any security protection domain independently. The LIBlet is executed inside the protection domain of the MIDlet it is bound to. The LIBlet-Permission-<n> and LIBlet-Permission-Opt-<n> attributes MUST NOT be treated as a permission request to the AMS. It is the MIDlet that makes the permission requests using MIDlet-Permission-<n> and MIDlet-Permission-Opt-<n> attributes.

LIBlet-Permission-<n>, LIBlet-Permission-Opt-<n>, LIBlet-Permissions and LIBlet-Permissions-Opt attributes are used for declarative purpose; that is, a MIDlet suite developer can instantly see which protected APIs or functions a given LIBlet needs to access. Since a LIBlet itself is able to depend on other LIBlets, it is crucial that all permissions requested in the MIDlet dependency chain are listed under MIDlet-Permission-<n>, MIDlet-Permission-Opt-<n>, LIBlet-Permissions and LIBlet-Permissions-Opt attributes. Only permissions requested in MIDlet suite will be validated at installation time. Permissions requested in MIDlet-Permission-<n>, MIDlet-Permission-Opt-<n>, LIBlet-Permissions and LIBlet-Permissions-Opt attributes are the union of Permissions required by MIDlet suite and all LIBlets in the dependency declaration chain. If any LIBlet-Permission-<n>, LIBlet-Permission-Opt-<n>, LIBlet-Permissions, or LIBlet-Permissions-Opt permissions requested by a LIBlet are not also requested by dependent MIDlet suite, then a runtime security exception may occur.

6.3.5 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" mode, if selected by the user, MUST NOT result in any additional prompts. If blanket mode is the default setting or otherwise not explicitly selected by the user, the user MUST be prompted once on or before the first invocation of the protected API or function. It is valid for every invocation of an API by a MIDlet suite until it is deleted or a different interaction mode is selected by the user.
- "session" mode MUST prompt the user on or before the first invocation of the protected API or function. This setting is valid from the invocation of a MIDlet until it terminates. When the user re-invokes the MIDlet the prompt MUST be repeated.

Session interaction mode allows APIs or functions to be used more than once without re-prompting the user. An implementation may re-prompt the user based on a count, interval or event to increase the security of the system and to improve the usability of the user interface.
Security for MIDP Applications

- "oneshot" mode MUST prompt the user on each invocation of the protected API or function.

The choice of user permission interaction modes is driven by the security policy and the device implementation. Each user permission has an initial interaction mode and a set of other available interaction modes. The user MUST be provided with a means to select one of the available user interaction modes.

The user MUST always be able to deny permission. If the user denies the permission the implementation SHOULD remember this answer and SHOULD NOT present further user prompts until the application is deleted (in Blanket mode) or terminated (in Session mode).

If a device reset occurs that results in application settings being returned to their original or default values, then the interaction mode settings for all installed applications SHOULD revert to their default values.

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

Whenever the interaction mode is changed by the user the behavior MUST follow the rules associated with the given interaction mode.

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

6.3.6 Granting Permissions to MIDlet Suites

The Permissions granted to a MIDlet Suite are based on protection domain and the Permissions requested in the MIDlet Suite. The Permissions granted to the MIDlet suite are established by the following:

- During installation the MIDlet suite MUST have been bound to a single protection domain and remains bound to the same domain until it is deleted.
- During installation the requested Permissions are identified from the MIDlet-Permission-<n>, MIDlet-Permission-Opt-<n>, MIDlet-Permissions, and MIDlet-Permissions-Opt attributes.
- For each Permission requested using MIDlet-Permission-Opt-<n> attributes, if either the Permission is not in the domain or the Permission class can not be found, then the attribute is ignored. Then each permission in the protection domain with the same permission class is compared with the requested permission to select the least privileged permission. If the requested permission is implied by the protection domain then the requested permission is granted to the MIDlet suite. If not, then if the protection domain permission is implied by the requested permission then the domain policy permission is granted to the MIDlet suite. If neither permission implies the other then neither is granted to the MIDlet suite.
- For each MIDlet-Permissions-Opt attribute, if any of the requested permissions are unknown to the device, they are removed from the requested permissions.
- For each Permission requested using MIDlet-Permission-<n> attributes, the Permission.implies methods are used to check that some Permission in the domain "implies" each requested permission. Each requested permission that is implied by the domain policy is granted to the MIDlet suite.
- For each MIDlet-Permissions attribute, if any of the requested permissions are unknown to the device, then the MIDlet suite MUST NOT be installed or invoked.
- 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, every protected API or function MUST check that the appropriate Permission is implied by some Permission granted to the MIDlet Suite. If the granted Permission is a User Permission, the user interaction mode determines when the user must be prompted and to require explicit user permission before being permitted.

6.3.6.1 Example of Granting Permissions to MIDlet Suites

An example illustrates the steps used to determine the permissions granted to a MIDlet suite. It focuses on the permissions that are available to the application.

For these examples the security domain policy contains permissions:

```
grant user "NetAccess" session,blanket,oneshot,no {
    permission javax.microedition.io.HttpProtocolPermission "http://*";
}
grant user "PhoneCall" session,blanket,oneshot,no {
    permission CallPermission "tel://*";
}
grant allowed "PrivateFiles" {
    permission javax.io.FilePermission "file:///User1/Home" "read,write";
}
grant allowed "WriteLog" {
    permission java.util.PropertyPermission "Logfile" "read,write"
    permission java.util.PropertyPermission "microedition.*" "read";
}
```

6.3.6.1.1 Permission Example Using Required and Optional Permissions

In the first example, the MIDlet suite requires access to the network and to a property and makes optional requests to read all properties and access all files:

- MIDlet-Permission-2: java.util.PropertyPermission "Logfile" "write"
- MIDlet-Permission-Opt-1: java.util.PropertyPermission "*" "read"
- MIDlet-Permission-Opt-2: javax.io.FilePermission "file:///*" "read,write"

When the MIDlet Suite is executed the resulting set of permissions granted is in conformance with the verification steps described in section Granting permissions to MIDlet suites. The permissions granted are:

```
grant user "NetAccess" session,blanket,oneshot,no {
}
grant allowed "PrivateFiles" {
    permission javax.io.FilePermission "file:///User1/Home" "read,write";
}
grant allowed "WriteLog" {
    permission java.util.PropertyPermission "Logfile" "write"
    permission java.util.PropertyPermission "microedition.*" "read";
}
```

In this first example, the java.io.FilePermission requests access to all possible files but because it is requested in the MIDlet-Permission-Opt-2 attribute the access granted is the narrower of the request and the protection domain policy. The CallPermission in the domain is not granted because the MIDlet Suite did not request it. The javax.microedition.io.HttpProtocolPermission is granted only to the requested host even though the domain allows access to any host. The java.util.PropertyPermission to write the "Logfile" property is granted because it is requested and is allowed by one of the
java.util.PropertyPermission in the domain policy. The MIDlet Suite is granted access to the "microedition.*" properties since it was requested in the MIDlet-Permission-Opt-1 attribute, but only for the narrower of the requested permission and the protection domain permission.

**6.3.6.1.2 Permission Example Using Only Required Permissions**

In the second example, the MIDlet suite requires all permissions to operate correctly and uses only the MIDlet-Permission-<n> attributes.

| MIDlet-Permission-1: java.util.PropertyPermission "**" "read" |
| MIDlet-Permission-2: javax.io.FilePermission "file:///*" "read,write" |

In this second example, the installation fails because permissions are denied. The java.io.FilePermission requires access to all possible files. But because the protection domain policy permission is narrower, access must be denied. The java.util.PropertyPermission requires access to all properties. But because the protection domain permission is narrower access must be denied.

**6.3.7 External Domain Policy Format**

An external representation for protection domains allows clear communication between developers, operators, and manufacturers. Implementations MAY use the external domain policy format to configure security policy and MAY support other policy representations. Provisioning of security policy is outside the scope of this specification.

The format includes all aspects of a domain policy including the domain name, optional bindings to Protection Domain Root Certificates, and permissions grouped when applicable and marked as Allowed or User permissions and the user interaction modes.

The policy format uses the UTF-8 encoding and semantics for granting permissions to eligible applications. A policy consists of a number of “grant” clauses; each clause identifies the condition under which it is granted and contains one or more Permissions. The grant clauses define alternatives to grant the permissions if the user responds positively to a prompt or to allow permissions requested by applications.

Permissions which must be granted by the user using a single set of prompts form a “grant user” clause that includes the function group name and a list of the initial and other available user permission interaction modes. The first listed mode is the initial mode.

The general form of the format is:

- The domain described by name optionally followed by the distinguished names of the corresponding Protection Domain Root Certificates.
- One or more grant clauses
- Each grant clause MUST include a directive to scope the permissions to be Allowed or User.
  - A User grant MUST include a name, the initial and other interaction modes. The permissions are granted only if the application has requested the permission and the user replies positively to a prompt.
    - The name is used to identify the user grant clause.
    - The initial and other available interaction modes MUST be supplied and may be one of “oneshot”, “session”, or “blanket”.
    - All Permissions in a single grant MUST share the same initial and available user interaction modes.
  - An Allowed grant contains a name and permissions that are granted only if the application has requested the permission.
Each grant group includes one or more Permissions, each permission is the name of a subclass of java.security.Permission optionally followed by arguments for name and action.

The following is an ANTLR grammar of the external domain policy format. For information about ANTLR, see www.antlr.org.

```
grammar domain_policy;

domain_policy: domain+;
domain: 'domain' Identifier root_dn_subjects? ';' grant+;

root_dn_subjects: Identifier (',' Identifier )* ;
grant: 'grant' (grant_user | grant_allowed );
grant_allowed: 'allowed' grant_name permissions;
grant_user: 'user' grant_name initial_mode other_modes permissions;
grant_name: '"' Identifier '"';
permissions: '{' permission+ '}';
permission: 'permission' permission_class target_name? target_action? ';';
permission_class: Identifier ( '.' Identifier)*;
target_name: '"' ( .~ '"')+ '"';
target_action: '"' ( .~ '"')+ '"';
initial_mode: interaction_mode;
other_modes: (',' interaction_mode)*;
interaction_mode:   'blanket' | 'session' | 'oneshot';

start       :       .+;

ALPHA:      ('a'..'z'|'A'..'Z');
DIGIT:       '0'..'9';
SYMBOL      :       ('*' | ':' | '/');
Identifier: ALPHA (ALPHA | DIGIT)*;
WS: (' '|'	'|'
'|'')+ ;
```

A brief example below illustrates the format. Refer to Appendix B for security policy examples using this format.

```
domain Sample1;
grant allowed {
    permission java.util.PropertyPermission "java.version";
}
grant user "network" oneshot,session,blanket {
    permission javax.microedition.io.HttpProtocolPermission "http://*";
}
```

### 6.4 Application Level Access Authorization

The MIDP 3 platform allows applications across different security domains to share RMS data, and communicate at runtime using the IMC protocol or application events. Access authorization at the application level is needed to secure application level data sharing and runtime inter-
Security for MIDP Applications

MIDlet communication. To enable an application to individually control access requested by other applications, a reliable mechanism to authenticate application identities is fundamental to any access authorization.

The application authorization framework relies on X.509 PKI signing and protection domain authorization. For the details on signing and signing certificates, refer to MIDlet Suites Trust Model Using X.509 PKI. Table 6-2 below summarizes what information can be used for application level access authorization (credentials of an application).

<table>
<thead>
<tr>
<th>Authorization result</th>
<th>Information Available for Application Access Authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDlet suite was not signed</td>
<td>Domain name, vendor name</td>
</tr>
<tr>
<td>MIDlet suite’s JAD has only one successfully verified certificate chain</td>
<td>Domain name, vendor name, signing certificate that was successfully verified against Application Access Root Certificate</td>
</tr>
<tr>
<td>MIDlet suite’s JAD has more than one successfully verified certificate chain</td>
<td>Domain name, vendor name, any of the signing certificates that were successfully verified against Application Access Root Certificate</td>
</tr>
</tbody>
</table>

Table 6-2: Determining Application Credentials

6.4.1 Declaring Application Level Access Authorization

Application level access authorization provides a mechanism for a MIDlet suite to restrict access to its shared resources. A MIDlet suite can restrict access to its resources on the basis of the domain of the accessing MIDlet, the vendor of the accessing MIDlet, the signer of the accessing MIDlet, or any combination of the above. The MIDlet-Access-Auth-Type-<n> attribute is used to define the basis for the restricted access. Each MIDlet-Access-Auth-Type-<n> attribute represents a combination of domain, vendor and signer that is valid to access this resource. Application level access authorization is declared only by MIDlet suites and never by LIBlets. The allowed access authorization types are defined in Table 6-3 below.

Table 6-3: Application Attributes Required for Application Level Access Authorization

| MIDlet-Access-Auth-Type-<n> | domain=SELF|ANY;vendor=<vendorname>|ANY;signer=<signercertalias>|ANY |
|----------------------------|---------------------------------------------------------------|
|                            | • domain=SELF|ANY Access is granted to MIDlets in the same domain as the current MIDlet if SELF is specified, or granted to MIDlets that are from any domain if ANY is specified. |
|                            | • vendor=<vendorname>|ANY Access is granted to MIDlets that have a vendor name that matches <vendorname>, or granted to MIDlets that are from any vendor if ANY is specified. <vendorname> : a string of characters including whitespace but specified within quotes. |
|                            | • signer=<signercertalias>|ANY Access is granted to MIDlets that are signed by the signing certificate that corresponds to the alias specified by <signercertalias>, or granted to MIDlets that are signed by any signing certificate if ANY is specified. If a <signercertalias> is used in this attribute, a corresponding attribute MIDlet-Access-Auth-Cert-<n> with a matching <alias> MUST exist, if not installation of the MIDlet suite MUST |
Security for MIDP Applications

fail. See description of MIDlet-Access-Auth-Cert-<n> for more details. <signercertalias>: a string of alpha numeric character, excluding whitespace.

In each authorization type declaration, the domain constraints, vendor constraints and signer constraints (triplet constraints) MUST all be present. If any one of them is missing, the attribute is invalid and installation MUST fail with Status Code 907 (Invalid JAR). In this attribute, the lowest value of <n> MUST be 1 and consecutive ordinals MUST be used. The first missing entry terminates the list. Any additional entries MUST be ignored.

<table>
<thead>
<tr>
<th>MIDlet-Access-Auth-Cert-&lt;n&gt;</th>
<th>&lt;alias&gt; &lt;base 64 encoded signing certificate&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• &lt;alias&gt; is a unique alias for a signing certificate, and is the same format as &lt;signercertalias&gt; specified in the signer field of the MIDlet-Access-Auth-Type-&lt;n&gt; attribute. &lt;alias&gt; is used to match with the &lt;signercertalias&gt; specified in the signer field of the MIDlet-Access-Auth-Type-&lt;n&gt; attribute.</td>
</tr>
<tr>
<td></td>
<td>• &lt;n&gt; in the attribute name is a non-negative integer starting with 1.</td>
</tr>
<tr>
<td></td>
<td>• Each unique &lt;signercertalias&gt; specified in the MIDlet-Access-Auth-Type-&lt;n&gt; MUST have a unique MIDlet-Access-Auth-Cert-&lt;n&gt; attribute. If the &lt;signercertalias&gt; in MIDlet-Access-Auth-Type-&lt;n&gt; does not have a MIDlet-Access-Auth-Cert-&lt;n&gt; to map to a signing certificate, the installation of the MIDlet suite MUST fail with Status Code 905 (Attribute mismatch). If the &lt;signercertalias&gt; has multiple entries that match in MIDlet-Access-Auth-Cert-&lt;n&gt;, then the installation MUST fail with Status Code 907 (Invalid JAR).</td>
</tr>
<tr>
<td></td>
<td>• A whitespace MUST separate the &lt;alias&gt; and the base64 encoded signing certificate. If this is not the case, the installation of this MIDlet suite MUST fail with Status Code 907 (Invalid JAR).</td>
</tr>
</tbody>
</table>

In each attribute, only one vendor name, and signer alias is specified. If a MIDlet developer would like to specify access based on multiple signers with the same vendor name (or vice versa), multiple MIDlet-Access-Auth-Type-<n> attributes must be used.

Access Authorization attributes MUST be present in a MIDlet suite’s Manifest, if used. Access authorization attributes present in a LIBlet JAD or Manifest MUST be ignored by the implementation. MIDlets within the same MIDlet suite MUST NOT be subject to the application level access authorization.

6.4.2 Granting Application Level Access

If Access Authorization restrictions are not defined by a MIDlet using the MIDlet-Access-Auth-Type-<n> attributes, i.e., the MIDlet does not contain at least one MIDlet-Access-Auth-Type-<n> attributes in its Manifest, the MIDlet does not intend to restrict access to its resources. Access is granted to all accessing MIDlets for the resources in such a scenario.

In each authorization type declaration MIDlet-Access-Auth-Type-<n>, the triplet constraints are combined (AND operation); the accessing MIDlet must satisfy all the constraints defined by the domain/signer/vendor fields within one attribute. In case there are multiple MIDlet-Access-Auth-Type-<n> attributes, that is n > 1, the accessing MIDlet MUST satisfy at least one of the MIDlet-Access-Auth-Type-<n> restrictions for application level access to be granted. That is, the resulting access restrictions of the MIDlet that is defining its access restriction is a result of an OR operation on all the MIDlet-Access-Auth-Type-<n> attributes.
Figure 6-1 depicts the verification steps that lead to granting or denying application level access. While the figure presents all the checks that must be performed, the actual sequence of checks is implementation specific.

**Figure 6-1 : Verification Steps for Granting & Denying Application Level Access**

- **MIDlet B is sharing a resource**
  - **MIDlet A is accessing this shared resource**
    - **Do MIDlet A and B belong to the same Suite?**
      - **True**
      - **False**
        - **Do MIDlet-Access-Auth-Type<i> attributes exist in MIDlet B?**
          - **True**
            - **i++**
          - **False**
            - **Increment i**

- **Does MIDlet A match the domain constraints defined in attribute MIDlet-Access-Auth-Type<i>?**
  - **False**

- **Does MIDlet A match the vendor constraints defined in attribute MIDlet-Access-Auth-Type<i>?**
  - **False**

- **Does MIDlet A match the signer constraints defined in attribute MIDlet-Access-Auth-Type<i>?**
  - **True**
  - **Access Granted**
  - **False**
    - **Access Denied**
In the above diagram, the accessing MIDlet (MIDlet A) is checked if it satisfies the access authorization type defined in the MIDlet-Access-Auth-Type-<n> of the sharing MIDlet (MIDlet B).

Checking for domain constraints in the above diagram implies the following. If domain=ANY is specified; or if domain=SELF is specified in the attribute and the domain of MIDlet A is the same as MIDlet B; then the implementation MUST move on to the vendor constraint check. If not, the implementation MUST move on to the next MIDlet-Access-Auth-Type-<n> attribute if one exists and start over the constraints checks.

Checking for vendor constraints in the above diagram implies the following. If vendor=ANY is specified; or if vendor name is specified and the vendor name of MIDlet A (as specified in its MIDlet-Vendor attribute) exactly matches the one specified in the vendor constraint; the signer constraints MUST be checked. If the name does not match, the implementation MUST move on to the next MIDlet-Access-Auth-Type-<n> attribute if one exits and start over with the constraints checks (starting with domain).

Checking for signer constraints in the above diagram implies the following. If signer=ANY is specified; or if signer alias is specified (using signer=<signercertalias>) and one of MIDlet's A valid signing certificates matches the certificate corresponding the signercertalias (obtained from MIDlet-Access-Auth-Cert-<n>); then access is granted to MIDlet A as it has satisfied all the constraints via the sequence of checks. If the signer does not match, the implementation MUST move on to the next MIDlet-Access-Auth-Type-<n> attribute if one exits and start over with the constraints checks (starting with domain).

An accessing MIDlet MUST satisfy at least one of the Access authorization types, where each type is specified using the MIDlet-Access-Auth-Type-<n> attribute. A sharing MIDlet suite may specify multiple types; however, it is not necessary to check against all the attributes if one of the types satisfies access requirements.

For example,

```
MIDlet-Access-Auth-Type-1:domain=SELF;vendor=ANY;signer=coolcompanycert
MIDlet-Access-Auth-Type-2:domain=ANY;vendor="Cool Company Inc.";signer=ANY
```

implies that a requesting MIDlet that belongs to the same domain as the current MIDlet AND is also signed by the certificate that corresponds to the coolcompanycert is allowed to access; OR, a requesting MIDlet that has a MIDlet-Vendor that is Cool Company Inc. is granted access.

### 6.5 Combined usage of MIDP 2.0 and MIDP 3.0 security models

As optional JSRs are gradually updated to incorporate the MIDP 3.0 permission model and provide the appropriate Permission classes, developers may face situations where the JSRs used by the application have not yet provided the appropriate Permission classes. As a transitional mechanism, MIDP 3.0 allows the simultaneous use of both the MIDP 2.0 permission attributes and MIDP 3.0 permission attributes.

The MIDlet-Permission-<n> or MIDlet-Permission-Opt-<n> attributes may appear in the same JAD and JAR Manifest as the MIDlet-Permissions and MIDlet-Permissions-Opt attributes. In such a case, all permissions requested by all MIDlet-Permission-<n>, MIDlet-Permission-Opt-<n>, MIDlet-Permissions, and MIDlet-Permissions-Opt attributes MUST be verified against the protection domain. If the MIDlet suite does not have sufficient authorization for any of the critical permissions requested by MIDlet-Permission-<n> and MIDlet-Permissions attributes, the installation MUST fail and return the Status Code.
Security for MIDP Applications

910 (Application authorization failure). For permissions requested via the MIDlet-Permission-Opt-<n> and MIDlet-Permission-Opt-<n> attribute, the MIDlet suite will be granted only as much access as is allowed by the protection domain policy, which will in some cases be less than was requested.

The LIBlet-Permission-<n> or LIBlet-Permission-Opt-<n> attributes may appear in the same JAD and JAR Manifest as the LIBlet-Permissions and LIBlet-Permissions-Opt attributes. These permission attributes are declarative and MUST NOT be treated as a permission request to the AMS.
MIDlet Suites Trust Model Using X.509 PKI

7.1 Introduction

The MIDlet suite is protected by signing the JAR. The mechanisms defined in this chapter allow signing and authentication of MIDlet suites based on X.509 Public Key Infrastructure, so that a device can verify the signer and trust a MIDlet suite. The signatures and certificates are added to the application descriptor as attributes. The device uses them to verify certificate chains and the signatures to determine whether the MIDlet suite is trusted and can be bound to a protection domain that will authorize the MIDlet suite to perform protected functions by granting permissions allowed in a given protection domain. Additionally, authenticating the signer of a MIDlet suite enables an application to individually control access requested by other applications, known as Application Level Access Authorization.

The security model involves the MIDlet suite, a signer, and 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 a SIM(WIM) card/USIM module. Implementations MUST support X.509 Certificates and corresponding algorithms. Devices MAY support additional signing mechanisms and certificate formats.

7.1.1 Transport & Security Standards

MIDP 3.0 devices are expected to operate using standard Internet and wireless protocols and techniques for transport and security. The mechanisms for securing Internet content are based on existing public key cryptography standards, including [RFC2437], [RFC3280], [RFC2560], and [WAPCert].

7.1.2 Definition of Terms

The terms Trusted MIDlet suite, Permission, and Protection Domain are defined by Security for MIDP Applications. Table 7-1 below defines additional terms.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root Certificate</td>
<td>The general term used in the context that does not specifically address Protection Domain Authorization or Application Access Authorization</td>
</tr>
<tr>
<td>Protection Domain Root Certificate</td>
<td>Root Certificate associated with a protection domain that the device implicitly trusts to verify and authorize downloaded MIDlet suites</td>
</tr>
</tbody>
</table>
Application Access Root Certificate | Root Certificate that device implicitly trust to verify downloaded MIDlet suites. Application Access Root Certificate is often the same certificate as Protection Domain Root Certificate

The device manufacturer and operator decide which of the root certificates used for verifying the signing certificate can be used for domain authorization, application access authorization, or both. Domain authorization and application access authorization do not impose any special requirements upon root certificates other than being able to participate in the verification procedure described in Verifying Signing Certificate.

7.2 Signing a MIDlet Suite

The signer of the MIDlet suite may be the developer or some entity that may be responsible for distributing, supporting, or billing for its use. The signer will need to have a certificate that can be validated against one of the Root Certificates found on the device or smart card. 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.

It should be noted that the signer of the MIDlet suite is responsible to its Root Certificate owner for protecting the protection domain or any other 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), Root Certificate owner may delegate signing MIDlet suites to a third-party and in some circumstances, the author of the MIDlet.

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 both in the manifest and in the application descriptor, 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.

7.2.1 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 Certification Authority (CA) creates a RSA X.509 (version 3) certificate and returns it to the signer.

7.2.2 Inserting Certificates into the Application Descriptor

1. The certificate chain MUST include the certificate of the signer and any additional intermediate certificate but MUST NOT include the Root Certificate. The Root Certificate is expected to be found on the device.
2. Each certificate in the path MUST be encoded using Base64 encoding and MUST NOT include line breaks. It is inserted into the application descriptor as:

   MIDlet-Certificate-<n>-<m>: <Base64 encoding of a certificate, without line breaks>

   <n> := a number equal to 1 for the first certificate chain in the descriptor and incremented by 1 for each additional certificate chain. The lowest value of <n> MUST be 1 and consecutive ordinals MUST be used. The first missing entry terminates the list. Any additional entry MUST be ignored. These numbers defines the order in which the certificate chains are tested; see the Authenticating a MIDlet Suite section below.
   <m> := a number equal to 1 for the signing certificate in the certificate chain and incremented by 1 for each additional intermediate certificate. For each <n>, the lowest value of <m> MUST
be 1 and consecutive ordinals MUST be used. The first missing entry terminates the list. Any additional entry MUST be ignored.

7.2.3 Creating the RSA SHA-1 signature of the JAR

1. The signature of the JAR MUST be created with the signer’s private key using the EMSA-PKCS1-v1_5 encoding method of the PKCS#1 version 2.0 standard [RFC2437].
2. The signature MUST be Base64 encoded, and MUST NOT include line breaks. It is inserted in the application descriptor as:

   MIDlet-Jar-RSA-SHA1-<n>: <Base64 encoding of Jar signature>

   <n> := a number equals to 1 for the first signature and incremented by 1 for each additional signature. This number corresponds to the number <n> in MIDlet-Certificate-<n>-<m> attributes. The lowest value of <n> MUST be 1 and consecutive ordinals MUST be used. The first missing entry terminates the list. Any additional entry MUST be ignored.

If the number of certificate chains is not equal to the number of signature attributes, installation MUST NOT proceed. The status 906 Invalid Descriptor must be returned in the status report.

7.3 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-<n> is present in the application descriptor then the JAR MUST be authenticated by verifying the signing 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.

7.3.1 Verifying Signing Certificate

The certificate chain consists of the signing certificate from the application descriptor and other certificates as needed up to but not including the Root Certificate.

1. Get the certificate chain for the signing certificate from the Application 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 certificate chain using the basic path validation processes described in [RFC3280] using available Root Certificates.
3. If attributes MIDlet-Certificate-<n>-<m> with <n> greater than 1 are present, 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 Table 7-2 below.

### Table 7-2 : Actions Upon Completion of Signing Certificate Verification

<table>
<thead>
<tr>
<th>Result</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attempted to validate &lt;n&gt; certificate chains. No public keys of the issuer for the certificate can be found or none of the certificate chains can be validated</td>
<td>Authentication fails, JAR Installation is not allowed</td>
</tr>
<tr>
<td>More than one full certificate chain established and validated</td>
<td>Implementation proceeds with the signature verification using all successfully verified</td>
</tr>
</tbody>
</table>
7.3.2 Verifying the MIDlet Suite JAR

1. Get the public key from the verified signing certificate (above).
2. Get the MIDlet-Jar-RSA-SHA1-<n> 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 corresponding certificate chain.
5. Repeat the above steps for each MIDlet-Jar-RSA-SHA1-<n> attribute

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.

7.3.3 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 signers. The results of the verification have a direct impact on authorization. Table 7-3 below summarizes the states to which the signature verification led and which are further used for authorization at install time.

<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 JAR is unsigned</td>
<td>Authentication can not be performed, may install JAR. MIDlet suite is treated as untrusted</td>
</tr>
<tr>
<td>One or more signature attributes appear in the JAD, 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, Status Code 909 (Application authentication failure) returned in the Status Report</td>
</tr>
<tr>
<td>One or more signature attributes appear in the JAD and Root Certificates present in the keystore to validate the certificate chain but none of Root certificates is Protection Domain Root certificate</td>
<td>Authentication can not be performed, authorization can not be performed. JAR installation is not allowed, Status Code 910 (Application authorization failure) returned in the Status Report</td>
</tr>
<tr>
<td>One or more signature attributes appear in the JAD, a certificate on one or more certificate chains is expired</td>
<td>Certificate chain with expired certificates is discarded. If all certificate chains are discarded, JAD is rejected and JAR installation is not allowed, Status Code 909 (Application authentication failure) returned in the Status Report</td>
</tr>
<tr>
<td>One or more signature attributes in the JAD, any certificate rejected for reasons other than expiration</td>
<td>Certificate chains with rejected certificates is discarded. If all certificate chains are discarded, JAD is rejected and JAR</td>
</tr>
</tbody>
</table>
### 7.3.4 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 cannot 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.

### 7.4 Security of Alternative Application Representation Formats

In environments that make use of alternate application representation formats (see CLDC, §5.3.4), 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 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.

### 7.5 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 [RFC3280] Internet X.509 Public Key Infrastructure Certificate and CRL Profile. Refer to the package documentation for javax.microedition.pki for details.

#### 7.5.1 Certificate Processing for Provisioning

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 [RFC3280] § 4.2.1.13).

The application descriptor SHOULD NOT include Root Certificate in a descriptor certificate chain. Root Certificate found in the application descriptor MUST be discarded.

### 7.5.2 Certificate Expiration and Revocation

Expiration and revocation of certificates supplied in the application descriptor is checked during the authorization procedure, specifically during certificate chain 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 chain 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.

### 7.6 Scenarios of MIDlet Suite Signing

#### 7.6.1 Scenario 1a

The MIDlet suite is signed once, and the developer owns the signing certificate:

1. Developer creates a MIDlet suite and creates JAR and JAD
2. Developer generates a private-public key pair and obtains a signing certificate from Certification Authority
3. The developer's certificate is used to sign the MIDlet JAR and create the associated JAD entries
4. MIDlet suite is ready for distribution and can run on any MIDP 3.0 compliant device that can verify signing certificate used in this scenario against an appropriate Protection Domain Root Certificate

#### 7.6.2 Scenario 1b

The MIDlet suite is signed once, but the developer does not own the signing certificate:

1. Developer creates a MIDlet suite and creates JAR and JAD
2. Developer passes his MIDlet suite to a Test House for testing and signing. Signed MIDlet suite is returned back to developer. Alternatively developer passes his MIDlet suite to a Distribution Authority that would sign and prepare MIDlet suite ready for distribution
3. MIDlet suite is ready for distribution and can run on any MIDP 3.0 compliant device that can verify signing certificate used in this scenario against an appropriate Protection Domain Root Certificate

#### 7.6.3 Scenario 2

The MIDlet suite is signed multiple times, and the developer may or may not own all of the signing certificates:

1. Developer creates a MIDlet suite and creates JAR and JAD
2. Developer generates one or more private-public key pair and obtains one or more signing certificate from Certification Authorities
3. One or more of developer's certificate are used to sign the MIDlet JAR and create the associated JAD entries
4. Developer may pass his MIDlet suite to a Test House for testing and signing. Signed MIDlet suite is returned back to developer. This and the previous step may be performed in reverse order.

5. MIDlet suite is ready for distribution and can run on any MIDP 3.0 compliant device that can verify signing certificates used in this scenario against appropriate Protection Domain Root Certificates.

6. In addition to or instead of the previous step, developer may pass his MIDlet suite to a Distribution Authority that would sign and prepare MIDlet suite ready for distribution.

7. With this scenario the entity that is the last one to sign the MIDlet suite and/or is responsible for MIDlet suite distribution must perform an extra step of evaluating all signing certificates that appear in MIDlet suite JAD and rendering certificate chains so that the one potentially leading to least restrictive protection domain will be verified first. E.g. if MIDlet suite was signed with signing certificates issued by independent Certification Authorities and later signed with operator signing certificate, numbering of certificate chains should be altered so that operator certificate chain is the first one to be verified.

7.6.4 Scenario 3

The MIDlet suite is signed one or more times, and the MIDlet suite is dependent on one or more LIBlets:

1. Developer creates a MIDlet suite that depends on one or more LIBlets.
2. Developer does due diligence in testing his application with LIBlets as well as verifies all dependency LIBlets for the presence of mandatory attributes.
3. Developer creates MIDlet suite JAR and JAD and includes all required attributes specific to his MIDlet suite.
4. Hash value of all the LIBlets that MIDlet suite directly depend on are computed and included in MIDlet suite JAR manifest and JAD.
5. Developer selects one of the procedures outlined in the previous scenarios to get his MIDlet suite signed.
6. MIDlet suite is ready for distribution and can run on any MIDP 3.0 compliant device that can verify signing certificate used in this scenario against an appropriate Protection Domain Root Certificate.
MIDP 3.0 Security Policy

8.1 Introduction

All implementations of MIDP 3.0 utilizing wireless data technologies such as GSM, UMTS, CDMA, WiMAX etc., MUST comply with the Security Policy requirements in this chapter. All implementations of the Security Policy MUST follow the security framework specified in Security for MIDP Applications. Finally, devices that support trusted MIDlets MUST support the PKI-based authentication scheme defined in MIDlet Suites Trust Model Using X.509 PKI.

Security for MIDP Applications defines the framework for authenticating the source of a MIDlet suite and authorizing a 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, it specifies the common rules for APIs that can be used together with MIDP but are specified outside of MIDP. This chapter defines extensions to the base MIDlet suite security framework in the following areas:

- The required trust model for devices compliant to cellular standards
- 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 3.0
- MIDlet behavior in the roaming network
- MIDlet behavior when a SIM/USIM is changed
- The use of user permission types
- Guidelines on user prompts and notifications

8.2 Protection Domains

A protection domain is a set of permissions that can be granted to a MIDlet. The representation of a protection domain and its security policy is implementation specific. A MIDlet suite MUST belong to one and only one protection domain. A MIDlet suite is bound to a Protection Domain based on its signature. MIDlet suites that are unsigned as per the MIDP 3.0 Security Framework MUST be bound to the Unidentified Third Party Protection Domain. For a signed MIDlet suite, if none of its associated certificate chains verify against a Protection Domain Root Certificate, that MIDlet suite MUST NOT be installed. For MIDlet suites that have more than one certificate chain found in the JAD, it may be possible that more than one certificate chain verifies against a Protection Domain Root Certificate. In such a case, the certificate chain with the smallest value <n> in the MIDlet-Certificate-<n>-<m> attribute determines the domain to which the MIDlet suite is bound.

This chapter defines the following protection domains:

- Unidentified Third Party Protection Domain
- Identified Third Party Protection Domain
- Operator Protection Domain
- Manufacturer Protection Domain
MIDP 3.0 Security Policy

Definition of additional protection domains, beyond Manufacturer, Operator, Identified Third Party and Unidentified Third Party Protection Domains, is allowed by this specification. The policy for additional protection domains is outside the scope of this specification. Additional protection domains intended to meet Network Operator requirements are referred to as Operator Supplementary Protection Domains.

For any X.509 Certificate under which an application is signed, the device MUST decide whether the certificate is issued under the certification hierarchy that leads to a Protection Domain Root Certificate as follows:

- MIDlet suites signed "under the authority[1]" of the Identified Third Party Protection Domain, as per the MIDP 3.0 security framework, MUST run under the Identified Third Party Protection Domain.
- MIDlet suites signed "under the authority[1]" of the Operator Protection Domain, as per the MIDP 3.0 security framework, MUST run under the Operator Protection Domain.
- MIDlet suites signed "under the authority[1]" of the Operator Supplementary Protection Domain, as per the MIDP 3.0 security framework, MUST be run under the Operator Supplementary or Operator Protection Domain.
- MIDlet suites signed "under the authority[1]" of the Manufacturer Protection Domain, as per the MIDP 3.0 security framework, MUST be run under the Manufacturer Protection Domain.
- MIDlet suites signed "under the authority[1]" of another protection domain not specified here.
  - MIDlet suites signed with a certificate bound to a protection domain not listed above run in the designated protection domain.
  - MIDlet suites signed with a certificate not bound to any protection domain MUST NOT be installed on the device.

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 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.

8.2.1 Manufacturer and Operator Protection Domains

8.2.1.1 Manufacturer Protection Domain

The Manufacturer Protection Domain Root Certificate is used to verify manufacturer MIDlet suites and 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.

The Manufacturer Protection Domain imposes no restriction on the capabilities specified in MIDP 3.0 and other JSRs.

The following OID identifies Manufacturer Protection Domain Root Certificate:
8.2.1.2 Operator Protection Domain

An Operator Protection Domain Root Certificate is used to verify operator MIDlet suites and MUST be mapped onto the security policy for the Operator Protection Domain on the device. Operator Supplementary Protection Domain Root Certificates MUST be mapped onto the designated policy, or in its absence they MUST be mapped onto the Operator Protection Domain. The following rules apply equally to the Supplementary Protection Domains as well.

A device MUST support the security policy for the Operator Protection Domain.

A device MUST support the mechanism [SCPROV] to read Root Certificates stored in the smart card (for example, SIM, USIM or WIM).

Additionally, the device MUST support Operator Protection Domain Root Certificates stored on the device.

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. There is more than one Operator Protection Domain Root Certificate if Operator Supplementary Protection Domain Root Certificates are mapped onto the Operator Protection Domain.

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 MIDlet Download and Execution While Roaming and After Changing the Smart Card.

MIDlet Suite execution in the Operator Protection Domain may be limited by the MIDlet-Operator-Allowed attribute. If this Application Attribute is present, the implementation MUST search for a tuple that matches the MCC-MNC portion of IMSI fetched from the currently inserted SIM against those listed in the attribute. If no match is found, MIDlets from the MIDlet Suite MUST NOT be launched. The check MUST be performed before execution of any MIDlet in the suite. If this attribute is not present, the implementation MUST NOT limit Operator Protection Domain MIDlet execution based on network.

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 Operator Protection Domain Root Certificates.
MIDP 3.0 Security Policy

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 met, the device MUST NOT install the MIDlet and MUST respond as described in MIDP 3.0 Provisioning, by sending the appropriate error code.

Root Certificates may be placed in the trustedCertificates Certificate Directory File (CDF) of a WIM, SIM, or USIM or on the device. If Root Certificates are stored directly on a SIM or USIM, that is, not under the WIM application, then they shall be stored in the EF trustedCertificates CDF located under DF ([PKCS#15]), as defined by [SCPROV]. Root Certificates can be obtained only from the trusted CDF (the card holder can not update this directory) and not from any other directory of the smart card. Root Certificates found in the trustedCertificates file are considered to be Operator Protection Domain, Supplementary Operator Protection Domain 1 to 3 or Identified Third Party Protection Domain Root Certificates, depending on the trustedUsage field in the CommonCertificateAttributes associated with the certificate [PKCS#15]:

If the trustedUsage field is present and contains the following OID for key usage, then the certificate is to be considered an Operator Protection Domain Root Certificate:

```
ASN.1:
{iso(1)org(3)dod(6)internet(1)private(4)enterprises(1)sun(42)
 products(2)javaXMLsoftware(110)mipa(2)spec(2)gsm-policy(2)operator(1)}
URN:
urn:oid:1.3.6.1.4.1.42.2.110.2.2.2.1
```

If the trustedUsage field is present and contains the following OID for key usage, then the certificate is to be considered Protection Domain Root Certificate of a Supplementary Operator Protection Domain 1:

```
ASN.1:
{iso(1)org(3)dod(6)internet(1)private(4)enterprises(1)sun(42)
 products(2)javaXMLsoftware(110)mipa(2)spec(2)gsm-policy(2)operatorSupplementary(4)}
URN:
urn:oid:1.3.6.1.4.1.42.2.110.2.2.2.4
```

If the trustedUsage field is present and contains the following OID for key usage, then the certificate is to be considered Protection Domain Root Certificate of a Supplementary Operator Protection Domain 2:

```
ASN.1:
{iso(1)org(3)dod(6)internet(1)private(4)enterprises(1)sun(42)
 products(2)javaXMLsoftware(110)mipa(2)spec(2)gsm-policy(2)operatorSupplementary(5)}
URN:
urn:oid:1.3.6.1.4.1.42.2.110.2.2.2.5
```
If the trustedUsage field is present and contains the following OID for key usage, then the certificate is to be considered Protection Domain Root Certificate of a Supplementary Operator Protection Domain 3.

ASN.1:
{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)operatorSupplementary(6)}
URN:
urn:oid:1.3.6.1.4.1.42.2.110.2.2.2.6

If the trustedUsage field contains the following OID, then the certificate is to be considered an Identified Third Party Protection Domain Root Certificate. See Identified Third Party Protection Domain

ASN.1:
{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)}
URN:
urn:oid:1.3.6.1.4.1.42.2.110.2.2.2.3

Figure 8-1 below summarizes the process of assigning Root Certificates obtained through [SCPROV] to Protection Domains.

MIDlet suites installed in the Operator Protection Domain MUST store, along with the application itself, a hash of the Protection Domain Root Certificate under which the certificate used to sign the application was issued. The hash algorithm to be used is the following: starting with the Protection Domain Root Certificate, compute the 20-byte SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits) of that certificate. This method is commonly used to compute key identifiers, especially to accelerate trust chain building ([RFC3280] §4.2.1.2). The implementation MUST NOT assume for optimization purposes that X.509 key identifiers or [PKCS#15] labels 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 User Prompts and Notifications.
8.2.2 Third Party Protection Domains

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

8.2.2.1 Identified Third Party Protection Domain

Applications that are authenticated using an Identified Third Party Protection Domain Root Certificate 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 Identifed Third Party Protection Domain. However, if there are no Identified Third Party Protection Domain Root Certificates available either on the device or at the specified location in the smart card, then the Identified Third Party Protection Domain MUST be disabled.

A device SHOULD support the mechanism [SCPROV] to read Identified Third Party Protection Domain Root Certificates stored in the smart card (for example, SIM, USIM or WIM). Additionally, Protection Domain Root Certificates for the Identified Third Party Domain MAY be stored on the device. A device MUST support at least one certificate storage mechanism. If both mechanisms are supported, the device MUST use Root Certificates from both mechanisms.
MIDP 3.0 Security Policy

Root Certificates may be placed in the trustedCertificates Certificate Directory File (CDF) of a WIM, SIM, or USIM or on the device. If Root Certificates are stored directly on a SIM or USIM, that is, not under the WIM application, then they shall be stored in the EF trustedCertificates CDF located under DF ([PKCS#15]), as defined by [SCPROV]. Root Certificates can be obtained only from the trusted CDF (the card holder can not update this directory) and not from any other directory of the smart card.

If the trustedUsage field contains the following OID, then the certificate is to be considered an Identified Third Party Protection Domain Root Certificate. See Figure 8-1 for the details.

ASN.1:

```plaintext
{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)}
```

URN:

```plaintext
urn:oid:1.3.6.1.4.1.42.2.110.2.2.2.3
```

Any Root Certificates obtained after device manufacture MUST NOT have any effect on previously provisioned MIDlet suites. For subsequently provisioned MIDlet suites, such certificates MUST NOT be used as Protection Domain Root Certificates but MAY be used as Application Access Root Certificates This MUST NOT prevent obtaining Operator or Identified Third Party Authority Certificates from the specified location in a SIM, USIM or WIM.

The user MUST be able to delete or disable Identified Third Party Protection Domain Root Certificates stored on the device but not those obtained using [SCPROV]. If an Identified Third Party Protection Domain Root Certificate is to be deleted, the implementation MUST adequately warn the user of the consequence of the deletion. A disabled Identified Third Party Protection Domain Root Certificate MUST NOT be used to verify downloaded MIDlet suites. The user MUST be able to re-enable a disabled Identified Third Party Protection Domain Root Certificate that was previously disabled by the user. Furthermore, if an Identified Third Party Protection Domain Root 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 Protection Domain Root Certificate. If the user chooses to delete or disable an Identified Third Party Protection Domain Root Certificate, the implementation MAY provide an option to delete the MIDlet suites authenticated to it.

8.2.2.2 Unidentified Third Party Protection Domain

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 Manufacturer, Operator or Identified Third Party Protection Domain Root Certificate that is different than for MIDlet suites that are not signed or can not be authenticated. The recommended icons are shown in Figure 8-2 below.

<table>
<thead>
<tr>
<th>Figure 8-2 : Recommended Security Prompt Icons</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Authenticated MIDlets</strong></td>
</tr>
</tbody>
</table>

Mobile Information Device Profile v3.0 - JSR 271
© Copyright Motorola, Inc. 2002-2009. All Rights Reserved. 
Java Community Process - Final Release 
Page 90 of 891
The Unidentified Third Party Protection Domain for Unidentified Third Party 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>
</tbody>
</table>

The Unidentified Third Party Domain for Unidentified Third Party 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>

It should be noted that versions of MIDP prior to 2.1 referred to the Unidentified Third Party Protection Domain as "Untrusted".

### 8.3 Permissions for MIDlet Suites

The security policy and permissions applied to all MIDlet suites that are installed on a device MUST conform to the requirements listed in this section, whether those MIDlet suites were preloaded or preinstalled or subsequently provisioned after device manufacture.

#### 8.3.1 Mapping MIDP 3.0 Permissions onto Function Groups in Protection 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, EDGE, WiMax, etc.); such functions MUST be mapped to this group.
MIDP 3.0 Security Policy

- 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-Receive — the group represents permissions to any function that allows receiving messages (for example, SMS, MMS, etc.)
- Messaging-Send — the group represents permissions to any function that allows sending messages (for example, SMS, MMS, etc.)
- Restricted Messaging - Receive — the group represents permissions to any function that allows receiving messages from a restricted messaging service (for example, Cell Broadcast, etc.)
- Restricted Messaging - Send — the group represents permissions to any function that allows sending messages to a restricted messaging service
- Application Auto Invocation — the group represents permissions to any function that allows a MIDlet 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).
  - File System Read — the group represents permissions to any function that gives a MIDlet suite the ability to read files or directories in the file system.
  - File System Write — the group represents permissions to any function that gives a MIDlet suite the ability to write files or directories in the file system.
  - PIM Read — the group represents permissions to any function that gives a MIDlet suite the ability to read user’s phone book, calendar or any other personal database accessible using the PIM API.
  - PIM Write — the group represents permissions to any function that gives a MIDlet suite the ability to add or modify items in user’s phone book, calendar or any other personal database accessible using the PIM API.
  - 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.
  - Private Sensor Access — the group represents permission to any function that gives the MIDlet suite access to data from private sensors. An example of a private sensor could be a heart rate monitor sensor.
  - Mobile Broadcast Service — the group represents permission to any function that gives access to mobile broadcast services. This may include reading and modifying broadcast service and tuner related data, managing recordings and obtaining information about the user’s purchased subscriptions and other information about the user’s viewing habits.

- Resource protection-related groups:
  - NFC Write Access — the group represents permissions to any function that gives a MIDlet suite the ability to write a Near Field Communication (NFC) compliant tag.
  - UI Customization — the group represents permissions to customize the device UI. This includes permissions to remove and modify user customization elements, such as UI themes that the user may have paid for.
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 8-3 presents the policy that SHOULD be enforced using the security framework defined in MIDlet Suite Security. 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>
<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></td>
<td>default setting</td>
<td>default setting</td>
</tr>
<tr>
<td></td>
<td>other settings</td>
<td>other settings</td>
</tr>
<tr>
<td></td>
<td>Blanket, Session, No</td>
<td>Session</td>
</tr>
<tr>
<td></td>
<td>other settings</td>
<td>other settings</td>
</tr>
<tr>
<td></td>
<td>Blanket, Oneshot, No</td>
<td>Oneshot</td>
</tr>
<tr>
<td></td>
<td>other settings</td>
<td>Session, No</td>
</tr>
<tr>
<td></td>
<td>default setting</td>
<td>default setting</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>other settings</td>
<td>other settings</td>
</tr>
<tr>
<td></td>
<td>Blanket, Oneshot, No</td>
<td>Session, No</td>
</tr>
<tr>
<td>Messaging-Send</td>
<td>default setting</td>
<td>default setting</td>
</tr>
<tr>
<td></td>
<td>other settings</td>
<td>other settings</td>
</tr>
<tr>
<td></td>
<td>Blanket, Session, No</td>
<td>Oneshot</td>
</tr>
<tr>
<td>Messaging-Receive</td>
<td>default setting</td>
<td>default setting</td>
</tr>
<tr>
<td></td>
<td>other settings</td>
<td>other settings</td>
</tr>
<tr>
<td></td>
<td>Blanket, Oneshot, Session, No</td>
<td>Blanket</td>
</tr>
<tr>
<td>Restricted Messaging-Send</td>
<td>default setting</td>
<td>default setting</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>
</tbody>
</table>
### MIDP 3.0 Security Policy

<table>
<thead>
<tr>
<th>Feature</th>
<th>Default Setting</th>
<th>Other Settings</th>
<th>Blanket Setting</th>
<th>Session Setting</th>
<th>No Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Messaging - Receive</td>
<td>Default Setting</td>
<td>Other Settings</td>
<td>Oneshot, Session, No</td>
<td>Other Settings</td>
<td>Oneshot, Session, No</td>
</tr>
<tr>
<td>Application Auto Invocation</td>
<td>Default Setting</td>
<td>Other Settings</td>
<td>Oneshot, Session, No</td>
<td>Other Settings</td>
<td>Session, No</td>
</tr>
<tr>
<td>Local Connectivity</td>
<td>Default Setting</td>
<td>Other Settings</td>
<td>Blanket, Oneshot, No</td>
<td>Other Settings</td>
<td>Blanket, Session, No</td>
</tr>
<tr>
<td>Multimedia Recording</td>
<td>Default Setting</td>
<td>Other Settings</td>
<td>Oneshot, Session, No</td>
<td>Other Settings</td>
<td>Session, No</td>
</tr>
<tr>
<td>File System Read</td>
<td>Default Setting</td>
<td>Other Settings</td>
<td>Session, No</td>
<td>Default Setting</td>
<td>Oneshot</td>
</tr>
<tr>
<td>File System Write</td>
<td>Default Setting</td>
<td>Other Settings</td>
<td>Session, No</td>
<td>Default Setting</td>
<td>Oneshot</td>
</tr>
<tr>
<td>PIM Read</td>
<td>Default Setting</td>
<td>Other Settings</td>
<td>Session, No</td>
<td>Default Setting</td>
<td>Oneshot</td>
</tr>
<tr>
<td>PIM Write</td>
<td>Default Setting</td>
<td>Other Settings</td>
<td>Session, No</td>
<td>Default Setting</td>
<td>Oneshot</td>
</tr>
<tr>
<td>Location</td>
<td>Default Setting</td>
<td>Other Settings</td>
<td>Blanket, Oneshot, No</td>
<td>Other Settings</td>
<td>Session, No</td>
</tr>
<tr>
<td>Landmark</td>
<td>Default Setting</td>
<td>Other Settings</td>
<td>Blanket, Oneshot, No</td>
<td>Other Settings</td>
<td>Session, No</td>
</tr>
<tr>
<td>Smart Card Communication</td>
<td>Default Setting</td>
<td>Other Settings</td>
<td>Session, No</td>
<td>Default Setting</td>
<td>No</td>
</tr>
<tr>
<td>Authentication</td>
<td>Default Setting</td>
<td>Other Settings</td>
<td>Blanket, Oneshot, No</td>
<td>Other Settings</td>
<td>No</td>
</tr>
<tr>
<td>Call Control</td>
<td>Default Setting</td>
<td>Other Settings</td>
<td>Oneshot, No</td>
<td>Default Setting</td>
<td>Oneshot</td>
</tr>
<tr>
<td>Private Sensor Access</td>
<td>Default Setting</td>
<td>Other Settings</td>
<td>Blanket, Oneshot, No</td>
<td>Other Settings</td>
<td>Session, No</td>
</tr>
<tr>
<td>NFC Write Access</td>
<td>Default Setting</td>
<td>Other Settings</td>
<td>Blanket, Oneshot, No</td>
<td>Other Settings</td>
<td>Session, No</td>
</tr>
<tr>
<td>UI Customization</td>
<td>Default Setting</td>
<td>Other Settings</td>
<td>Blanket, Oneshot, No</td>
<td>Other Settings</td>
<td>Session, 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 8-3. 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-Permission-<n> and MIDlet-Permission-Opt-<n> 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 any of the function groups, are identical then the function groups may be merged into a single function group. When these function groups are merged, the implementation MUST behave as if all of the permissions are included in the single group.

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.

Table 8-4 presents individual permissions and maps them to the function groups specified in this section. An individual permission MUST occur in only one function group.

### Table 8-4 : Mapping Permissions to Function Groups

<table>
<thead>
<tr>
<th>Permission</th>
<th>Protocol</th>
<th>Function group</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.microedition.io DatagramProtocolPermission(&quot;datagram://*&quot;).</td>
<td>datagram</td>
<td>Low Level Net Access</td>
</tr>
</tbody>
</table>
MIDP 3.0 Security Policy

<table>
<thead>
<tr>
<th>Permission</th>
<th>Identified Third Party Protection Domain</th>
<th>Unidentified Third Party Protection Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.microedition.io.SSLProtocolPermission(&quot;ssl://*&quot;). Defined in [CLDC]</td>
<td>ssl</td>
<td>Low Level Net Access</td>
</tr>
<tr>
<td>javax.microedition.io.PushRegistryPermission(&quot;*&quot;, &quot;static,dynamic,alarm&quot;). Defined in [CLDC]</td>
<td>All</td>
<td>Application Auto Invocation</td>
</tr>
<tr>
<td>java.io.FilePermission(path, &quot;read&quot;). Defined in [CDC]</td>
<td>Refer to java.io.File for the methods that read files, directories, or attributes.</td>
<td>File System Read</td>
</tr>
<tr>
<td>java.io.FilePermission(path, &quot;write&quot;). Defined in [CDC]</td>
<td>Refer to java.io.File for the methods that write or delete files or directories and set attributes.</td>
<td>File System Write</td>
</tr>
<tr>
<td>java.io.SocketPermission(host, action). Defined in [CDC]</td>
<td>Refer to java.net.Socket and java.net.DatagramSocket for methods that access the network.</td>
<td>Low Level Net Access</td>
</tr>
</tbody>
</table>

Table 8-5 collects permissions that are not mapped to any function group and sets access level for Third Party Protection Domains. Permissions indicated as Allowed are granted to MIDlet suites as shown in the Table. (See the definition of Allowed permissions in Security for MIDP Applications). Same permissions if not granted by domain policy are indicated as Not Allowed in the table. Permissions indicated as Not Permitted are those that MUST NOT be mapped to any function group, and MUST NOT be available in either the Identified or Unidentified Third Party Protection Domain.

Table 8-5 : Permissions Not Mapped To Function Groups

<table>
<thead>
<tr>
<th>Permission</th>
<th>Identified Third Party Protection Domain</th>
<th>Unidentified Third Party Protection Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.util.PropertyPermission (&quot;microedition.deviceid.*&quot;, &quot;read&quot;). See java.lang.System.getProperty(). Defined in [CLDC], [CDC].</td>
<td>Allowed</td>
<td>Not Allowed</td>
</tr>
<tr>
<td>java.util.PropertyPermission (&quot;microedition.subscriberid.*&quot;, &quot;read&quot;). See java.lang.System.getProperty(). Defined in [CLDC], [CDC].</td>
<td>Allowed</td>
<td>Not Allowed</td>
</tr>
<tr>
<td>java.util.PropertyPermission</td>
<td>Allowed</td>
<td>Allowed</td>
</tr>
</tbody>
</table>
### MIDP 3.0 Security Policy

<table>
<thead>
<tr>
<th>Permission</th>
<th>Allowed</th>
<th>Not Allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>java.util.PropertyPermission</code> (&quot;microedition.locale&quot;, &quot;read&quot;). See <code>java.lang.System.getProperty()</code>. Defined in [CLDC], [CDC].</td>
<td>Allowed</td>
<td>Allowed</td>
</tr>
<tr>
<td><code>java.util.PropertyPermission</code> (&quot;microedition.profile&quot;, &quot;read&quot;). See <code>java.lang.System.getProperty()</code>. Defined in [CLDC], [CDC].</td>
<td>Allowed</td>
<td>Allowed</td>
</tr>
<tr>
<td><code>java.util.PropertyPermission</code> (&quot;microedition.platform&quot;, &quot;read&quot;). See <code>java.lang.System.getProperty()</code>. Defined in [CLDC], [CDC].</td>
<td>Allowed</td>
<td>Allowed</td>
</tr>
<tr>
<td><code>java.util.PropertyPermission</code> (&quot;microedition.<em>&quot;, &quot;read&quot;). covers all &quot;microedition.</em>&quot; system properties not indicated above. See <code>java.lang.System.getProperty()</code>. Defined in [CLDC], [CDC].</td>
<td>Allowed</td>
<td>Allowed</td>
</tr>
<tr>
<td><code>javax.microedition.event.EventPermission</code> (&quot;*&quot;, &quot;read&quot;)</td>
<td>Allowed</td>
<td>Allowed</td>
</tr>
<tr>
<td><code>javax.microedition.event.EventPermission</code> (&quot;*&quot;, &quot;register&quot;)</td>
<td>Allowed</td>
<td>Allowed</td>
</tr>
<tr>
<td><code>javax.microedition.event.EventPermission</code> (&quot;*&quot;, &quot;post&quot;)</td>
<td>Allowed</td>
<td>Allowed</td>
</tr>
<tr>
<td><code>javax.microedition.event.EventPermission</code> (&quot;*&quot;, &quot;postsystem&quot;)</td>
<td>Allowed</td>
<td>Not Allowed</td>
</tr>
<tr>
<td><code>java.lang.RuntimePermission(&quot;*&quot;). See </code>java.lang.Thread.checkAccess()`. Defined in [CLDC], [CDC].</td>
<td>Not Permitted</td>
<td>Not Permitted</td>
</tr>
<tr>
<td><code>java.lang.reflect.ReflectPermission</code> Defined in [CDC].</td>
<td>Not Permitted</td>
<td>Not Permitted</td>
</tr>
<tr>
<td><code>java.security.AllPermission</code> Defined in [CDC].</td>
<td>Not Permitted</td>
<td>Not Permitted</td>
</tr>
<tr>
<td><code>java.security.SecurityPermission</code> Defined in [CDC].</td>
<td>Not Permitted</td>
<td>Not Permitted</td>
</tr>
<tr>
<td><code>java.security.UnresolvedPermission</code> Defined in [CDC].</td>
<td>Not Permitted</td>
<td>Not Permitted</td>
</tr>
<tr>
<td><code>java.net.NetPermission</code> Defined in [CDC].</td>
<td>Not Permitted</td>
<td>Not Permitted</td>
</tr>
<tr>
<td><code>java.io.SerializablePermission</code> Defined in [CDC].</td>
<td>Not Permitted</td>
<td>Not Permitted</td>
</tr>
<tr>
<td><code>javax.microedition.midlet.ActionsDeniedPermission</code></td>
<td>Not Permitted</td>
<td>Not Permitted</td>
</tr>
<tr>
<td><code>javax.microedition.midlet.AutoStartPermission</code></td>
<td>Not Permitted</td>
<td>Not Permitted</td>
</tr>
</tbody>
</table>

### 8.3.2 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.

8.3.3 Requirements on Restricted APIs

When the user grants permission to a function group, this action effectively grants access to all individual permissions that have been requested under this function group. An implementation MUST guarantee that a SecurityException is thrown when the caller has not been granted 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 3.0 security framework MUST have permissions defined in those JSRs, and follow the naming rules identified in this specification.
- 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 MUST only allow access to the network with appropriate user notification and consent.
- All Licensee Open Classes MUST adhere to the permission framework as defined in this document.

8.3.4 Wireless Messaging Third Party Domain Policy

The named permissions compatible with MIDP 2.x in the Messaging Function Group are defined in JSR 205 Wireless Messaging API 1.0.2, Appendix E (Deploying JSR 205 Interfaces on a MIDP 2.0 Platform [JSR205]). The corresponding permissions needed to deploy JSR 205 on a MIDP 3.0 platform must be defined by a future revision of [JSR205].

8.3.4.1 User Prompting

The Third Party Protection Domain Policy for wireless messaging 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.

8.3.5 Mobile Media Third Party Domain Policy

The permissions needed to deploy the mandatory [JSR 135] APIs with MIDP 3.0 are defined in the Multimedia Security Addendum to JSR 135 Mobile Media API, version 1.2 of [JSR 135].

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.

8.4 Presenting Function Group Information

MIDlet suites that are in either of the Third Party Protection domains MUST request permissions using MIDlet-Permission-<n> and MIDlet-Permission-Opt-<n> attributes placed in the application descriptor and JAR manifest. The requested User permissions are presented to the user using function groups. 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 Permissions for MIDlet Suites and with default and available sets of user permission types provided as guides in the tables in Permissions for MIDlet Suites. This latitude will allow the user to upgrade or downgrade the default permissions as required.
- When 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. The default permission then 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 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. The device MUST have a unique set of function group settings for each MIDlet suite that is in the Unidentified Third Party Protection Domain.
- A compliant implementation MUST be able to present the MIDlet suite name, vendor, version number and Function Group settings to the user for each installed MIDlet suite. An implementation MAY present additional implementation specific security related information to the user.
- A function group cannot be a union of permissions with different default settings and other settings. Therefore all permissions mapped to a single function group follow the convention of having the same default and available settings for all permissions in this function group. This rule MUST be taken into account when designing new permissions and policies.
- When the setting for the function group is changed it applies consistently to each permission in this function group. The initial value of the setting is the default listed in Table 8-3.
8.5 Presenting Security Related Information

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 that a signed MIDlet was authenticated to; at a minimum the Subject field of the Protection Domain Root Certificate.
- Data related to all Application Access Root Certificates that can be used to validate application access authorization for the MIDlet; at a minimum the Subject field of the Application Access Root Certificates.
- Data related to all certificates that signed the MIDlet suite and were successfully verified at installation; at a minimum all certificate's Subject, Issuer, and Serial Number fields. (As an alternative, a device may store in their entirety all certificate chains that came with the MIDlet suite's application descriptor.)
- 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.

At installation of a signed MIDlet suite, a compliant implementation MUST present the user with the Organization and Country fields within the Subject field of the signing certificate that decided on protection domain of the 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.

When the user is prompted to grant permissions for a MIDlet suite to function groups, the prompt SHOULD identify the identified source with the appropriate fields within the Subject field of the signing certificate that decided on protection domain of the MIDlet suite.

8.6 User Prompts and Notifications

The following rules MUST be followed in order to ensure informed user consent to MIDlet actions:

- The implementation MUST ensure that the user is made aware of which MIDlet the security prompt originates from.
- 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 bypass or override runtime security checks, or dialogs, prompts, and notifications to the user, generated by the system or virtual machine.
- A MIDlet MUST NOT be able to simulate key-press events to mislead the user.
8.7 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, Protection Domain Root 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 Protection Domain Root Certificate becomes available again.

If a MIDlet suite (either related to Operator or Identified Third Party Protection Domain) cannot be executed due to a smart card change, the implementation MUST NOT delete the MIDlet suite. The implementation 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 MIDlet suites. If an implementation 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 implementation 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 Subject field of the Protection Domain Root Certificate.

8.8 Revocation Checking

For all signed applications, a compliant implementation SHOULD check revocation status using the Online Certificate Status Protocol Mobile Profile as specified in [OSCP-MP]. Alternatively, an implementation MAY implement OCSP according to [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.

[1] Meaning: signed by a certificate issued under the certification hierarchy of this certificate
Legacy Compatibility for MIDP 1.0 and 2.x

9.1 Overview

MIDP 3.0 introduces several new features that require some level of adaptation to retain backward compatibility with MIDP 1.0 and MIDP 2.x content. This chapter defines these areas of adaptation, the requirements placed on MIDP 3.0 implementations to accommodate legacy content, and the behaviors that legacy content can expect when running in a MIDP 3.0 environment.

9.2 Security Compatibility for MIDP 1.0 and MIDP 2.x

The MIDP 1.0 specification constrained each MIDlet suite to operate in a sandbox wherein all of the APIs available to the MIDlets would control access to sensitive APIs or functions of the device. The MIDP 1.0 Security section below specifies the security related behaviors of MIDP 1.0 MIDlet suites that are identified with the attribute and value MicroEdition-Profile: MIDP-1.0.

MIDP 2.0 introduced the concept of trusted applications that are permitted to use APIs that are considered sensitive and are restricted by device policy. If and when a device determines that a MIDlet suite can be trusted, then access is allowed as indicated by the domain policy. The MIDP 2.0 Security section below specifies the security related behaviors of MIDlet suites that are identified with the attribute and value MicroEdition-Profile: MIDP-2.0.

9.2.1 Security for MIDP 1.0 MIDlet Suites

Every MIDP 1.0 compliant MIDlet suite MUST be able to run as an untrusted MIDlet suite. The sandbox concept defined in MIDP 1.0 is mapped to a security domain. The Security Policy Chapter defines the policy for untrusted MIDlet suites. Every implementation of this specification MUST support running untrusted MIDlet suites.

For backward compatibility, MIDP 1.0 Untrusted MIDlet suites do not request permissions explicitly in the JAR manifest or application descriptor; permissions are granted according to the policy for untrusted applications.

9.2.2 Security for MIDP 2.x MIDlet Suites

MIDP 2.0 defined the domain security model based on Named Permissions. Each permission, if requested and granted, enables access to an API or function. Named Permissions have a hierarchical organization similar to Java package names. The names of permissions are case sensitive. If the permission is for a function of a specific class in the package, then the permission name MUST include the package and class name. The set of valid characters for permission names 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,
Java Microedition.Io. Following the permission name, whether by package or class, additional modifiers may be appended with a separator of "." (Unicode U+002E).

For backward compatibility, MIDP 2.x Untrusted MIDlet suites do not request permissions explicitly in the JAR manifest or application descriptor; permissions are granted according to the policy for untrusted applications.

For each MIDlet suite that is identified with the attribute and value MicroEdition-Profile: MIDP-2.0 or MicroEdition-Profile: MIDP-2.1, the following permission attribute processing rules apply:

- If there exists one or more instances of the MIDlet-Permission-<n> or MIDlet-Permission-Opt-<n> attribute in the JAD or JAR Manifest, then the Permission classes associated with these attribute values MUST be used, and any instances of MIDlet-Permissions or MIDlet-Permissions-Opt MUST be ignored.

- If there exists no instances of MIDlet-Permission-<n> or MIDlet-Permission-Opt-<n> attribute in JAD or JAR Manifest, then the following MIDP 2.0 specific attributes MUST be processed when they appear in the JAD or JAR Manifest:
  - MIDlet-Permissions — attribute contains 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. The permissions are critical to the function of the MIDlet suite and it will not operate correctly without them.
  - MIDlet-Permissions-Opt — attribute contains 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. The permissions are not critical to the function of the MIDlet suite and it will operate correctly without them.

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, and any MIDlets in it MUST NOT be invoked. The Status Code 905 (Attribute Mismatch) must be returned in the status report.

MIDP 2.0 named permissions MUST be mapped to permissions based on java.security.Permission as described in Permissions. For backward compatibility, each named permission is mapped to a Permission as described below in Named Permissions to Class Permission Mapping. The mapping must be performed before named permissions can be checked against the security policy which consists of PermissionS. The mapping must be applied to the argument to javax.microedition.midlet.MIDlet.checkPermission.

9.2.2.1 Named Permission to Class Permission Mapping

The Configuration ([CLDC] or [CDC]) defines the Permission classes that allow access to networking protocols in the Generic Connection Framework.

<table>
<thead>
<tr>
<th>Named Permission</th>
<th>MIDP 3.0 Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.microedition.io.Connector.serversocket</td>
<td>javax.microedition.io. SocketProtocolPermission(&quot;socket://<em>:</em>&quot;)</td>
</tr>
<tr>
<td>Legacy Compatibility for MIDP 1.0 and 2.x</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------</td>
<td></td>
</tr>
</tbody>
</table>

| javax.microedition.io.Connector.ssl      | javax.microedition.io. SSLProtocolPermission("ssl://*:*") |
| javax.microedition.io.PushRegistry      | javax.microedition.io. PushRegistryPermission ("*", "static,dynamic,alarm") |

### 9.2.2.2 Example Mapping Named Permissions to Class Permissions

For example, if the application attributes for a MIDP 2.0 MIDlet suite included the following:

MicroEdition-Configuration: CLDC-1.1
MicroEdition-Profile: MIDP-2.1

The security mechanisms and policy are evaluated as specified in Granting Permissions to MIDlet Suites using the equivalent permissions as if the application attributes were:

MicroEdition-Configuration: CLDC-1.1.1
MicroEdition-Profile: MIDP-3.0
MIDlet-Permission-1: javax.microedition.io.HttpPermission "http://*:*
MIDlet-Permission-2: javax.microedition.io.HttpsPermission "https://*:*

### 9.2.2.3 Recommendation for Permissions for Optional Packages

Each API that provides access to a protected function defines the needed Permissions. For APIs defined outside of MIDP there must be a single document that specifies the permissions and the behavior of the API when it is implemented on MIDP 2.x. Optional package specifications must also specify the mapping from MIDP 2.0 named permissions to the corresponding MIDP 3.0 Permissions.

### 9.2.2.4 JAR Signature

For each MIDlet suite that is identified with the attribute and value MicroEdition-Profile: MIDP-2.0 or MicroEdition-Profile: MIDP-2.1, the following rules apply when processing the signature attributes:

- If there exists one or more instances of the MIDlet-Jar-RSA-SHA1-<n> attribute in the JAD, then any instances of MIDlet-Jar-RSA-SHA1 MUST be ignored.
- If there exist no instances of MIDlet-Jar-RSA-SHA1-<n> attribute in the JAD, then the MIDP 2.x specific attribute MIDlet-Jar-RSA-SHA1: <Base64 encoding of Jar signature> MUST be processed when it appears in the JAD. If the application descriptor contains more than one certificate chain, all the signing certificates in the application descriptor are assumed to contain the same public key.
9.2.3 Attribute Overrides in Application Descriptor

When installing a MIDlet suite with the attribute Microedition-Profile equal to "MIDP-2.0" that is a Trusted MIDlet Suite the value of each attribute that appears in both the manifest and application descriptor MUST be identical. If not identical, the installation MUST fail and return Status Code 905 (Attribute Mismatch) in the Status Report.

For Untrusted MIDlet Suites, including those with Microedition-Profile equal to "MIDP-1.0" or "MIDP-2.0" the value of an attribute that appears in the application descriptor overrides the corresponding attribute value, if any, in the manifest. The value used and returned by the MIDlet.getAppProperty method will be the value from the application descriptor, otherwise the value, if any from the manifest is used.

The following attributes MUST have identical values if they appear in both the application descriptor and JAR manifest for all MIDP 1.0 and MIDP 2.0 MIDlet suites. If not identical, the installation MUST fail and return Status Code 905 (Attribute Mismatch) in the Status Report.

- Microedition-Profile
- Microedition-Configuration

When installing a MIDlet suite with the attribute Microedition-Profile equal to "MIDP-1.0", "MIDP-2.0" or "MIDP-2.1", installation MUST NOT fail due to the absence of the attributes below as long as each attribute is present in either the application descriptor or JAR manifest:

- Microedition-Profile
- Microedition-Configuration
- MIDlet-<n>

9.2.4 Application Attributes Compatibility

Some of the application attributes defined in Table A-1: Application Descriptor Attributes are also defined in earlier versions of MIDP.

For each MIDlet suite that is identified with the attribute and value Microedition-Profile: MIDP-2.0 or MicroEdition-Profile: MIDP-2.1 in the JAD or JAR Manifest, if the following attributes also are present, and any of the attribute values contain MIDP 3.0 specific values, then installation MUST fail and return Status Code 905 (Attribute Mismatch) in the Status Report.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>MIDP 2.x Allowed Values</th>
<th>MIDP 3.0 Allowed Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDlet-Icon</td>
<td>The case-sensitive absolute name of a PNG file within the JAR used to represent the MIDlet suite.</td>
<td>The case-sensitive absolute name of an image file within the JAR, used to represent the MIDlet suite. The image file MUST be in one of the mandatory supported image formats.</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. icon is the case-sensitive absolute path name of an image (PNG)</td>
<td>The name, icon, and class of the nth MIDlet in the JAR separated by a comma. icon is the case-sensitive absolute path name of an image within</td>
</tr>
<tr>
<td>Legacy Compatibility for MIDP 1.0 and 2.x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>within the JAR for the icon of the nth MIDlet. The icon may be omitted.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>the JAR for the icon of the nth MIDlet. The image file MUST be in one of the mandatory supported image formats. The icon may be omitted.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Package

java.lang

Description

Descriptions of modified and extended features included from Java Platform Micro Edition Configurations (CLDC and CDC). MIDP is designed to be based on either the Connected, Limited Device Configuration [CLDC] or Connected Device Configuration [CDC]. Some features of [CLDC] and [CDC] are modified or extended by MIDP.

10.1 System Functions

MIDP requires a set of modified behaviors for certain [CLDC] and [CDC] system functions related to the exit of MIDlets.

10.1.1 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.

10.1.2 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.

10.2 System Properties

MIDP defines the following property values (in addition to those defined in [CLDC] and [CDC]) that MUST be made available to the application using java.lang.System.getProperty. Other properties may be available from other configurations, profiles, or from the implementation.

<table>
<thead>
<tr>
<th>System Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>microedition.deviceid.uuid</td>
<td>A unique hardware device identifier corresponding to a UUID</td>
</tr>
<tr>
<td>microedition.deviceid.imei</td>
<td>A unique hardware device identifier corresponding to the device's IMEI</td>
</tr>
<tr>
<td>microedition.deviceid.esn</td>
<td>A unique hardware device identifier corresponding to the device's ESN</td>
</tr>
<tr>
<td>microedition.deviceid.meid</td>
<td>A unique hardware device identifier corresponding to the device's MEID</td>
</tr>
<tr>
<td>microedition.deviceid.pesn</td>
<td>A unique hardware device identifier corresponding to the device's pESN</td>
</tr>
</tbody>
</table>
java.lang - Java Platform Standard Edition Classes for MIDP

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>microedition.subscriberid.uuid</td>
<td>A unique subscriber or user identifier represented by a UUID string</td>
</tr>
<tr>
<td>microedition.subscriberid.imsi</td>
<td>A unique subscriber or user identifier corresponding to an IMSI</td>
</tr>
<tr>
<td>microedition.subscriberid.msisdn</td>
<td>A unique subscriber or user identifier corresponding to an MSISDN</td>
</tr>
<tr>
<td>microedition.subscriberid.iccid</td>
<td>A unique subscriber or user identifier corresponding to an ICCID</td>
</tr>
<tr>
<td>microedition.subscriberid.euimid</td>
<td>A unique subscriber or user identifier corresponding to an EUIMID</td>
</tr>
<tr>
<td>microedition.locale</td>
<td>The current locale of the device, MUST NOT be null</td>
</tr>
<tr>
<td>microedition.profiles</td>
<td>is a blank (Unicode U+0020) separated list of the Java ME profiles that this device supports; this property MUST contain at least &quot;MIDP-3.0&quot;</td>
</tr>
<tr>
<td>microedition.platform</td>
<td>Name of the host platform or device</td>
</tr>
<tr>
<td>microedition.commports</td>
<td>A comma separated list of available logical serial ports</td>
</tr>
<tr>
<td>microedition.hostname</td>
<td>The local hostname (if available)</td>
</tr>
</tbody>
</table>

### 10.2.1 Device Identifier System Properties

- The `microedition.deviceid.uuid` property MUST contain a UUID string value that can be used to uniquely identify the device. The format of the value of this property MUST correspond to a Universally Unique Identifier (UUID) as defined in [RFC4122]. This value MUST consist of 32 hexadecimal digits. Implementations SHOULD use UUID version 1 if a 48 bit IEEE address is available; otherwise, implementations SHOULD use version 4 UUID using a randomly or pseudo-randomly generated 48 bit value. The value of this property MUST be identical for all concurrently executing or consecutively invoked MIDlets on a device.

- The `microedition.deviceid.imei` property MAY contain an IMEI string value that can be used to uniquely identify the device. The format of the value of this property, if present, MUST correspond to an International Mobile Station Equipment Identity (IMEI) as defined in [3GPP-TS23.003].

- The `microedition.deviceid.esn` property MAY contain an ESN string value that can be used to uniquely identify the device. The format of the value of this property, if present, MUST correspond to an Electronic Serial Number (ESN) as defined in [TIA-ESN] and MUST consist of 8 hexadecimal digits.

- The `microedition.deviceid.meid` property MAY contain an MEID string value that can be used to uniquely identify the device. The format of the value of this property, if present, MUST correspond to a Mobile Equipment Identifier (MEID) as defined in [3GPP2 S.R0048-A] and MUST consist of 14 hexadecimal digits.

- The `microedition.deviceid.pesn` property MAY contain a pESN string value that can be used to uniquely identify the device. The format of the value of this property, if present, MUST correspond to a Pseudo Electronic Serial Number (pESN) as defined in [3GPP2 X.S0008-0] and MUST consist of 8 hexadecimal digits.

### 10.2.2 Subscriber Identifier System Properties

- The `microedition.subscriberid.uuid` property MUST contain a UUID string value that can be used to uniquely identify the current subscriber or user for the device. The format of the value of this property MUST correspond to a Universally Unique Identifier (UUID) as defined in [RFC4122]. This value MUST consist of 32 hexadecimal digits. Implementations SHOULD use
UUID version 1 if a 48 bit IEEE address is available; otherwise, implementations SHOULD use version 4 UUID using a randomly or pseudo-randomly generated 48 bit value. The value of this property MUST be identical for all concurrently executing or consecutively invoked MIDlets on a device.

- The `microedition.deviceid.imsi` property MAY contain an IMSI string value that can be used to uniquely identify the current subscriber or user for the device. The format of the value of this property, if present, MUST correspond to an International Mobile Subscriber Identity (IMSI) as defined in [3GPP-TS23.003].
- The `microedition.deviceid.msisdn` property MAY contain an MSISDN string value that can be used to uniquely identify the current subscriber or user for the device. The format of the value of this property, if present, MUST correspond to a Mobile Station International PSTN/ISDN Number (MSISDN) as defined in [3GPP-TS24.008].
- The `microedition.deviceid.iccid` property MAY contain an ICCID string value that can be used to uniquely identify the current subscriber or user for the device. The format of the value of this property, if present, MUST correspond to an Integrated Circuit Card ID (ICCID) as defined in [ITU-T-E.118].
- The `microedition.deviceid.euimid` property MAY contain an EUIMID string value that can be used to uniquely identify the current subscriber or user for the device. The format of the value of this property, if present, MUST correspond to either a Short Form Expanded User Identity Module Identifier (SF_EUIMID) or a Long Form Expanded User Identity Module Identifier (LF_EUIMID) as defined in [3GPP2-C.S0023-C].

10.2.3 microedition.locale

The `microedition.locale` property MUST include at least the language code, and SHOULD also include the country code, followed by an OPTIONAL variant, separated by "-" (Unicode U+002D). For example, fr-FR or en-US or zh-CN-Hans.

The language codes MUST be the lower-case, two-letter codes as defined by ISO 639-1:2002.

The country codes MUST be the upper-case, two-letter codes as defined by ISO 3166-1:1997.

(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 Java SE definition for `java.util.Locale` printed strings where fields are separated by "_" (Unicode U+005F).)

10.2.4 microedition.profiles

The `microedition.profiles` system property MUST NOT contain different versions of the same profile. In particular, MIDP-1.0 and MIDP-2.x MUST NOT be listed, only MIDP-3.0.

10.2.5 microedition.platform

The value of the `microedition.platform` system property MUST conform to the following syntax:

```
Manufacturer_name
Device_model_identifier
#/version_identifier
[###additional_comments ]
```

Manufacturer name and device model number are mandatory and MUST be concatenated without spaces between them. Optional additional comments MAY be present. The version identifier and optional additional comments MUST be separated from the rest of the string with a forward slash (/). The value of the property MUST NOT contain any forward slash (/) characters other than those that are used to separate the version number and additional comments from the rest of the information.
10.2.6 microedition.commports

The value of the microedition.commports system property MUST contain a comma separated list of available logical port names, any of which can be combined with a comm: prefix as the URL string to be used to open a serial port connection. See javax.microedition.io.CommConnection for more information.

10.2.7 microedition.hostname

The value of the microedition.hostname system property MUST contain the host name of the local device, if it is available.

10.3 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 MIDlet suite and LIBlet JAR files.

Resource names refer to the contents of a MIDlet Suite or LIBlet JAR. Absolute pathnames, beginning with "/" are fully qualified file names within the JAR. The resource file is loaded from the same context as the Class instance it is called upon. Only a single JAR is searched for the resource. For example, if called on a class loaded from the MIDlet Suite, the resource will be retrieved from the MIDlet Suite JAR.

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, and resources outside the JAR MUST NOT be accessible. For example, using "./../" does NOT point outside the JAR. 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 SHOULD NOT allow class files to be read from the JAR as resources, but all other files MUST be accessible.

Since: MIDP 1.0
11.1 HTTP Networking

In addition to the javax.microedition.io classes specified in the Connected Limited Device Configuration the Mobile Information Device Profile includes the following interface for the HTTP access.

- javax.microedition.io.HttpConnection

An HttpConnection is returned from Connector.open() when an "http://" connection string is accessed.

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 IP protocol in this chapter covers both IPv4 and IPv6 addressing.

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.

11.1.1 HTTP Network Connection

Figure 11-1 : HTTP Network Connection Over Various Stacks
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 ([RFC3986], 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, POST, PUT and DELETE requests. The implementation MUST also support the absolute forms of URIs. Support for PUT and DELETE requests was added to version 3.0 of the MIDP specification.

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 for applications in other than Unidentified Third Party Protection Domains. The application itself is responsible for setting any request headers that it needs. For the MIDlets in the Unidentified Third Party Protection Domain, the `HttpConnection` and `HttpsConnection` MUST include in the User-Agent header the Product-Token "UNTRUSTED/1.0". User-Agent headers supplied by the application MUST NOT be deleted. This behavior for the Unidentified domain applications is specified in the Mobile Service Architecture (JSR 248) specification.
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.)

11.1.2 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.

11.1.3 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 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.

11.1.4 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 Java ME profiles that this device supports. For MIDP 3.0 implementations, this property MUST contain at least &quot;MIDP-3.0&quot;.</td>
</tr>
<tr>
<td>microedition.configuration</td>
<td>The Java ME configuration supported by this device. For example, &quot;CLDC-1.1.&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>

11.1.4.1 HTTP Request Header Example

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

11.2 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.

11.3 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

11.4 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 '[ ]' MUST 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.DatagramPacket
11.5 Push Applications

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

11.6 Serial Port Communications

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

11.7 IP Version Support

When a MIDlet executes on a device that has support for the Internet Protocol (IP), it MAY specify which version of IP to use, version 4, version 6 or either version. The specification of IP version MUST be done at installation time in the manifest and JAD. The attribute MIDlet-Required-IP-Version = value (ipv4|ipv6|any) specifies the IP version all MIDlets in a suite MUST use, or whether both are allowed. If the attribute is not given in the manifest, it means that the MIDlet is able to handle any type of IP addresses. If the implementation can not support the requested version, the MIDlet suite MUST fail to install and return Status Code 905 (Attribute Mismatch) to report the problem.

The attribute MIDlet-Required-IP-Version also has a runtime meaning. It means that the implementation MUST use the specified IP version in all the network connections used by the MIDlet. The implementation MUST NOT allow a MIDlet to make connections to ports that have been dedicated to different IP version usage than what the MIDlet is requiring. If the implementation cannot provide the MIDlet with the required IP version at runtime, an IOException MUST be thrown.

11.8 Inter-MIDlet Communications (IMC)

The MIDP Inter-MIDlet Communication protocol defines a low-level, asynchronous, bi-directional stream connection to support direct communication links between MIDlets. Very similar to the socket protocol, the IMC protocol provides for both client and server connection. When an IMC client makes a connection to a server, the IMC server is identified by MIDlet UID, and the IMC server name and version. The MIDlet UID is the unique identifier of a MIDlet suite, composed of the combination of the MIDlet's name, vendor, and version. The MIDlet UID is omitted in the connection URL strings when opening an IMC server connection. More information about the MIDlet UIDs can be found in Chapter 16.

The Inter-MIDlet Communication protocol MUST be supported by all implementations.

LIBlets enable MIDlet suites to share software components; however, a MIDlet suite and the LIBlets it depends upon are statically bound at installation into a single isolated runtime execution environment. Further, code belonging to a single LIBlet and bound to multiple MIDlet suites' execution environments cannot easily exchange data at runtime.

With the introduction of Inter-MIDlet communication, a MIDlet will be able to access a shareable component running inside another execution environment through thin client APIs, and thus make
possible component-based programming for MIDP applications. The shareable component runs inside another MIDlet execution environment and handles requests from other MIDlets.

An example use case for Inter-MIDlet communication is a media decoder. In this use case, a media player MIDlet invokes a GUI-less decoder MIDlet through an IMC connection to decode a media stream with a content type not supported by the device. These decoder MIDlets are executed as if they were LIBlets bound to the player MIDlet, but can be individually installed or deleted and run inside isolated execution environments.

The following interfaces are defined for Inter-MIDlet communication:

- `javax.microedition.io.IMCConnection`
- `javax.microedition.io.IMCServerConnection`

An IMC server has the ability to restrict the set of clients that are allowed to connect to it. This is achieved by specifying the need for access authorization when a server connection is created. Only clients that match a server MIDlet's access authorization requirements are allowed to connect to the server; see Application Level Access Authorization for more details.

A MIDlet starting an IMC server can be launched in the background when the device is powered on, or at client request. Unlike LIBlet code static sharing, a MIDlet's IMC server can be independently shut down and unloaded from memory if it is no longer serving client requests. An IMC server can either display screens to the user to gather inputs upon receiving a request from a client, or do the processing in the background silently without any GUI, as if it were part of the original MIDlet's execution to the user's experience.

### 11.9 Security of Networking Functions

The MIDP security model 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 MIDP Applications 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 usage may result in charges. MIDP 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.

Connections made using the Generic Connection Framework Connector.open to local addresses on the device do not cause chargeable events and as such SHOULD NOT result in a prompt to the user. This includes access to hostname "localhost", the IP address 127.0.0.1, and other local IP addresses.

Each protocol is accessed by invoking `javax.microedition.io.Connector.open` with a URI including the protocol and arguments. CLDC 1.1.1 defines the permissions that protect access to protocols accessed via `Connector.open`. Devices are NOT REQUIRED to implement every protocol.

### 11.10 Security of PushRegistry

The PushRegistry is protected using the security framework and permissions. The MIDlet suite must have the `javax.microedition.io.PushRegistryPermission` to register an alarm based launch, to register dynamically using the PushRegistry and to make a static registration in the application descriptor. `PushRegistryPermission` also determines if the user needs to be prompted prior to invoking a MIDlet in response to a Push connection event or an alarm. When user is prompted on the event of an alarm or push registration, the implementation SHOULD allow for user to select whether to be prompted or not for application auto-launch.
The interaction modes control when the user is prompted to access the protected `PushRegistry` functionality. Depending on the policy, they user MAY be prompted only once or each time restricted function is called, including the automatic launch of the application. If the user is prompted at installation time to allow static push registration and the user does not accept the prompt, the installation of the MIDlet suite SHOULD fail with status code 910.

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-Permission-1: javax.microedition.io.PushRegistryPermission "socket:" "static,dynamic"
MIDlet-Permission-2: javax.microedition.io.SocketProtocolPermission "socket://"
```

Since: MIDP 1.0
<table>
<thead>
<tr>
<th>Interfaces</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>IMCConnection</td>
<td>This interface defines a low-level asynchronous bi-directional stream connection for the Inter-MIDlet Communication protocol (IMC).</td>
</tr>
<tr>
<td>IMCServerConnection</td>
<td>This interface defines a server connection for the Inter-MIDlet Communication (IMC) protocol.</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>UDP DatagramConnection</td>
<td>This interface defines a datagram connection which knows its local end point address.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Classes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PushRegistry</td>
<td>The PushRegistry maintains a list of inbound connections.</td>
</tr>
<tr>
<td>PushRegistryPermission</td>
<td>The PushRegistryPermission class is used to check the static and dynamic registration of push connections and for registration of an alarm.</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 [RFC3986]) 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 implementation MAY provide access to serial ports through the CommConnection interface. Which serial ports are available to MIDlets is implementation-specific. The valid identifiers for the ports 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>
</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>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>
<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
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

---

```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();
```
### getBaudRate

**public int getBaudRate()**

Gets the baudrate for the serial port connection.

**Returns:**
the baudrate of the connection

**See Also:**
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:**
getBaudRate()
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`)*

```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.
Example using POST with HttpConnection

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

```java
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();
    }
}
```
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);
            }
        }
    } finally {
        if (os != null) os.close();
        if (is != null) is.close();
        if (c != null) c.disconnect();
    }
}
Simplified Stream Methods on Connector

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

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

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

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

Since: MIDP 1.0

Field Summary

<table>
<thead>
<tr>
<th>public static final DELETE</th>
</tr>
</thead>
<tbody>
<tr>
<td>The HTTP DELETE method.</td>
</tr>
<tr>
<td>Value: DELETE</td>
</tr>
</tbody>
</table>

javax.microedition.io.HttpConnection

{ catch (ClassCastException e) { throw new IllegalArgumentException("Not an HTTP URL"); } finally { if (is != null) is.close(); if (os != null) os.close(); if (c != null) c.close(); } }
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>The HTTP GET method.</td>
<td>GET</td>
</tr>
<tr>
<td>HEAD</td>
<td>The HTTP HEAD method.</td>
<td>HEAD</td>
</tr>
<tr>
<td>HTTP_ACCEPTED</td>
<td>202: The request has been accepted for processing, but the processing has not been completed.</td>
<td>202</td>
</tr>
<tr>
<td>HTTP_BAD_GATEWAY</td>
<td>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.</td>
<td>502</td>
</tr>
<tr>
<td>HTTP_BAD_METHOD</td>
<td>405: The method specified in the Request-Line is not allowed for the resource identified by the Request-URI.</td>
<td>405</td>
</tr>
<tr>
<td>HTTP_BAD_REQUEST</td>
<td>400: The request could not be understood by the server due to malformed syntax.</td>
<td>400</td>
</tr>
<tr>
<td>HTTP_CLIENT_TIMEOUT</td>
<td>408: The client did not produce a request within the time that the server was prepared to wait.</td>
<td>408</td>
</tr>
<tr>
<td>HTTP_CONFLICT</td>
<td>409: The request could not be completed due to a conflict with the current state of the resource.</td>
<td>409</td>
</tr>
<tr>
<td>HTTP_CREATED</td>
<td>201: The request has been fulfilled and resulted in a new resource being created.</td>
<td>201</td>
</tr>
<tr>
<td>HTTP_ENTITY_TOO_LARGE</td>
<td>413: The server is refusing to process a request because the request entity is larger than the server is willing or able to process.</td>
<td>413</td>
</tr>
<tr>
<td>HTTP_EXPECT_FAILED</td>
<td>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.</td>
<td>417</td>
</tr>
<tr>
<td>HTTP_FORBIDDEN</td>
<td>403: The server understood the request, but is refusing to fulfill it.</td>
<td>403</td>
</tr>
<tr>
<td>Method</td>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>javax.microedition.io.HttpConnection.HTTP_GATEWAY_TIMEOUT</td>
<td>504</td>
<td>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. Value: <strong>504</strong></td>
</tr>
<tr>
<td>javax.microedition.io.HttpConnection.HTTP_GONE</td>
<td>410</td>
<td>410: The requested resource is no longer available at the server and no forwarding address is known. Value: <strong>410</strong></td>
</tr>
<tr>
<td>javax.microedition.io.HttpConnection.HTTP_INTERNAL_ERROR</td>
<td>500</td>
<td>500: The server encountered an unexpected condition which prevented it from fulfilling the request. Value: <strong>500</strong></td>
</tr>
<tr>
<td>javax.microedition.io.HttpConnection.HTTP_LENGTH_REQUIRED</td>
<td>411</td>
<td>411: The server refuses to accept the request without a defined Content-Length. Value: <strong>411</strong></td>
</tr>
<tr>
<td>javax.microedition.io.HttpConnection.HTTP_MOVED_PERM</td>
<td>301</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. Value: <strong>301</strong></td>
</tr>
<tr>
<td>javax.microedition.io.HttpConnection.HTTP_MOVED_TEMP</td>
<td>302</td>
<td>302: The requested resource resides temporarily under a different URI. Value: <strong>302</strong></td>
</tr>
<tr>
<td>javax.microedition.io.HttpConnection.HTTP_MULT_CHOICE</td>
<td>300</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. Value: <strong>300</strong></td>
</tr>
<tr>
<td>javax.microedition.io.HttpConnection.HTTP_NO_CONTENT</td>
<td>204</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. Value: <strong>204</strong></td>
</tr>
<tr>
<td>javax.microedition.io.HttpConnection.HTTP_NOT_ACCEPTABLE</td>
<td>406</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. Value: <strong>406</strong></td>
</tr>
<tr>
<td>javax.microedition.io.HttpConnection.HTTP_NOT_AUTHORITY</td>
<td>203</td>
<td>203: The returned meta-information in the entity-header is not the definitive set as available from the origin server. Value: <strong>203</strong></td>
</tr>
<tr>
<td>javax.microedition.io.HttpConnection.HTTP_NOT_FOUND</td>
<td>404</td>
<td>404: The server has not found anything matching the Request-URI. Value: <strong>404</strong></td>
</tr>
<tr>
<td>Method</td>
<td>Status Code</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>public static final</td>
<td>HTTP_NOT_IMPLEMENTED</td>
<td>501: The server does not support the functionality required to fulfill the request.</td>
</tr>
<tr>
<td>public static final</td>
<td>HTTP_NOT_MODIFIED</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>public static final</td>
<td>HTTP_OK</td>
<td>200: The request has succeeded.</td>
</tr>
<tr>
<td>public static final</td>
<td>HTTP_PARTIAL</td>
<td>206: The server has fulfilled the partial GET request for the resource.</td>
</tr>
<tr>
<td>public static final</td>
<td>HTTP_PAYMENT_REQUIRED</td>
<td>402: This code is reserved for future use.</td>
</tr>
<tr>
<td>public static final</td>
<td>HTTP_PRECON_FAILED</td>
<td>412: The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server.</td>
</tr>
<tr>
<td>public static final</td>
<td>HTTP_PROXY_AUTH</td>
<td>407: This code is similar to 401 (Unauthorized), but indicates that the client must first authenticate itself with the proxy.</td>
</tr>
<tr>
<td>public static final</td>
<td>HTTP_REQ_TOO_LONG</td>
<td>414: The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret.</td>
</tr>
<tr>
<td>public static final</td>
<td>HTTP_RESET</td>
<td>205: The server has fulfilled the request and the user agent SHOULD reset the document view which caused the request to be sent.</td>
</tr>
<tr>
<td>public static final</td>
<td>HTTP_SEE_OTHER</td>
<td>303: The response to the request can be found under a different URI and SHOULD be retrieved using a GET method on that resource.</td>
</tr>
<tr>
<td>public static final</td>
<td>HTTP_TEMP_REDIRECT</td>
<td>307: The requested resource resides temporarily under a different URI.</td>
</tr>
<tr>
<td>public static final</td>
<td>HTTP_UNAUTHORIZED</td>
<td>401: The request requires user authentication.</td>
</tr>
</tbody>
</table>
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.
Value: 416

public static final 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.
Value: 415

public static final HTTP_USE_PROXY
305: The requested resource MUST be accessed through the proxy given by the Location field.
Value: 305

public static final HTTP_VERSION
505: The server does not support, or refuses to support, the HTTP protocol version that was used in the request message.
Value: 505

public static final POST
The HTTP POST method.
Value: POST

public static final PUT
The HTTP PUT method.
Value: PUT

Method Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>getDate()</td>
<td>Returns the value of the date header field.</td>
</tr>
<tr>
<td>long</td>
<td>getExpiration()</td>
<td>Returns the value of the expires header field.</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getFile()</td>
<td>Returns the file portion of the URL of this HttpURLConnection.</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getHeaderField(int n)</td>
<td>Gets a header field value by index.</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getHeaderField(String name)</td>
<td>Returns the value of the named header field.</td>
</tr>
<tr>
<td>long</td>
<td>getHeaderFieldDate(String name, long def)</td>
<td>Returns the value of the named field parsed as date.</td>
</tr>
<tr>
<td>int</td>
<td>getHeaderFieldInt(String name, int def)</td>
<td>Returns the value of the named field parsed as a number.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td><code>java.lang.String getHeaderFieldKey(int n)</code></td>
<td>Gets a header field key by index.</td>
<td></td>
</tr>
<tr>
<td><code>java.lang.String getHost()</code></td>
<td>Returns the host information of the URL of this <code>HttpConnection</code>, such as the hostname or IPv4 or IPv6 address.</td>
<td></td>
</tr>
<tr>
<td><code>long getLastModified()</code></td>
<td>Returns the value of the <code>last-modified</code> header field.</td>
<td></td>
</tr>
<tr>
<td><code>int getPort()</code></td>
<td>Returns the network port number of the URL for this <code>HttpConnection</code>.</td>
<td></td>
</tr>
<tr>
<td><code>java.lang.String getProtocol()</code></td>
<td>Returns the protocol name of the URL of this <code>HttpConnection</code>.</td>
<td></td>
</tr>
<tr>
<td><code>java.lang.String getQuery()</code></td>
<td>Returns the query portion of the URL of this <code>HttpConnection</code>.</td>
<td></td>
</tr>
<tr>
<td><code>java.lang.String getRef()</code></td>
<td>Returns the fragment part of the URL of this <code>HttpConnection</code>.</td>
<td></td>
</tr>
<tr>
<td><code>java.lang.String getRequestMethod()</code></td>
<td>Get the current request method.</td>
<td></td>
</tr>
<tr>
<td><code>java.lang.String getRequestProperty(String key)</code></td>
<td>Returns the value of the named general request property for this connection.</td>
<td></td>
</tr>
<tr>
<td><code>int getResponseCode()</code></td>
<td>Returns the HTTP response status code.</td>
<td></td>
</tr>
<tr>
<td><code>java.lang.String getResponseMessage()</code></td>
<td>Gets the HTTP response message, if any, returned along with the response code from a server.</td>
<td></td>
</tr>
<tr>
<td><code>java.lang.String getURL()</code></td>
<td>Return a string representation of the URL for this connection.</td>
<td></td>
</tr>
<tr>
<td><code>void setRequestMethod(String method)</code></td>
<td>Set the method for the URL request, one of:</td>
<td></td>
</tr>
<tr>
<td><code>void setRequestProperty(String key, String value)</code></td>
<td>Sets the general request property.</td>
<td></td>
</tr>
</tbody>
</table>

**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

DELETE

public static final java.lang.String DELETE

The HTTP DELETE method.
Constant value: DELETE
Since: MIDP 3.0

GET

public static final java.lang.String GET

The HTTP GET method.
Constant value: GET

HEAD

public static final java.lang.String HEAD

The 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
javax.microedition.io.HttpConnection

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

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.
Constant value: 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.
Constant value: 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.
Constant value: 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. Constant value: 410

**HTTP_INTERNAL_ERROR**

```java
public static final int HTTP_INTERNAL_ERROR = 500;
```

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

**HTTP_LENGTH_REQUIRED**

```java
public static final int HTTP_LENGTH_REQUIRED = 411;
```

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

**HTTP_MOVED_PERM**

```java
public static final int HTTP_MOVED_PERM = 301;
```

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. Constant value: 301

**HTTP_MOVED_TEMP**

```java
public static final int HTTP_MOVED_TEMP = 302;
```

302: The requested resource resides temporarily under a different URI. (Note: the name of this status code reflects the earlier publication of [RFC2068], which was changed in [RFC2616] from "moved temporarily" 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;
```

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;
```

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;
```

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
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
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
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
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

The HTTP POST method. Constant value: POST

PUT

public static final java.lang.String PUT

The HTTP PUT method. Constant value: PUT

Since: MIDP 3.0

Methods

getDate

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

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

public java.lang.String getFile()  

Returns the file portion of the URL of this HttpConnection. [RFC3986] defines the path component in the URI, that path value is returned from getFile.

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

getHeaderField

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

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
javax.microedition.io.HttpConnection

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

public java.lang.String getHost()

Returns the host information of the URL of this HttpConnection, such as the hostname or IPv4 or IPv6 address.

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

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

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

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

public java.lang.String getQuery()

Returns the query portion of the URL of this HttpConnection. [RFC3986] defines the query component as the text after the first question-mark (?) character in the URL and before a number sign (#) or the end of the URL.

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

getRef

public java.lang.String getRef()

Returns the fragment part of the URL of this HttpConnection. [RFC3986] specifies the optional fragment identifier as 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 of the retrieved information (see [RFC2046]).

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

getRequestMethod

public java.lang.String getRequestMethod()

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

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: setRequestProperty(String, String)

generateResponseCode

public int generateResponseCode()

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.

generateResponseMessage

public java.lang.String generateResponseMessage()

throws java.io.IOException

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
- PUT
- DELETE

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: 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: getRequestMethod(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. [RFC2818] 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 [RFC2246] (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 [RFC2246], WTLS as specified in Wireless Transport Layer Security Specification, WAP-261-WTLS, WAP Forum, and TLS Profile and Tunneling Specification as specified in WAP-219-TLS, WAP Forum, MAY also be supported.

HTTPS is the secure version of HTTP ([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

DELETE, 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, HTTPMoved_PERM, HTTPMOVED_TEMP, HTTP_MULT_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,
HTTPPRECON_FAILED, HTTP_PROXY_AUTH, HTTPREQ_TOO_LONG, HTTP_RESET, HTTPSEE_OTHER,
HTTP_TEMP_REDIRECT, HTTP_UNAUTHORIZED, HTTP_UNAVAILABLE, HTTP_UNSUPPORTED_RANGE,
HTTP_UNSUPPORTED_TYPE, HTTP_USE_PROXY, HTTP_VERSION, POST, PUT
javax.microedition.io.HttpsConnection

int getPort()

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

javax.microedition.io.SecurityInfo getSecurityInfo()

Returns the security information associated with this successfully opened connection.

Methods inherited from interface javax.microedition.io.HttpConnection

gdate, getExpiration, getFile, getHeaderField, getHeaderField, getHeaderFieldDate, getHeaderFieldInt, getHeaderFieldKey, getHost, getLastModified, getProtocol, getQuery, getRef, getRequestMethod, getRequestProperty, getResponseCode, getResponseMessage, getUrl, setRequestMethod, setRequestProperty

Methods inherited from interface ContentConnection

gencoding, 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

gPort

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.

gSecurityInfo

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
IMCConnection

Declaration

public interface IMCConnection extends StreamConnection

Description

This interface defines a low-level asynchronous bi-directional stream connection for the Inter-MIDlet Communication protocol (IMC).

An instance of this interface is returned from Connector.open() when it is called with an IMC client URI string. An IMCConnection instance is also returned from IMCServerConnection.acceptAndOpen() for incoming client connection.

An IMC server MIDlet can register a server connection with push registry and be automatically launched upon client connection request. An implementation MUST support registration of IMC server MIDlets for application launch on incoming IMC connection. If the AMS fails to launch the IMC server MIDlet due to the system limit of maximum number of concurrent MIDlets has been reached or the user denies such auto-launch request or any other reasons, a ConnectionNotFoundException MUST be thrown from Connector.open() to the IMC client MIDlet.

BNF Format for Connector.open() string

The IMC client connection URIs must conform to the BNF syntax defined below. If a client URI passed into Connector.open() does not conform to this syntax, an IllegalArgumentException is thrown.

<table>
<thead>
<tr>
<th>&lt;client_conn_string&gt;</th>
<th>::= &quot;imc://&quot; (&lt;MIDlet UID&gt;</th>
<th>&lt;MIDlet suite vendor&gt;</th>
<th>&lt;MIDlet suite name&gt;</th>
<th>&lt;MIDlet suite version&gt;</th>
<th>&lt;server name&gt;</th>
<th>&lt;server version&gt;</th>
<th>&lt;authmode&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;MIDlet UID&gt;</td>
<td>::= &lt;MIDlet suite vendor&gt; : &quot; &lt;MIDlet suite name&gt; : &quot; &lt;MIDlet suite version&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;MIDlet suite vendor&gt;</td>
<td>::= MIDlet suite vendor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;MIDlet suite name&gt;</td>
<td>::= MIDlet suite name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;MIDlet suite version&gt;</td>
<td>::= MIDlet suite version or wildcard character &quot;***&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;server name&gt;</td>
<td>::= IMC server name following the class naming syntax</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;server version&gt;</td>
<td>::= version of the IMC server. Version backward compatibility is assumed. Versioning follows the format defined for the MIDlet-Version attribute.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;authmode&gt;</td>
<td>::= authmode=true</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The BNF syntax allows the wildcard "***" to be used instead of a specific MIDlet UID when opening an IMC client connection. When the wildcard character is used, it allows the client to connect to any of the MIDlets from different vendors if they all provide the same IMC service and meet the authorization requirements. However, which MIDlet's IMC server the client will be connected is implementation specific.

An IMC server may restrict access to a authorized clients only using Application level access authorization. See IMCServerConnection for further details. If the client connection is not authorized to connect to the server, the Connector.open(..) call MUST throw a SecurityException. Similarly, an IMC Client connection can specify that it connect only to authorized servers. Authorized servers are identified using the Application level access authorization mechanisms. The MIDlet that is making the client connection declares it authorization attributes that identify the server as authorized. Only authorized servers will cause a Connector.open(..) call from a client to succeed.
Examples

// Create a connection to any MIDlet providing the com.foo.services.barServer
IMCConnection conn = (IMCConnection)Connector.open(
   "imc://*:com.foo.services.barServer:1.0;authmode=false");

try {
   DataOutputStream request = conn.openDataOutputStream();

   // writes request data
   ...

   // flush the request input stream
   request.flush();

   DataInputStream response = conn.openDataInputStream();

   // retrieve response data
   ...
} finally {
   conn.close();
}

Since: MIDP 3.0

Method Summary

<table>
<thead>
<tr>
<th>javax.microedition.midlet.MIDletIdentity</th>
<th>getRemoteIdentity()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the identity of the MIDlet on the remote end of this connection</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getRequestedServerVersion()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the server version the IMC client requested to connect to in the client connection URI</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getServerName()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the name of the IMC server of this connection.</td>
<td></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

**getRemoteIdentity**

```java
public javax.microedition.midlet.MIDletIdentity getRemoteIdentity()
```

Get the identity of the MIDlet on the remote end of this connection

**Returns:**
MIDlet identity of the other end of the connection

**getRequestedServerVersion**

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

Get the server version the IMC client requested to connect to in the client connection URI

**Returns:**
The requested server version

**getServerName**

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

Get the name of the IMC server of this connection. The name is the server name specified in the connection URI.

**Returns:**
Server name of the IMC connection
javax.microedition.io
IMCServerConnection

Declaration

public interface IMCServerConnection extends StreamConnectionNotifier

Description

This interface defines a server connection for the Inter-MIDlet Communication (IMC) protocol. A server connection is obtained through Connector.open() with the server connection URI string according to the BNF syntax defined below.

The acceptAndOpen() method returns an IMCConnection instance. In addition to the normal behavior of a StreamConnection, the IMCConnection allows the IMC server to get the requested server version and the client MIDlet's identity. The call to IMCServerConnection.acceptAndOpen() is synchronous and the current thread will be blocked until a client connection arrives or some exception is thrown. If multiple connection requests arrive at the same time, the remaining connections will be queued up by the implementation to be returned by the next acceptAndOpen() call.

In the connection URI string, the IMC server can specify authorization requirements for the IMC clients. If authorization is required (by specifying authmode=true in the connection URI), only the connection requests from access authorized MIDlets will be accepted. See Application Level Access Authorization for more details. If authmode=false is specified or if this does not appear in the connection URI, the server connection will accept connection from any client that meets the name and version requirements.

BNF Format for Connector.open() string

The IMC server connection URIs must conform to the BNF syntax defined below. If the IMC server connection URI does not conform to this syntax, an IllegalArgumentException is thrown.

| <server_conn_string> | ::= "imc:// " :<server name> ":" <server version> ";" [authmode] |
| <server name>       | ::= IMC server name following the class naming syntax |
| <server version>    | ::= version of the IMC server. Backward compatibility is assumed. Versioning follows the format defined for the MIDlet-Version attribute |
| <authmode>          | ::= authmode=true|false |

The IMC server connection URI is used for the connection URL for registering a MIDlet for auto launch (as part of the MIDlet-Push-<n> attribute). The AllowedSender is either "***" or MIDlet_UID. See IMCConnection for description of MIDlet_UID. IMC Servers may also be registered dynamically using the PushRegistry.registerConnection() API. Here is an example on registering a IMC Server at installation time,

```
MIDlet-Push-1:imc://:com.foo.barServer:1.0;authmode=true, MyIMCServer, *
```

An IMC server is assumed to be always backward compatible with earlier versions. An IMC client can be connected to a server if the version requested is same or lower than the declared version in the server connection string. An IMC server can get the requested server version of an established connection through IMCConnection.getRequestedServerVersion() method and respond to the client accordingly based on the version requested.
Examples

```
IMCServerConnection serverConn = (IMCServerConnection) Connector.open(
    "imc://:com.foo.barServer:1.0;authmode=true);

while(true) {
    IMCConnection conn = (IMCConnection) serverConn.acceptAndOpen();

    try {
        String requestedVersion = conn.getRequestedServerVersion();
        DataInputStream requestData = conn.openDataInputStream();
        DataOutputStream responseData = conn.openDataOutputStream();

        // read request data from requestData input stream
        int requestType = requestData.readInt();
        ...

        // Handle the request
        ...

        // write response into the output stream
        responseData.writeUTF("Hello! This is the barServer");
        responseData.close();
    } finally {
        conn.close();
    }
}
```

**Since:** MIDP 3.0

### Method Summary

<table>
<thead>
<tr>
<th>Method Type</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>java.lang.String</code></td>
<td><code>getName()</code></td>
<td>Get the name of the IMC server</td>
</tr>
<tr>
<td><code>java.lang.String</code></td>
<td><code>getVersion()</code></td>
<td>Get the version of the IMC server</td>
</tr>
</tbody>
</table>

**Methods inherited from interface** `StreamConnectionNotifier`

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>acceptAndOpen</code></td>
</tr>
</tbody>
</table>

**Methods inherited from interface** `Connection`

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>close</code></td>
</tr>
</tbody>
</table>

### Methods
**getName**

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

Get the name of the IMC server

**Returns:**

The name of the IMC server

---

**getVersion**

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

Get the version of the IMC server

**Returns:**

The version of the IMC server
javax.microedition.io

PushRegistry

Declaration

public class PushRegistry

Object

+-javax.microedition.io.PushRegistry

Description

The PushRegistry maintains a list of inbound connections. A MIDlet can register the inbound connections with an entry in the application descriptor or dynamically by calling the registerConnection method. The connection strings are URIs that are used with Connector.open to open the appropriate server connection.

While a MIDlet is running and has Connections open, it is responsible for all I/O operations associated with the inbound connection using the appropriate Generic Connection Framework API. When the MIDlet is not running or and MIDlet does not have the URI open, the application management software (AMS) listens for inbound notification requests. When a notification arrives for a registered MIDlet, the AMS will start the MIDlet, if necessary, via the normal invocation of MIDlet.startApp method.

Implementations MUST guarantee that each inbound connection successfully registered (statically or dynamically) is logically unique. The logical uniqueness is determined by using the comparison ladder scheme as defined in [RFC3986]. Implementations MUST perform at least the simple string comparison, and SHOULD perform one or more of the latter steps of the comparison ladder scheme.

Installation Handling of Declared Connections

To avoid collisions on inbound generic connections, the application descriptor 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 MIDP 3.0 Provisioning 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 not be fulfilled, it must register the Push connections using the dynamic registration with PushRegistry.registerConnection method.

A conflict-free installation reserves each requested connection for the exclusive use of a MIDlet in the suite. While the suite is installed, any attempt by other MIDlets 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 the MIDlet will always succeed, assuming the MIDlet does not already have the connection open.

If two MIDlet Suites have a static push connection in common, they can not be installed together. The end user would typically have to delete one before being able to successfully install the other.

Static Push Registration

In some cases the MIDlet may not function properly if it cannot listen to a certain protocol or port for incoming traffic. The static push registration has been designed for these cases. The MIDlet that must have access to certain protocol or port announces this need in the application descriptor or in the JAR manifest. The implementation check already at installation time that the requested protocol or port is
available and registers the MIDlet to listen to the incoming traffic. If the request cannot be fulfilled because protocol or port is already reserved, the installation of the application fails.

Static Push registrations are done in the application descriptor or in the JAR manifest with MIDlet-Push-<n> attribute. Each push registration entry contains the following information: MIDlet-Push-<n>:<ConnectionURI>, <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.
- ConnectionURI = the connection string used in Connector.open()
- MIDletClassName = the MIDlet that is responsible for the connection. The named MIDlet MUST be registered in the application descriptor or the JAR manifest with a MIDlet-<n> record. (This information is needed when displaying messages to the user about the MIDlet when Push connections are detected, or when the user grants/revokes privileges for the MIDlet.) 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 AllowedSender field is just the wildcard character "*", connections will be accepted from any originating source. For Push attributes using the datagram and socket URIs (if supported by the platform), AllowedSender field contains a numeric IP address in the same format for IPv4 and IPv6 as used in the respective URIs (IPv6 address including the square brackets as in the URI). It is possible to use the wildcards also in these IP addresses, e.g. "129.70.40."" would allow subnet resolution. The wildcards can also be used to match address delimiters, e.g. "72.5.1" will match "72.5.124.161". 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 AllowedSender 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(?).

This 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 URI string.

Example Application Descriptor Declarative Notation

The following is a sample application descriptor 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]

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

Dynamic Push Registration from a Running MIDlet

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

To accommodate this type of MIDlet, a mechanism is provided to dynamically allocate a connection and to register that information, as if it was known, when the MIDlet 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 MIDlet is requesting a dynamic port to be allocated from the ports that are currently available. By calling `PushRegistry.registerConnection()` method 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). Once the MIDlet has registered the connection with `PushRegistry.registerConnection` method, the connection is listed in the `PushRegistry.listConnections(false)` return value. If the MIDlet has an open connection to the registered connection, the AMS starts listening to the inbound connection once the connection has been closed with `Connector.close` method. If the MIDlet is deleted from the phone, then its dynamic communication connections are unregistered automatically.

### 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, a MIDlet 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 MIDlets (if necessary) prior to the invocation of the push MIDlet.

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

If a MIDlet has finished with all inbound data it MAY the connection with `Connector.close()` method. If the connection is closed, then the AMS resumes listening for Push notifications. This avoids the loss of data that might occur if neither the MIDlet nor the AMS was listening.

When a registered Push MIDlet is not running the AMS listens for incoming connections and launches the MIDlet as necessary. If a Push MIDlet exits and there are incoming connections, either new or unhandled, for the MIDlet, then the MIDlet MUST be started to handle the available input.

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

### AMS Runtime Handling

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. Once the installation has been successfully completed, (e.g. via provisioning - when the Installation notification message has been successfully transmitted, the MIDlet 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 MIDlet uses `acceptAndOpen()` to get the socket connected and for a datagram connection the MIDlet 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 MIDlet. The MIDlet life cycle defines the expected behaviors that an interrupted MIDlet could see from a call to `destroyApp()`.

**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 MUST be available to the MIDlet when the MIDlet opens the `UDPDatagramConnection` after startup.

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

**Connection vs Push Registration Support**

Not all generic connections will be appropriate for use as Push application transport. Even if a protocol is supported on the device as an inbound connection type, it is not required to be enabled as a valid Push mechanism. E.g. a platform might support server socket connections in a MIDlet, but might not support inbound socket connections for Push launch capability. A `ConnectionNotFoundException` is thrown from the `registerConnection` method when the platform does not support that optional capability. The `registerAlarm` method MUST be supported.

**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 Application Descriptor**

In this sample, the application descriptor 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.
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 application descriptor.
        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. ...
    }

    // Inner class to handle inbound messages on a separate thread.
    class MessageHandler implements Runnable {
        String connUrl;
        MessageHandler(String uri) {
            connUri = uri;
        }

        // Fetch messages in a blocking receive loop.
        public void run() {
            try {
                // Get a connection handle for inbound messages
                // and a buffer to hold the inbound message.
                DatagramConnection conn =
                    (DatagramConnection) Connector.open(connUri);
                Datagram data = conn.newDatagram(conn.getMaximumLength());

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

Sample Ping Application Descriptor

In this sample, the application descriptor includes an entry indicating that the MIDlet 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 an inbound connection notification.

Sample Ping MIDlet Processing
public class SamplePingMe extends MIDlet {
    // Name of the current MIDlet for push registration.
    String myName = "example.chat.SamplePingMe";

    // List of registered push connections.
    String connections[];

    // Inbound datagram connection
    UDPEndPoint 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 = (UDPEndPoint) 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 = (UDPEndPoint) 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

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

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 - a registered generic connection URI string

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 or if the connection argument was null

See Also: registerConnection(String, String, String)

getMIDlet

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

Retrieve the registered MIDlet for a requested connection.

Parameters:
connection - a registered generic connection URI string

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 or if the connection argument was null

See Also: registerConnection(String, String, String)

listConnections

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

Return a list of registered connections for the current MIDlet.

The URI of every registered connection is returned from listConnections(false).

The URI of every connection that has available input is returned from listConnections(true). URIs of connections opened with Connector.open(URI) are not returned. After the Connection is closed new input may become available and the URI will again be included in the return of listConnections(true). URIs of connections that timeout or otherwise no longer have available input are not returned from listConnections(true). Due to race conditions, a call to listConnections(true) may return URIs that will fail to open with Connector.open because timeouts or other connection errors.

When the MIDlet opens the URI, the MIDlet takes over listening for input and the AMS stops listening. The listConnections(true) method will only see URIs with available input during the time that the MIDlet does NOT have the connection open.

Parameters:
available - if true, only return the list of connections ready for the handoff to the MIDlet, otherwise return the complete list of registered connections for the current MIDlet

Returns:
array of registered connection strings, where each connection is represented by the generic connection URI string.

registerAlarm

public static long registerAlarm(String midlet, long time)

throws java.lang.ClassNotFoundException,
javax.microedition.io.ConnectionNotFoundException
Register a time to launch the specified MIDlet. The PushRegistry supports one outstanding wake up time per MIDlet in the current suite. A MIDlet is expected to use java.util.TimerTask for notification of time based events while the MIDlet is running.

If a wakeup time was registered and is still pending, the wakeup time will be returned, otherwise zero is returned. If the wakeup time has passed then the wakeup is no longer pending and zero is returned.

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 application descriptor or the JAR 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(). If the time is zero, or is in the past, or the MIDlet is already running at the time then the MIDlet MUST not be launched.

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 application descriptor or the JAR manifest or if the midlet argument is null
- SecurityException - if the midlet does not have permission to register an alarm

See Also: Date.getTime()

registerConnection

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

While the MIDlet has opened the connection with Connector.open(), the AMS will NOT be listening for input. The MIDlet is responsible for the connection. If the MIDlet has not opened a connection to the registered URI, the AMS MUST listen for input regardless of whether the MIDlet is running or not.

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

Parameters:
- connection - the URI for the connection
- midlet - class name of the midlet to be launched when new external data is available. The named midlet MUST be registered in the application descriptor or the JAR manifest with a MIDlet-<n> record. This parameter has the same semantics as the MIDletClassName in the Push registration attribute defined in the class description.
- filter - a connection URI string indicating which senders are allowed to cause the midlet to be launched

Throws:
- IllegalArgumentException - if the connection string is null or is not valid, or if the filter string is null or not valid
- ConnectionNotFoundException - if the runtime system does not support push delivery for the requested connection protocol
unregisterConnection

public static boolean unregisterConnection(String connection)

Remove a dynamic connection registration.

Parameters:
  connection - the URI for the connection

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: unregisterConnection(String)
javax.microedition.io

PushRegistryPermission

Declaration

public class PushRegistryPermission extends Permission

Object

+--Permission

+--javax.microedition.io.PushRegistryPermission

Description

The PushRegistryPermission class is used to check the static and dynamic registration of push connections and for registration of an alarm. The permission covers static registration via application attributes, and dynamic registration via `PushRegistry.registerConnection` and alarm registration with `PushRegistry.registerAlarm`.

For the purposes of Push Registration permission, the URI MUST consist only of the scheme and delimiter (":") as defined by [RFC3986]. The scheme may contain the wildcard character "*", which allows registration of all schemes. For alarm registration, the uri is "*" and the action is `alarm`. Push registration and alarm registration can be combined in a single permission. For example, uri="file:" and actions= "static,dynamic,alarm".

The permissible actions and descriptions are listed in the Actions Table:

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static</td>
<td>Allows registration of a Push Connection in the packaging of the MIDlet Suite</td>
</tr>
<tr>
<td>dynamic</td>
<td>Allows registration of a Push Connection using <code>PushRegistry.registerConnection</code></td>
</tr>
<tr>
<td>alarm</td>
<td>Allows registration of an alarm using <code>PushRegistry.registerAlarm</code></td>
</tr>
</tbody>
</table>

Since: MIDP 3.0

See Also: `PushRegistry.registerConnection`, `PushRegistry.registerAlarm`

Constructor Summary

| public PushRegistryPermission(String uri, String actions) |
| Creates a new instance of PushRegistryPermission. |

Method Summary

| boolean equals(Object object) |
| Checks if another object is equal to this one. |

| java.lang.String getActions() |
| Returns the canonical string representation of the actions. |

| int hashCode() |
| Gets the hash code value for this object. |
javax.microedition.io.PushRegistryPermission

<table>
<thead>
<tr>
<th>boolean implies(Permission p)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checks if the specified permission is &quot;implied&quot; by this object.</td>
</tr>
</tbody>
</table>

Methods inherited from class Permission

getName, newPermissionCollection, toString

Methods inherited from class Object

getClass, notify, notifyAll, toString, wait, wait, wait

Constructors

PushRegistryPermission

public PushRegistryPermission(String uri,
String actions)

Creates a new instance of PushRegistryPermission. The URI MUST include only the scheme and delimiter (":" as defined by [RFC3986]) or the wildcard character ">" and actions as listed above. The URI must be valid regardless of the actions supplied. For example, a URI of "*" would be used with the "alarm" action.

Parameters:
uri - The resource name is the URI of the incoming connection; the URI MUST contain only the scheme and the delimiter ":" or wildcard ">".
actions - one or more of the actions listed in the Actions Table, separated by "," if more than 1.

Throws:
IllegalArgumentException - If the URI contains anything other than the non-empty scheme and the delimiter ":" or wildcard ">".
IllegalArgumentException - if actions contains actions that are not listed in the Actions Table.

Methods

equals

public boolean equals(Object object)

Checks if another object is equal to this one.

Parameters:
object - an object to compare

Returns:
true if the objects are of the same type and the respective names and actions are equal.

getActions

public java.lang.String getActions()

Returns the canonical string representation of the actions. This method always returns actions in the following order: static, dynamic, alarm. For example, if this permission object allows static, dynamic, and alarm actions, a call to getActions must return the string "static,dynamic,alarm".

Returns:
the canonical form of the actions.
hashCode

public int hashCode()

Gets the hash code value for this object.

Returns:
    the sum of the hashcode of getName + the hashcode of getActions.

implies

public boolean implies(Permission p)

Checks if the specified permission is "implied" by this object. Returns true iff:

• p's class is the same as this Object's class, and
• p's name(URI) equals this Object's name (URI) or this Object's name equals "*" and
• p's actions are a proper subset of this object's actions.

Parameters:
    p - the permission to check against.

Returns:
    true if the specified permission is implied by this object, otherwise false.
javax.microedition.io

SecureConnection

Declaration

public interface SecureConnection extends SocketConnection

All Superinterfaces:
    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 or IPv6 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 [RFC2246] (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 [RFC2246] and WAP(TM) TLS Profile and Tunneling Specification as specified in WAP-219-TLS-2001041T-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("\n\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>Return the security information associated with this connection when it was opened.</td>
<td></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

Methods

getSecurityInfo

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

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

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String getCipherSuite()</td>
<td>Returns the name of the cipher suite in use for the connection.</td>
</tr>
<tr>
<td>java.lang.String getProtocolName()</td>
<td>Returns the secure protocol name.</td>
</tr>
<tr>
<td>javax.microedition.pki.Certificate getServerCertificate()</td>
<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 [RFC2246]. 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:
javax.microedition.io.SecurityInfo

- a String containing the secure protocol identifier; if TLS ([RFC2246]) 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

```java
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

```java
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.

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

Examples

The following examples show how a ServerSocketConnection would be used to access a sample loopback program.
// Create the server listening socket for port 1234
ServerSocketConnection scn =
(ServerSocketConnection)Connector.open("socket://:1234");

// Wait for a connection.
SocketConnection sc = (SocketConnection)scn.acceptAndOpen();

// Set application specific hints on the socket.
sc.setSocketOption(DELAY, 0);
sc.setSocketOption(LINGER, 0);
sc.setSocketOption(KEEPALIVE, 0);
sc.setSocketOption(RCVBUF, 128);
sc.setSocketOption(SNDBUF, 128);

// Get the input stream of the connection.
DataInputStream is = sc.openDataInputStream();

// Get the output stream of the connection.
DataOutputStream os = sc.openDataOutputStream();

// Read the input data.
String result = is.readUTF();

// Echo the data back to the sender.
os.writeUTF(result);

// Close everything.
is.close();
os.close();
sc.close();
scn.close();
...

Since: MIDP 2.0

### Method Summary

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

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

Gets the local address to which the socket is bound. The address MUST be returned in the format of the requested IP version. The required IP version is specified by the MIDlet with MIDlet-Required-IP-Version manifest attribute. If the attribute is not specified, the implementation MAY decide the format in which the local address is returned.

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

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: 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 or IPv6 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.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;socket_connection_string&gt;</code></td>
<td>::= &quot;socket://&quot;&lt;hostport&gt;</td>
</tr>
<tr>
<td><code>&lt;hostport&gt;</code></td>
<td>::= host:&quot;port</td>
</tr>
<tr>
<td><code>&lt;host&gt;</code></td>
<td>::= host name or IP address (omitted for inbound connections. See ServerSocketConnection)</td>
</tr>
<tr>
<td><code>&lt;port&gt;</code></td>
<td>::= numeric port number</td>
</tr>
</tbody>
</table>

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</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final DELAY</td>
<td>Socket option for the small buffer writing delay (0). Value: 0</td>
</tr>
<tr>
<td>public static final KEEPALIVE</td>
<td>Socket option for the keep alive feature (2). Value: 2</td>
</tr>
<tr>
<td>public static final 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>public static final RCVBUF</td>
<td>Socket option for the size of the receiving buffer (3). Value: 3</td>
</tr>
<tr>
<td>public static final 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>
<tr>
<td>int getPort()</td>
<td>Returns the remote port to which this socket is bound.</td>
</tr>
<tr>
<td>int getSocketOption(byte option)</td>
<td>Get a socket option for the connection.</td>
</tr>
</tbody>
</table>
### Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DELAY</strong></td>
<td><img src="delay" alt="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" /></td>
</tr>
<tr>
<td><strong>KEEPALIVE</strong></td>
<td><img src="keepalive" alt="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" /></td>
</tr>
<tr>
<td><strong>LINGER</strong></td>
<td><img src="linger" alt="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" /></td>
</tr>
<tr>
<td><strong>RCVBUF</strong></td>
<td><img src="rcvbuf" alt="Socket option for the size of the receiving buffer (3)." /></td>
</tr>
<tr>
<td><strong>SNDBUF</strong></td>
<td><img src="sndbuf" alt="Socket option for the size of the sending buffer (4)." /></td>
</tr>
</tbody>
</table>

```java
void setSocketOption(byte option, int value)

Set a socket option for the connection.
```

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

### getAddress

```java
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

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

Gets the local address to which the socket is bound. The address MUST be returned in the format of the requested IP version. The required IP version is specified by the MIDlet with MIDlet-Required-IP-Version manifest attribute. If the attribute is not specified, the implementation MAY decide the format in which the local address is returned.

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

```java
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

```java
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:** `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:** `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.

Since: MIDP 2.0

Method Summary

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getLocalAddress()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets the local address to which the datagram connection is bound.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int</th>
<th>getLocalPort()</th>
</tr>
</thead>
<tbody>
<tr>
<td></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**

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

Gets the local address to which the datagram connection is bound. The address MUST be returned in the format of the requested IP version. The required IP version is specified by the MIDlet with `MIDlet-Required-IP-Version` manifest attribute. If the attribute is not specified, the implementation MAY decide the format in which the local address is returned.

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**

```java
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`
Package
javax.microedition.lcdui

Description
The APIs in the LCDUI package provide a set of features for implementing user interfaces in MIDP applications.

Unless otherwise noted, passing a null argument to a constructor or method in any class or interface in this package MUST cause a NullPointerException to be thrown.

12.1 User Interface

The User Interface features in MIDP have been specifically designed 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 with the native applications so that the user finds them easy to use.

In addition, these devices may have limited memory and processing power. Since the user interface is a primary consumer of such resources, the API's have been designed to avoid the creation of garbage objects and other performance issues wherever possible.

12.2 Structure of the MIDP UI API

The API is logically composed of two sets of APIs: the high-level and the low-level.

The high-level API is designed for applications where portability across devices is important. To achieve this portability, the high-level API employs a high level of abstraction and provides less control over the look and feel of the user interface. This abstraction is further manifested in the following ways:

• The actual drawing to the MID's display is performed by the implementation. Applications have limited influence over 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 does not have control over 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. 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 from keys and touchscreens.

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 use of the low-level API involves details that are specific to a particular device. It is recommended that applications using low-level API be written such that they can adapt to different device characteristics wherever 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.

### 12.2.1 Class Hierarchy

The `Display` class represents a given MIDlet's use of a particular display device and it also provides methods to retrieve information about that display device's capabilities. For each active MIDlet, a `Display` object may be obtained for each display device; a primary `Display` object is provided for accessing the device's main display. In addition to the primary `Display`, there may be other display hardware that is an integral part of device, such as the second screen on the outside of the mobile phone's flip; these display hardware are classified as Built-In displays. There may be other display resources that are available to the device via a suitable connection, these are classified as Auxiliary displays. In addition to the primary `Display`, one or more Built-in and Auxiliary displays may be available to the MIDlet. The `javax.microedition.lcdui.Display` class provides more details on accessing `Display` objects.

The `Display` class is responsible for controlling access to a display device if multiple MIDlets are trying to use simultaneously. `Display` objects have a state that indicates their relative priority for using the display device.

The main abstraction of the UI is a `Displayable` object, which encapsulates device-specific graphics rendering with user input. Only one `Displayable` may be shown at a time on a given `Display`, and the user can see and interact with only contents of that `Displayable`. A `Displayable` is made visible by calling the `setCurrent()` method of the appropriate `Display`. When a `Displayable` is made current, it replaces the previous `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.

### 12.2.2 Class Overview

Many 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 the Form class 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.

A default layout scheme is provided for laying out the Items in a Form, but the developer may implement a custom layout scheme by creating a subclass of FormLayoutPolicy.

### 12.2.3 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 may assume the following conditions from the application management software:

- `getDisplay()` and `getDisplays()` are callable starting from the MIDlet's constructor until `destroyApp()` has returned.
- The `Display` objects for built-in display devices are the same until `destroyApp()` is called.
- The `Displayable` object set by `setCurrent()` is not changed by the application manager.

The application manager assumes the following application behavior with respect to the MIDlet events:
• The application may call `setCurrent()` to display its first screen at any point after its constructor has been called. However, the `Displayable` will be shown by the application manager only after `startApp()` returns.
• `destroyApp` - The application should release resources and objects.

12.3 Event Handling

User interaction generates events, and the implementation notifies the application of the events by making corresponding callbacks. There are four kinds of UI callbacks:

• Abstract commands that are part of the high-level API
• Low-level events that represent single key presses and releases (and pointer events, if a pointer is available)
• Calls to the `paint()` method of a `Canvas` class
• Calls to a `Runnable` object's `run()` method requested by a call to `callSerially()` of class `Display`

All UI callbacks are serialized, so they will never occur in parallel. That is, the implementation will never call an callback before a prior call to any other callback has returned. This property enables applications to be assured that processing of a previous user event will have completed before the next event is delivered. If multiple UI callbacks are pending, the next is called as soon as possible after the previous UI callback returns. The implementation also guarantees that the call to `run()` requested by a call to `callSerially()` is made after any pending repaint requests have been satisfied.

There is one exception to the callback serialization rule, which occurs when the `Canvas.serviceRepaints` 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`
javax.microedition.lcdui - User Interface for MIDP

- CustomItem.pointerPressed
- CustomItem.pointerReleased
- CustomItem.showNotify
- CustomItem.sizeChanged
- CustomItem.traverse
- CustomItem.traverseOut
- Displayable.sizeChanged
- DisplayListener.displayAdded
- DisplayListener.displayStateChanged
- DisplayListener.hardwareStateChanged
- DisplayListener.orientationChanged
- DisplayListener.sizeChanged
- FormLayoutPolicy.doLayout
- IdleItem.addedToDisplay
- IdleItem.removedFromDisplay
- ItemCommandListener.commandAction
- ItemStateListener.itemStateChanged
- ItemTraversalListener.itemTraversedIn
- ItemTraversalListener.itemTraversedOut
- Runnable.run resulting from a call to Display.callSerially
- TabListener.tabChangeEvent

Note that java.util.Timer events are not considered UI events. Timer callbacks may run concurrently with UI event callbacks, although java.util.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.

12.3.1 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. There are two methods for deciding where Commands are placed: native style (default) and exact placement (introduced in MIDP 3.0).

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 exact placement method lets the application developer specify exact placement of Commands and Menus on a Displayable, when this is appropriate (for example, placement of soft buttons on a screen, or associating a Command with an offscreen key). The normal placements of commands are available from a Display. The available placements for soft keys and the location of the labels can be retrieved from the Displayable. The choice of exact placement is made by the Displayable object adding the Command or Menu object by adding an optional placement attribute to the addCommand() or addMenu() method.
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.

In MIDP 3.0 Commands are mutable, and its attributes may change at any time. It is up to the implementation to act as soon as possible on the change.

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.
- **Enabled:** Used to enable/disable a Command that is disabled will typically remain visible, but greyed out, and cannot be chosen.

### 12.3.2 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.

12.3.3 Jog Dial Interaction

A device may 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. Key events generated by directional movements may use different key codes to reflect different spatial orientations.
- An implementation MUST map key press events generated by jog dial movements to 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, UP events are always generated when the jog dial is rolled upwards from the user point of view). Consequently, an implementation MAY need to alter its mapping of key events to game actions depending on the spatial orientation of the device. In the same vein, a clamshell device MAY generate different jog dial events when the clamshell is in OPEN or CLOSED mode.
- Since the mapping of key events and game actions may change in response to orientation changes, application should call Canvas.getGameAction() for each event to ensure that the current mapping is used. Applications may also register a DisplayListener to be notified when the orientation changes and the key code mappings may be revised.
- A 3-way jog dial MUST generate key events with key codes mapping to game actions UP, DOWN and FIRE, or game actions LEFT, RIGHT and FIRE, depending on the relative orientation currently in effect.
- 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).
12.3.4 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:

```java
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

```java
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.

A listener interface is also provided for events related to focus traversal between Items in a Form. The methods

```java
void itemTraverseIn(Item item);
void itemTraverseOut(Item item);
```

defined in interface ItemTraversalListener are called when an Item gains or loses focus, respectively.

12.3.5 Low-Level API for Events

Low-level graphics and events have the following methods to handle low-level key events:

```java
public void keyPressed(int keyCode);
public void keyReleased(int keyCode);
public void keyRepeated(int keyCode);
```

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:

```java
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:

```java
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 action, in which case the `getCardCode` 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:
                ... 
        }
    }
}
```

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();
```

Some devices may support multi-touch user interfaces (i.e. they can detect and track multiple simultaneous touch points instead of a single 'pointer' location). Since applications cannot distinguish between the different touch points using the MIDP APIs, the delivery of multiple simultaneous touch events has the potential to cause unpredictable behavior. Therefore, implementations MUST NOT
deliver secondary touch events to MIDlets using the MIDP APIs; only the primary touch event and its corresponding drag and release events are to be delivered using the MIDP APIs.

12.3.6 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.

12.4 Graphics and Text in Low-Level API

12.4.1 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.

### 12.4.2 Drawing Model

All implementations MUST support double-buffered graphics. Graphics may be rendered either to the display's offscreen buffer 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` or `CustomItem` 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 rendering operations may be performed with them at any time.

A 32-bit color model is provided with 8 bits each for the red, green, blue, and alpha components of a color. Not all devices support 32-bit resolution, so they will map colors and alpha values requested by the application into values 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 colors or gray levels are available. This enables applications to adapt their behavior to a device without compromising device independence.

The `Graphics` class has a current color and alpha level. These two values can be set with the following methods:

```java
Graphics.setAlpha(int alpha)
Graphics.setAlphaColor(int ARGB)
Graphics.setAlphaColor(int alpha, int red, int green, int blue)
Graphics.setColor(int RGB)
Graphics.setColor(int red, int green, int blue)
Graphics.setGrayscale(int graylevel)
```

All geometric rendering, including lines, rectangles, text, and arcs, uses the current color and alpha. There is no background color; painting of any background must be performed explicitly by the application.

Two Porter-Duff blending modes are supported by the `Graphics` class. `SRC_OVER` is the default blending mode and blends the source pixel's color value on top of the destination pixel. If the source pixel is fully opaque, the destination pixel is effectively replaced with the source pixel. If the source pixel is fully transparent, the destination pixel is unchanged. If the source pixel is partially transparent, its color is blended with the color of the destination pixel. The opacity of the destination pixel cannot be reduced using this blending mode, and thus it is available for `Graphics` objects.

The `SRC` blending mode replaces the destination pixel with the source pixel's value, regardless of the source pixel's opacity. Both the color and the alpha value of the destination pixel are replaced with those of the source pixel, thus allowing the opacity of the destination pixel to be decreased as well.
as increased. For this reason, the SRC blending mode can only be used for Graphics objects that render to an Image with an alpha channel.

12.4.3 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
class Canvas {
    public int getWidth();
    public int getHeight();
}
```

12.4.4 Font Support

Each implementation MAY support a different set of system installed fonts. When an application requests a Font using a specific name, style and pixel size, the implementation will return a Font that most closely matches the request. An application may also use the Font class to query the list of available fonts.

To improve portability across devices, applications may use the following abstract attributes to request an appropriate Font without knowledge of the specific names or pixel sizes that are available on the device:

- **Size:** SMALL, MEDIUM, LARGE.
- **Face:** PROPORTIONAL, MONOSPACE, SYSTEM.
- **Style:** PLAIN, BOLD, ITALIC, UNDERLINED.

However, if an application needs to have complete control over text layout and appearance, it may use custom fonts that are loaded via an InputStream. Custom fonts may be packaged in the application's JAR and accessed as a named resource for this purpose. The application may also download a custom font, but it is responsible for persistently storing the font data on the device if required. Implementations MUST NOT automatically store downloaded font data between MIDlet invocations, and making downloaded fonts persistent (if needed) is solely an application's responsibility.

All implementations MUST support OpenType fonts with TrueType outlines. Implementations SHOULD support TrueType hinting and MAY support advanced typographic functions. Support for other font formats is optional.

12.4.4.1 Downloadable Fonts

MIDlets can use also custom fonts for the rendering of text content. Fonts may be packaged in a MIDlet suite's JAR or in the JARs of the LIBlets a MIDlet suite depends on. Fonts can also be downloaded at runtime and stored on the device in persistent storage for subsequent use (if required by the application). Implementations MUST NOT retain downloaded fonts between
MIDlet invocations, but applications can store downloaded fonts in RMS Record Stores. It is the responsibility of a MIDlet to prepare and instantiate the fonts downloaded at runtime (and/or stored in RMS) by explicitly referencing a font resource using the `Font.createFont` method. Implementations MUST make all individual fonts available (whether downloaded or packaged) to a MIDlet at runtime if the individual font file size does not exceed 200KB. Any individual fonts with a file size that exceeds 200KB MAY be discarded by an implementation.

Fonts that are packaged within a MIDlet Suite JAR or present in dependent LIBlet JARs SHOULD be declared using `MIDlet-Font` or `LIBlet-Font` attribute in their respective JAR manifests. Implementations MUST prepare all declared fonts for later instantiation; any such font can then be instantiated using static method calls (e.g. `Font.getFont` method). Fonts that are packaged but not declared in a JAR manifest will not be prepared by the implementation and can only be instantiated using the `Font.createFont` method. Applications are responsible for the management of all custom fonts not declared with this attribute.

Implementations MUST ensure that the availability and use of fonts packaged with a MIDlet suite in a JAR, packaged with any dependency LIBlets, or downloaded at runtime and created using `createFont` method are limited to the MIDlet's runtime execution environment. If a font packaged with a MIDlet or downloaded at runtime has the same font name as a system font available on a device, the downloaded or packaged font overrides the system font and MUST be used for text rendering purposes whenever a font is selected by name by a MIDlet that created it.

### 12.5 Threading Model

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:
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>;
    }
}

12.6 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.

12.7 Activation of Idle Screen MIDlets

The application context of an idle screen MIDlet is the normal MIDlet. The IdleItem is an additional user interface for the MIDlet. The MIDlet can use the available Displays on the device in addition to the IdleItem on the idle screen of each Display that supports idle.

When an idle screen MIDlet is installed to the device, the platform SHOULD add it to the list of idle screen applications. This makes it possible for the user to select an idle screen MIDlet to be added to the idle screen. The MIDlet name and icon information SHOULD be used to identify the MIDlet in the list of idle screen applications. The system MAY restrict the number of idle screen MIDlets added to the idle screen.

When an idle screen MIDlet is added to the idle screen, the system MUST start the idle screen MIDlet if it is not already running. When the idle screen MIDlet is started it should call Display.setIdleItem to set the IdleItem for one or more Displays and be prepared to render content to it. The system MUST call the addedToDisplay method, announcing that the MIDlet's IdleItem has been added to the idle screen. The following list illustrates the steps that SHOULD be taken by the idle screen MIDlet when it is started.

1. Add IdleItem object to the idle screen with Display.setIdleItem method
2. If the MIDlet will use the normal Display then it should set a DisplayListener to be notified when the Displayable needs to be set
3. Handle the callback method IdleItem.addedToDisplay and render content to the idle screen when its paint method is called
4. When the idle screen MIDlet has been removed from the idle screen handle callback method
   IdleItem.removedFromDisplay

If an idle screen MIDlet has been added to the idle screen and it does not add any content to the idle screen, the system MAY remove the idle screen MIDlet from the idle screen. The MIDlet may be terminated.
If a MIDlet that has not announced itself as an idle screen MIDlet with the JAD or JAR Manifest attribute tries to add content to the idle screen, the system MUST ignore this request.

12.8 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
## Class Summary

### Interfaces

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Choice</strong></td>
<td>Choice defines an API for a user interface components implementing selection from predefined number of choices.</td>
</tr>
<tr>
<td><strong>CommandLayoutPolicy</strong></td>
<td>This interface is used to implement exact placement of commands.</td>
</tr>
<tr>
<td><strong>CommandListener</strong></td>
<td>This interface is used by applications which need to receive high-level events from the implementation.</td>
</tr>
<tr>
<td><strong>DisplayListener</strong></td>
<td>The DisplayListener interface defines a series of methods that are called in response to Display events.</td>
</tr>
<tr>
<td><strong>ItemCommandListener</strong></td>
<td>A listener type for receiving notification of commands that have been invoked on Item objects.</td>
</tr>
<tr>
<td><strong>ItemListLayoutHint</strong></td>
<td>ItemListLayoutHint is an interface to identify classes containing hints that control the layout of Items by subclasses of FormLayoutPolicy.</td>
</tr>
<tr>
<td><strong>ItemStateListener</strong></td>
<td>This interface is used by applications that need to receive events indicating changes in the internal state of the interactive items within a Form screen.</td>
</tr>
<tr>
<td><strong>ItemTraversalListener</strong></td>
<td>This interface is used by applications that need to receive events indicating changes in focus for Items in a Form screen.</td>
</tr>
<tr>
<td><strong>KeyListener</strong></td>
<td>Classes implementing this interface provide methods that are called when user of the device will generate key events, for example, pressing the keys available in a system keypad or keyboard.</td>
</tr>
<tr>
<td><strong>NotificationListener</strong></td>
<td>This interface is used by applications that need to receive events indicating changes in the state of a Notification.</td>
</tr>
<tr>
<td><strong>TabListener</strong></td>
<td>This interface is used to receive events related to changes on a TabbedPane.</td>
</tr>
<tr>
<td><strong>TextEditorChangeListener</strong></td>
<td>A listener for receiving notification of content changes and other editor events that have been invoked on TextEditor objects.</td>
</tr>
</tbody>
</table>

### Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Alert</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>AlertType</strong></td>
<td>The AlertType provides an indication of the nature of alerts.</td>
</tr>
<tr>
<td><strong>AnimatedImage</strong></td>
<td>An AnimatedImage is a special type of Image that encapsulates a series frames and the length of time that each frame should be shown.</td>
</tr>
<tr>
<td><strong>Canvas</strong></td>
<td>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.</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CanvasItem</td>
<td>CanvasItem abstracts the generic features of its subclasses, such as TextEditor component.</td>
</tr>
<tr>
<td>ChoiceGroup</td>
<td>A ChoiceGroup is a group of selectable elements intended to be placed within a Form.</td>
</tr>
<tr>
<td>Command</td>
<td>The Command class is a construct that encapsulates the semantic information of an action.</td>
</tr>
<tr>
<td>CustomItem</td>
<td>A CustomItem is customizable by subclassing to introduce new visual and interactive elements into Forms.</td>
</tr>
<tr>
<td>DateField</td>
<td>A DateField is an editable component for presenting date and time (calendar) information that may be placed into a Form.</td>
</tr>
<tr>
<td>Display</td>
<td>The Display class provides a MIDlet with access to the device's user interface hardware resources.</td>
</tr>
<tr>
<td>Displayable</td>
<td>An object that has the capability of being placed on the display.</td>
</tr>
<tr>
<td>FileSelector</td>
<td>The FileSelector class allows the user to select a file from file system to be loaded, saved, or to select a directory.</td>
</tr>
<tr>
<td>Font</td>
<td>The Font class represents fonts, which are used to render text in a visible way.</td>
</tr>
<tr>
<td>Form</td>
<td>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.</td>
</tr>
<tr>
<td>FormLayoutPolicy</td>
<td>FormLayoutPolicy is subclassed to provide custom layout algorithms.</td>
</tr>
<tr>
<td>Gauge</td>
<td>Implements a graphical display, such as a bar graph, of an integer value.</td>
</tr>
<tr>
<td>Graphics</td>
<td>Provides simple 2D geometric rendering capability.</td>
</tr>
<tr>
<td>IdleItem</td>
<td>This class represents a dedicated UI component that can be used to render content to the idle screen.</td>
</tr>
<tr>
<td>Image</td>
<td>The Image class is used to hold graphical image data.</td>
</tr>
<tr>
<td>ImageItem</td>
<td>An item that can contain an image.</td>
</tr>
<tr>
<td>Item</td>
<td>A superclass for components that can be added to a Form.</td>
</tr>
<tr>
<td>List</td>
<td>A Screen containing list of choices.</td>
</tr>
<tr>
<td>Menu</td>
<td>A visual container for Commands and other Menus.</td>
</tr>
<tr>
<td>Notification</td>
<td>Represents a small unobtrusive informational note to be shown to the user.</td>
</tr>
<tr>
<td>NotificationType</td>
<td>Represents the Notification type (or category) used for grouping, sorting and filtering Notification objects of the same type.</td>
</tr>
<tr>
<td>ScalableImage</td>
<td>A ScalableImage object encapsulates vector graphics content.</td>
</tr>
<tr>
<td>Screen</td>
<td>The common superclass of all high-level user interface classes.</td>
</tr>
<tr>
<td>Spacer</td>
<td>A blank, non-interactive item that has a settable minimum size.</td>
</tr>
<tr>
<td>StringItem</td>
<td>An item that can contain a string.</td>
</tr>
<tr>
<td>TabbedPane</td>
<td>TabbedPane is a Screen subclass that presents a series of Screens to the users and allows them to navigate between screens by selecting the corresponding tab.</td>
</tr>
<tr>
<td>TableLayoutPolicy</td>
<td>TableLayoutPolicy displays the Items in a Form aligned in columns.</td>
</tr>
<tr>
<td>Text</td>
<td>The Text class is used to layout and render text within a specific area.</td>
</tr>
<tr>
<td>TextBox</td>
<td>The TextBox class is a Screen that allows the user to enter and edit text.</td>
</tr>
<tr>
<td>TextEditor</td>
<td>A TextEditor is an editable text component that is drawn on a parent object; in LCDUI Canvas or CustomItem (including IdleItem).</td>
</tr>
<tr>
<td>TextField</td>
<td>A TextField is an editable text component that may be placed into a Form.</td>
</tr>
<tr>
<td>Ticker</td>
<td>Implements a &quot;ticker-tape&quot;, a piece of text that runs continuously across the display.</td>
</tr>
</tbody>
</table>

### Exceptions

| DisplayCapabilityException | Indicates that a Display's capabilities are insufficient for the requested operation. |
| FontFormatException | Indicates that a font format is not supported, or that font data is invalid or is not conformant with the specified font format (OpenType with TrueType outlines). |
| NotificationException | Indicates that an operation on a Notification has failed. |
javax.microedition.lcdui.Alert
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 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 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 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 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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final</td>
<td>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</td>
<td>FOREVER</td>
<td>FOREVER indicates that an Alert is kept visible until the user dismisses it.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value: -2</td>
</tr>
</tbody>
</table>

Constructor Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>Alert(String title)</td>
<td>Constructs a new, empty Alert object with the given title.</td>
</tr>
</tbody>
</table>
### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public Alert(String title, String alertText, Image alertImage, AlertType alertType)</code></td>
<td>Constructs a new <code>Alert</code> object with the given title, content string and image, and alert type.</td>
</tr>
<tr>
<td><code>void addCommand(Command cmd)</code></td>
<td>Similar to <code>Displayable.addCommand(Command)</code>, however when the application first adds a command to an <code>Alert</code>, <code>DISMISS_COMMAND</code> is implicitly removed.</td>
</tr>
<tr>
<td><code>int getDefaultValue()</code></td>
<td>Gets the default time for showing an <code>Alert</code>.</td>
</tr>
<tr>
<td><code>int getHeight()</code></td>
<td>Gets the height in pixels of the displayable area available for placing Image, String, and Gauge items in the <code>Alert</code>.</td>
</tr>
<tr>
<td><code>javax.microedition.lcdui.Image getImage()</code></td>
<td>Gets the <code>Image</code> used in the <code>Alert</code>.</td>
</tr>
<tr>
<td><code>javax.microedition.lcdui.Gauge getIndicator()</code></td>
<td>Gets the activity indicator for this <code>Alert</code>.</td>
</tr>
<tr>
<td><code>java.lang.String getString()</code></td>
<td>Gets the text string used in the <code>Alert</code>.</td>
</tr>
<tr>
<td><code>int getTimeout()</code></td>
<td>Gets the time this <code>Alert</code> will be shown.</td>
</tr>
<tr>
<td><code>javax.microedition.lcdui.AlertType getType()</code></td>
<td>Gets the type of the <code>Alert</code>.</td>
</tr>
<tr>
<td><code>int getWidth()</code></td>
<td>Gets the width in pixels of the displayable area available for placing Image, String, and Gauge items in the <code>Alert</code>.</td>
</tr>
<tr>
<td><code>void removeCommand(Command cmd)</code></td>
<td>Similar to <code>Displayable.removeCommand(Command)</code>, however when the application removes the last command from an <code>Alert</code>, <code>DISMISS_COMMAND</code> is implicitly added.</td>
</tr>
<tr>
<td><code>void setCommandListener(CommandListener l)</code></td>
<td>The same as <code>Displayable.setCommandListener(CommandListener)</code> but with the following additional semantics.</td>
</tr>
<tr>
<td><code>void setImage(Image img)</code></td>
<td>Sets the <code>Image</code> used in the <code>Alert</code>.</td>
</tr>
<tr>
<td><code>void setIndicator(Gauge indicator)</code></td>
<td>Sets an activity indicator on this <code>Alert</code>.</td>
</tr>
<tr>
<td><code>void setString(String str)</code></td>
<td>Sets the text string used in the <code>Alert</code>.</td>
</tr>
<tr>
<td><code>void setTimeout(int time)</code></td>
<td>Set the time for which the <code>Alert</code> is to be shown.</td>
</tr>
<tr>
<td><code>void setType(AlertType type)</code></td>
<td>Sets the type of the <code>Alert</code>.</td>
</tr>
</tbody>
</table>
Since: MIDP 2.0

**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 on an 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.

**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:** `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`, `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
- `DisplayCapabilityException` - if the `Alert` is for use with a `Display` that does not support `Commands`

### 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`
**getHeith**

```java
public int getHeight()
```

Gets the height in pixels of the displayable area available for placing Image, String, and Gauge items in the Alert.

**Returns:**
height of the area available for Image, String, and Gauge items

**Since:** MIDP 3.0

**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:** `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:** `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:** `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 `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:** `setTimeout(int)`

**getType**

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

**javamicroedition.lcdui.Alert**
getWidth

public int getWidth()

Gets the width in pixels of the displayable area available for placing Image, String, and Gauge items in the Alert.

Returns:
width of the area available for Image, String, and Gauge items

Since: MIDP 3.0

removeCommand

public void removeCommand(Command cmd)

Similar to Displayable.removeCommand(Command), however when the application removes the last command from an 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

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

public void setImage(Image img)

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.
**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:** `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:** `getString()`

---

**setTimeout**

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

Set the time for which the `Alert` is to be shown. This must either be a positive time value in milliseconds, or the special value `FOREVER`.

**Parameters:**
- `time` - timeout in milliseconds, or `FOREVER`

**Throws:**
- `IllegalArgumentException` - if `time` is not positive and is not `FOREVER`

**See Also:** `getTimeout()`

---

**setType**

```java
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:** `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 MIDlet using a Form with a progress bar can use AlertType.CONFIRMATION.playSound(display) to indicate that an operation has completed. The predefined types are INFO, WARNING, ERROR, ALARM, and CONFIRMATION.

Since: MIDP 1.0

See Also: Alert

Field Summary

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

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/information 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 to 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 **AlertType**s or use any other means suitable to alert the user. The device indication to the user may not necessarily be sound, but may instead be visual or tactile feedback based on device settings or capabilities. For example, a user may have configured a device to be silent, so the implementation may instead flash the backlight or vibrate.
javax.microedition.lcdui.AlertType

Parameters:
   display - to which the AlertType's sound should be played.

Returns:
   true if the user was alerted, false otherwise.

Throws:
   NullPointerException - if display is null
javax.microedition.lcdui

AnimatedImage

Declaration

```java
public class AnimatedImage extends Image
```

Object

```java
javax.microedition.lcdui.AnimatedImage
```

Description

An AnimatedImage is a special type of Image that encapsulates a series frames and the length of time that each frame should be shown.

An AnimatedImage is obtained by calling Image.createImage with animated image data from a named resource, byte array, or input stream. GIF89a image data format MUST be supported. The returned object will be an instance of an AnimatedImage and can be cast accordingly. The `Image.isAnimated()` method can be called on any Image instance to determine whether or not it is animated and can be cast to an AnimatedImage.

An immutable Image representing an individual frame may be retrieved by calling `getFrame` with the desired frame index. The index of the first frame is zero. To reduce file size, some formats may store partial frames that need to be overlaid on top of prior frames in order to create a complete rendered frame. For these formats, the implementation is responsible for combining partial frames with prior frames as needed, and each Image returned by `getFrame` must contain a complete frame that can be rendered as-is.

As a subclass of Image, an AnimatedImage may itself be used as an argument for various methods that render, copy, or retrieve pixel data from an Image. When used in this manner, the first frame of the AnimatedImage is used. For example, the operation

```java
g.drawImage(myAnimatedImage, 0, 0, Graphics.LEFT + Graphics.TOP);
```

is equivalent to

```java
g.drawImage(myAnimatedImage.getFrame(0), 0, 0, Graphics.LEFT + Graphics.TOP);
```

When used with an Imageltem, List, or other high-level UI element, the implementation may automatically animate an AnimatedImage when the UI element is shown. The implementation may stop animations after a period of time to reduce power consumption. If animations are not supported by the device’s UI, the first frame of the animation is shown instead.

Applications using Canvas or CustomItem are responsible for triggering frame changes and the corresponding repaint requests. For example, a TimerTask or dedicated background thread may be used for this purpose. However, to avoid excessive power consumption, care should be taken to ensure that animations are not run indefinitely or while the Canvas or CustomItem is not visible.

The following example illustrates how an AnimatedImage may be shown on a Canvas. This example restarts the animation each time the Canvas is shown, but it is also possible to pause and resume the
javax.microedition.lcdui.AnimatedImage

animation instead.
public class MyCanvas extends Canvas implements Runnable {

    AnimatedImage img = (AnimatedImage) Image.createImage("MyAnimatedLogo.gif");

    int currentFrame = 0;
    int updateCount = 0;
    boolean loopForever = false;
    boolean canvasIsShown = false;

    public void paint(Graphics g) {
        // Render the current frame
        g.drawImage(img.getFrame(currentFrame), 0, 0, Graphics.TOP + Graphics.LEFT);
    }

    public void showNotify() {
        // Start the animation at the first frame
        canvasIsShown = true;
        currentFrame = 0;
        new Thread(this).start();
    }

    public synchronized void hideNotify() {
        // End the animation
        canvasIsShown = false;
        notify();
    }

    public void run() {

        // Determine if the animation should run for a finite number of frames
        int loopCount = img.getLoopCount();
        if (loopCount == -1)
            loopForever = true;
        else
            updateCount = (loopCount + 1) * img.getFrameCount();

        synchronized (this) {
            // Run the animation while the Canvas is shown
            while (canvasIsShown) {

                // Paint the current frame
                repaint();

                // Wait until the next frame should be painted
                try {
                    wait(img.getFrameDelay(currentFrame));
                } catch (Exception e) { }

                // Move to the next frame
                currentFrame = (currentFrame + 1) % img.getFrameCount();
            }
        }
    }
}
javax.microedition.lcdui.AnimatedImage

// Check if the animation should end
if (!loopForever && (--updateCount == 0)) return;

Since: MIDP 3.0

Method Summary

javax.microedition.lcdui.Image getFrame(int index)
Gets the Image for the specified frame.

int getFrameCount()
Gets the number of frames in this AnimatedImage.

int getFrameDelay(int index)
Gets the frame delay for the specified frame.

int getLoopCount()
Gets the number of times that the animation loop should be repeated.

Methods inherited from class javax.microedition.lcdui.Image
createImage, createImage, createImage, createImage, createImage, createImage, createImage, createImage, createRGBImage, getARGB16, getGraphics, getHeight, getRGB, getRGB16, getWidth, hasAlpha, isAnimated, isMutable, isScalable

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

Methods

getFrame

public javax.microedition.lcdui.Image getFrame(int index)

Gets the Image for the specified frame. To avoid the creation of garbage objects, the same Image object is returned when this method is called with the same index value. The index of the first frame is 0.

Parameters:
index - the index of the frame

Returns:
An Image containing the pixel data for the frame

Throws:
IndexOutOfBoundsException - if the index is invalid
**getFrameCount**

```java
public int getFrameCount()
```

Gets the number of frames in this AnimatedImage.

**Returns:**
the number of frames

---

**getFrameDelay**

```java
public int getFrameDelay(int index)
```

Gets the frame delay for the specified frame. The delay indicates the amount of time that the specified frame should be shown before being replaced with the next frame.

**Parameters:**
- `index` - the index of the frame

**Returns:**
The frame delay, in milliseconds

**Throws:**
- `IndexOutOfBoundsException` - if the index is invalid

---

**getLoopCount**

```java
public int getLoopCount()
```

Gets the number of times that the animation loop should be repeated. A value of 0 indicates that the sequence of frames should be animated only once and then stop. A value of 1 indicates that the sequence of frames should be repeated once, resulting in 2 complete frame sequences. A value of -1 indicates that the animation should be repeated indefinitely.

**Returns:**
the loop count for this animation
javax.microedition.lcdui.Canvas

Declaration

public abstract class Canvas extends Displayable

Object

\|--javax.microedition.lcdui.Displayable
\|--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 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 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 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 fewer listener interfaces, but this would require listeners to filter out events in which they had no interest.

CanvasItem subclasses, such as a TextEditor, may be drawn on Canvas or CustomItem. See CanvasItem and TextEditor for more details.

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 a ITU-T standard telephone keypad.) Other keys may be present on the keyboard, and they will generally have key codes distinct from those list above. In order to guarantee portability, applications should use only the standard key codes.
The standard key codes' values are equal to the Unicode encoding for the character that represents the key. If the device includes any other keys that have an obvious correspondence to a Unicode character, their key code values should equal the Unicode encoding for that character. For keys that have no corresponding Unicode character, the implementation must use negative values. Zero is defined to be an invalid key code. It is thus possible for an application to convert a keyCode into a Unicode character using the following code:

```java
if (keyCode > 0) {
    char ch = (char) keyCode;
    // ...
}
```

This technique is useful only in certain limited cases. In particular, it is not sufficient for full textual input, because it does not handle upper and lower case, keyboard shift states, and characters that require more than one keystroke to enter. For textual input, applications should always use TextBox 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 getKeyName() method to find a key's name.

### Actions (aka Game Actions)

Portable applications that need soft key and arrow events and action-related events should use actions in preference to key codes and key names. Soft key codes are mapped to actions defined by exact placement values as specified for the [Exact placement of Commands](#). Navigation and game keys are mapped to the 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 action. However, an action may be associated with more than one key code. The application can translate a key code into an action using the `getGameAction(int keyCode)` method, and it can translate an action into a key code using the `getKeyCode(int gameAction)` method. There may be multiple keycodes associated with a particular action, but `getKeyCode` returns only one of them. Supposing that `g` is a valid action and `k` is a valid key code for a key associated with a action, consider the following expressions:

```java
g == getGameAction(getKeyCode(g))  // (1)
k == getKeyCode(getGameAction(k))   // (2)
```

Expression (1) is *always* true. However, expression (2) might be true but is *not necessarily* true.

Portable applications that are interested in using actions should translate every key event into an action by calling the `getGameAction()` method and then testing the result. For example, on some devices the actions **UP**, **DOWN**, **LEFT** and **RIGHT** may be mapped to 4-way navigation arrow keys. In this case, `getGameAction(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, `getGameAction(UP)` would return KEY_NUM2. In both cases, the `getGameAction()` method would return the `LEFT` action when the user presses the key that is a "natural left" on her device.

The implementation is not allowed to change the mapping of actions and key codes during execution of the application. However, 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 action does not change during the execution of the application. Consequently, applications should use the `getGameAction()` method to determine which action is associated with a given key code. It is recommended that applications avoid using the `getKeyCode()` method to determine which key is assigned to a specific action.
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 MUST 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 setFullScreenMode(boolean) method.

Calling 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

<table>
<thead>
<tr>
<th>Field Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td>Constant for requesting the full set of actions comprised of UP, DOWN, LEFT, RIGHT, FIRE, GAME_A, GAME_B, GAME_C, and GAME_D.</td>
</tr>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td>Constant for requesting the basic set of actions comprised of UP, DOWN, LEFT, RIGHT, and FIRE ACTIONS_NAVIGATION has a value of -1.</td>
</tr>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td>Constant for requesting the empty set of actions ACTIONS_NONE has a value of 0.</td>
</tr>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td>Constant for the DOWN action.</td>
</tr>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td>Constant for the FIRE action.</td>
</tr>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td>Constant for the general purpose &quot;A&quot; action.</td>
</tr>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td>Constant for the general purpose &quot;B&quot; action.</td>
</tr>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td>Constant for the general purpose &quot;C&quot; action.</td>
</tr>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td>Constant for the general purpose &quot;D&quot; action.</td>
</tr>
<tr>
<td>Method Signature</td>
</tr>
<tr>
<td>----------------------</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>public static final</td>
</tr>
</tbody>
</table>

**Constructor Summary**

| protected Canvas() | Constructs a new Canvas object |

**Method Summary**
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int getGameAction(int keyCode)</code></td>
<td>Gets the action associated with the given key code of the device.</td>
</tr>
<tr>
<td><code>int getHeight()</code></td>
<td>Gets the height in pixels of the displayable area of the Canvas.</td>
</tr>
<tr>
<td><code>int getKeyCode(int action)</code></td>
<td>Gets a key code that corresponds to the specified action on the device.</td>
</tr>
<tr>
<td><code>java.lang.String getKeyName(int keyCode)</code></td>
<td>Gets an informative key string for a key.</td>
</tr>
<tr>
<td><code>int[] getSoftkeyLabelCoordinates(int placement)</code></td>
<td>Gets the upper-left coordinates on the Canvas coordinate system, width, and height of the given placement.</td>
</tr>
<tr>
<td><code>int getWidth()</code></td>
<td>Gets the width in pixels of the displayable area of the Canvas.</td>
</tr>
<tr>
<td><code>boolean hasPointerEvents()</code></td>
<td><strong>Deprecated.</strong> Support for pointer press and release events are determined by the Display. Use Display.hasPointerEvents()</td>
</tr>
<tr>
<td><code>boolean hasPointerMotionEvents()</code></td>
<td><strong>Deprecated.</strong> Support for pointer motion events are determined by the Display. Use Display.hasPointerMotionevents()</td>
</tr>
<tr>
<td><code>boolean hasRepeatEvents()</code></td>
<td>Checks if the platform can generate repeat events when key is kept down.</td>
</tr>
<tr>
<td><code>void hideNotify()</code></td>
<td>The implementation calls hideNotify() shortly after the Canvas has been removed from the display.</td>
</tr>
<tr>
<td><code>boolean isDoubleBuffered()</code></td>
<td>Checks if the Canvas is double buffered by the implementation.</td>
</tr>
<tr>
<td><code>void keyPressed(int keyCode)</code></td>
<td>Called when a key is pressed.</td>
</tr>
<tr>
<td><code>void keyReleased(int keyCode)</code></td>
<td>Called when a key is released.</td>
</tr>
<tr>
<td><code>void keyRepeated(int keyCode)</code></td>
<td>Called when a key is repeated (held down).</td>
</tr>
<tr>
<td><code>abstract void paint(Graphics g)</code></td>
<td>Renders the Canvas.</td>
</tr>
<tr>
<td><code>void pointerDragged(int x, int y)</code></td>
<td>Called when the pointer is dragged.</td>
</tr>
<tr>
<td><code>void pointerPressed(int x, int y)</code></td>
<td>Called when the pointer is pressed.</td>
</tr>
<tr>
<td><code>void pointerReleased(int x, int y)</code></td>
<td>Called when the pointer is released.</td>
</tr>
<tr>
<td><code>void repaint()</code></td>
<td>Requests a repaint for the entire Canvas.</td>
</tr>
</tbody>
</table>
```java
void repaint(int x, int y, int width, int height)
Requests a repaint for the specified region of the Canvas.

void serviceRepaints()
Forces any pending repaint requests to be serviced immediately.

void setFullScreenMode(boolean mode)
Controls whether the Canvas is in full-screen mode or in normal mode.

void setKeyListener(KeyListener listener)
Sets a key listener for key events to this Canvas, replacing any previous KeyListener.

void setPaintMode(boolean opaque)
Sets the paint mode for this Canvas.

void setRequiredActions(int actionSet)
Specifies the set of actions that must be available on touch screen devices while this Canvas is current on the foreground Display.

void showNotify()
The implementation calls showNotify() immediately prior to this Canvas being made visible on the display.

void sizeChanged(int w, int h)
Called when the drawable area of the Canvas has been changed.
```

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

- addCommand, getCommand, getCommandLayoutPolicy, getCommands, getCurrentDisplay, getMenu, getTicker, getTitle, invalidateCommandLayout, isShown, removeCommand, removeCommandOrMenu, setCommand, setCommandLayoutPolicy, setCommandListener, setMenu, setTicker, setTitle,

**Methods inherited from class** `Object`

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

---

**Fields**

**ACTIONS_ALL**

```java
public static final int ACTIONS_ALL
```

Constant for requesting the full set of actions comprised of UP, DOWN, LEFT, RIGHT, FIRE, GAME_A, GAME_B, GAME_C, and GAME_D. ACTIONS_ALL has a value of -2.

Constant value: -2

**Since:** MIDP 3.0

**ACTIONS_NAVIGATION**

```java
public static final int ACTIONS_NAVIGATION
```

Constant for requesting the basic set of actions comprised of UP, DOWN, LEFT, RIGHT, and FIRE. ACTIONS_NAVIGATION has a value of -1.

Constant value: -1

**Since:** MIDP 3.0
**ACTIONS_NONE**

```java
public static final int ACTIONS_NONE
```

Constant for requesting the empty set of actions ACTIONS_NONE has a value of 0.

**Since:** MIDP 3.0

---

**DOWN**

```java
public static final int DOWN
```

Constant for the DOWN action.

**Constant value:** 6 is set to DOWN.

**Constant value:** 6

---

**FIRE**

```java
public static final int FIRE
```

Constant for the FIRE 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" 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" 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" 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" action.

**Constant value:** 12 is set to GAME_D.

**Constant value:** 12
javax.microedition.lcdui.Canvas

javax.microedition.lcdui.Canvas

KEY_BACKSPACE
public static final int KEY_BACKSPACE

keyCode for the backspace key, U+0008 BACKSPACE.

Constant value 8 is set to KEY_BACKSPACE.
Constant value: 8

KEY_DELETE
public static final int KEY_DELETE

keyCode for the delete key, U+007F DELETE.

Constant value 127 is set to KEY_DELETE.
Constant value: 127

KEY_DOWN
public static final int KEY_DOWN

keyCode for the down key.

Constant value -2 is set to KEY_DOWN.
Constant value: -2

KEY_ENTER
public static final int KEY_ENTER

keyCode for the enter key, U+000A LINE FEED.

Constant value 10 is set to KEY_ENTER.
Constant value: 10

KEY_ESCAPE
public static final int KEY_ESCAPE

keyCode for the escape key, U+001B ESCAPE.

Constant value 27 is set to KEY_ESCAPE.
Constant value: 27

KEY_LEFT
public static final int KEY_LEFT

keyCode for the left key.

Constant value -3 is set to KEY_LEFT.
Constant value: -3

KEY_NUM0
public static final int KEY_NUM0

keyCode for ITU-T key 0.

Constant value 48 is set to KEY_NUM0.
javax.microedition.lcdui.Canvas

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_RIGHT

public static final int KEY_RIGHT

keyCode for the right key.

Constant value -4 is set to KEY_RIGHT.
Constant value: -4

KEY_SELECT

public static final int KEY_SELECT

keyCode for the select key.

Constant value -5 is set to KEY_SELECT.
Constant value: -5

KEY_SPACE

public static final int KEY_SPACE

keyCode for the space key, U+0020 SPACE.

Constant value 32 is set to KEY_SPACE.
Constant value: 32

KEY_STAR
keyCode for ITU-T key "star" (\*).
Constant value \texttt{42} is set to \texttt{KEY\_STAR}.
Constant value: \texttt{42}

**KEY\_TAB**

\begin{verbatim}
public static final int KEY_TAB

keyCode for the tabulation key, U+0009 HORIZONTAL TABULATION.
Constant value \texttt{9} is set to \texttt{KEY\_TAB}.
Constant value: \texttt{9}
\end{verbatim}

**KEY\_UP**

\begin{verbatim}
public static final int KEY_UP

keyCode for the up key.
Constant value \texttt{-1} is set to \texttt{KEY\_UP}.
Constant value: \texttt{-1}
\end{verbatim}

**LEFT**

\begin{verbatim}
public static final int LEFT

Constant for the \texttt{LEFT} action.
Constant value \texttt{2} is set to \texttt{LEFT}.
Constant value: \texttt{2}
\end{verbatim}

**RIGHT**

\begin{verbatim}
public static final int RIGHT

Constant for the \texttt{RIGHT} action.
Constant value \texttt{5} is set to \texttt{RIGHT}.
Constant value: \texttt{5}
\end{verbatim}

**UP**

\begin{verbatim}
public static final int UP

Constant for the \texttt{UP} action.
Constant value \texttt{1} is set to \texttt{UP}.
Constant value: \texttt{1}
\end{verbatim}

**Constructors**

**Canvas**

\begin{verbatim}
protected Canvas()

Constructs a new Canvas object
\end{verbatim}

**Methods**
**getGameAction**

public int getGameAction(int keyCode)

Gets the action associated with the given key code of the device. Returns zero if no action is associated with this key code. For soft key keycodes, the action returned is the placement as defined in Display for Exact Placement of Commands. For navigation and game keys the action values are listed in the Actions description.

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

**Parameters:**
- keyCode - the key code

**Returns:**
- the Action corresponding to this key, or the soft key placement for soft keys, or 0 if none

**Throws:**
- IllegalArgumentException - if keyCode is not a valid key code

**getHeigh**

public int getHeight()

Gets the height in pixels of the displayable area of the Canvas. The displayable area is the area that can be directly drawn on by the Canvas Graphics object. The value returned may change during execution. If it does, the application will be notified through a call to the sizeChanged(int, int) method. If setCurrent() has not yet been called to associate this Canvas object with a particular display, this call returns the height of the primary display. In fullScreenMode, the height is the same as the value returned by Display getHeight().

**Returns:**
- height of the area available for the application to draw into

**get.KeyCode**

public int getKeyCode(int action)

Gets a key code that corresponds to the specified action on the device. The implementation is required to provide a mapping for every action, so this method will always return a valid key code for every action. There may be multiple keys associated with the same action; however, this method will return only one of them. Applications should translate the key code of every key event into an action using getGameAction(int) and then interpret the resulting action, instead of generating a table of key codes at using this method during initialization.

Valid soft key actions are those placements returned from Display.getExactPlacementPositions for any of the borders as defined in Exact placement of Commands.

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

**Parameters:**
- action - the action

**Returns:**
- a key code corresponding to this action

**Throws:**
- IllegalArgumentException - if action is not a valid 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 actions to key names. To get the string name for an 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

getSoftkeyLabelCoordinates

public int[] getSoftkeyLabelCoordinates(int placement)

Gets the upper-left coordinates on the Canvas coordinate system, width, and height of the given placement. The returned values represent a rectangular area which the implementation uses for drawing a softkey at the given placement. These dimensions do not take into account any graphical effects (for example, rounded corners, key size that change due to focus change etc). If setCurrent has not yet been called to associate this Canvas object with a particular Display, this call returns the upper-left coordinates on the primary Display coordinate system, width and height for the given placement. In normal mode, this method may return values beyond the Canvas width and height or negative values. The method only returns coordinates for placements on Display.SOFTKEY_TOP, Display.SOFTKEY_BOTTOM, Display.SOFTKEY_LEFT or Display.SOFTKEY_RIGHT borders. The method throws IllegalArgumentException for placements on the Display.SOFTKEY_OFFSCREEN border.

Returns:
    Gets the upper-left coordinates, width and height (x,y,width,height) of the particular placement.

Throws:
    IllegalArgumentException - if placement is not valid or if valid placement on
    SOFTKEY_OFFSCREEN border.

Since: MIDP 3.0

getWidth

public int getWidth()

Gets the width in pixels of the displayable area of the Canvas. The displayable area is the area that can be directly drawn on by the Canvas Graphics object. The value returned may change during execution. If it does, the application will be notified through a call to the sizeChanged(int, int) method. If setCurrent() has not yet been called to associate this Canvas object with a particular display, this call returns the width of the primary display. In fullScreenMode, the width is the same as the value returned by Display.getWidth().
hasPointerEvents

public boolean hasPointerEvents()

*Deprecated. Support for pointer press and release events are determined by the Display. Use Display.hasPointerEvents()*

Checks if the Display to which the Canvas is set to be visible on, using Display.setCurrent(), supports pointer press and release events. Pointer press and release events MUST be supported if the underlying hardware supports this feature. If setCurrent() has not yet been called to associate this Canvas object with a particular display, this call returns information based on the primary display.

*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()

*Deprecated. Support for pointer motion events are determined by the Display. Use Display.hasPointerMotionEvents()*

Checks if the Display to which the Canvas is set to be visible on, using Display.setCurrent(), supports pointer motion events (pointer dragged). Applications may use this method to determine if the associated Display is capable of supporting motion events. Pointer motion events MUST be supported if the underlying hardware supports this feature. If setCurrent() has not yet been called to associate this Canvas object with a particular display, this call returns information based on the primary display.

*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 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 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 action, if any, is mapped to the key. Class `Canvas` has an empty implementation of this method, and the subclass has to redefine it if it wants to listen this method.

Parameters:
keyCode - the key code of the key that was repeated

See Also: `hasRepeatEvents()`

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, unless the paint mode has been set to transparent (see Canvas.setPaintMode). 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 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

```java
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)

```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());
```

---

```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), 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

setKeyListener

public void setKeyListener(KeyListener listener)

Sets a key listener for key events to this Canvas, replacing any previous KeyListener. A null reference is allowed and has the effect of removing any existing listener. The set of key delivery methods of Canvas (keyPressed(), keyRepeated() and keyReleased()) MUST work even if a KeyListener has been set, however the application should use only one of the ways to listen to key events. The method call order of Canvas methods and KeyListener methods is undefined.

Parameters:
listener - the new listener, or null to remove a listener.

Since: MIDP 3.0

setPaintMode

public void setPaintMode(boolean opaque)
Sets the paint mode for this Canvas.

If the paint mode is *opaque*, the implementation may assume that the every pixel in the clip region will be rendered by the `paint` method. Thus, the implementation does not need to clear or reset the pixels to a suitable state prior to calling the `paint` method since they will all be rendered by the Canvas.

In this example, the opaque paint mode is used and the paint method renders a white background and black text, thus fully obscuring the device’s background wallpaper:

```java
paint(Graphics g) {
    g.setColor(0xFFFFFF);
    g.fillRect(0, 0, getWidth(), getHeight());
    g.setColor(0x000000);
    g.drawString("Some Text", 10, 10, Graphics.TOP + Graphics.LEFT);
}
```

If the paint mode is *transparent*, the implementation is responsible for appropriately filling the entire clip region prior to calling the `paint` method. Some devices may fill the pixels to a suitable background color, while others may fill them with a suitable background image. Hence, any pixels untouched by the `paint` method will be colored appropriately. In this mode, the `paint` method does not need to render every pixel within the clip region; it should render only those pixels within the clip region that it wishes to control the contents of. By rendering a subset of the pixels in this manner, the contents of the Canvas will appear rendered on top of the background content provided by the implementation.

In this example, the transparent paint mode is used and the paint method renders only the black text, thus making it appear on top of the device’s background wallpaper:

```java
paint(Graphics g) {
    g.setColor(0x000000);
    g.drawString("Some Text", 10, 10, Graphics.TOP + Graphics.LEFT);
}
```

The paint mode is opaque by default.

**Parameters:**

- `opaque` - true to set the paint mode to opaque, false to set it to transparent

**See Also:** `paint(Graphics)`

**Since:** MIDP 3.0

---

**setRequiredActions**

```java
public void setRequiredActions(int actionSet)
```
javax.microedition.lcdui.Canvas

Specifies the set of actions that must be available on touch screen devices while this Canvas is current on the foreground Display. All MIDP devices MUST support basic actions (UP, DOWN, LEFT, RIGHT, and FIRE) and MAY provide support for additional actions (GAME_A, GAME_B, GAME_C, and GAME_D). On touch screen devices, some or all of these actions may be provided using an on-screen keypad, and this method may be used to specify the specific set of actions required, thereby permitting the implementation to optimize the size of the on-screen keypad. Calls to this method should be ignored by keypad devices that do not include a touch screen. By default, the required set of actions is set to ACTIONS_NAVIGATION, and the implementation MUST ensure that at least the basic actions (UP, DOWN, LEFT, RIGHT, and FIRE) are supported. If the required set of actions is set to ACTIONS_ALL, the implementation MUST ensure that all actions (UP, DOWN, LEFT, RIGHT, and FIRE GAME_A, GAME_B, GAME_C, and GAME_D) are supported. If the required set of actions is set to ACTIONS_NONE, the implementation is not required to support any game actions and may hide the on-screen keypad if desired. The application may still receive key events for an action even if it is not included in the required set.

Parameters:
actionSet - The set of actions required by this Canvas (ACTIONS_NAVIGATION, ACTIONS_ALL, or ACTIONS_NONE)

Throws:
IllegalArgumentException - if actionSet is not one of ACTIONS_NAVIGATION, ACTIONS_ALL, or ACTIONS_NONE

Since: MIDP 3.0

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.
CanvasItem

Declaration

public abstract class CanvasItem

<table>
<thead>
<tr>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.microedition.lcdui.CanvasItem</td>
</tr>
</tbody>
</table>

Direct Known Subclasses:

javax.microedition.lcdui.TextEditor

Description

CanvasItem abstracts the generic features of its subclasses, such as TextEditor component. Such features include setting the size and position component. Also associating the item, such as TextEditor, with its parent, such as Canvas or CustomItem, may be done through this API with setParent() method.

Since: 3.0

See Also: TextEditor, Canvas, CustomItem

Constructor Summary

public CanvasItem()

Constructor for CanvasItem.

Method Summary

<table>
<thead>
<tr>
<th>int getHeight()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gets the height of this CanvasItem in pixels.</td>
</tr>
</tbody>
</table>

java.lang.Object getParent()

Gets the parent object of this CanvasItem.

int getPositionX()

Gets the x-coordinate of the rendering position of this CanvasItem.

int getPositionY()

Gets the y-coordinate of the rendering position of this CanvasItem.

int getWidth()

Gets the width of this CanvasItem in pixels.

int getZPosition()

Gets the Z-position, or the elevation, of the item.

void setParent(Object parent)

Sets the parent object of this CanvasItem.

void setPositionX(int x)

Sets the x-coordinate of the rendering position of this CanvasItem.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setPositionY(int y)</td>
<td>Sets the y-coordinate of the rendering position of this CanvasItem.</td>
</tr>
<tr>
<td>setSize(int width, int height)</td>
<td>Sets the size of this CanvasItem in pixels.</td>
</tr>
<tr>
<td>setVisible(boolean visible)</td>
<td>Sets the visibility value of the CanvasItem.</td>
</tr>
<tr>
<td>setZPosition(int z)</td>
<td>Sets the Z-position, or the elevation, of the item.</td>
</tr>
</tbody>
</table>

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

Constructors

**CanvasItem**

public CanvasItem()

Constructor for CanvasItem.

Methods

**getHeight**

public int getHeight()

Gets the height of this CanvasItem in pixels.

Returns:
height in pixels

**getParent**

public java.lang.Object getParent()

Gets the parent object of this CanvasItem. Typically the parent object would be Canvas or CustomItem.

Returns:
the parent object, null if no parent has been set

**getPositionX**

public int getPositionX()

Gets the x-coordinate of the rendering position of this CanvasItem. The origin is relative to the coordinate system of the parent.

Returns:
the x-coordinate position of the CanvasItem
getPositionY

```java
public int getPositionY()
```

Gets the y-coordinate of the rendering position of this CanvasItem. The origin is relative to the coordinate system of the parent.

**Returns:**
the y-coordinate position of the CanvasItem

getWidth

```java
public int getWidth()
```

Gets the width of this CanvasItem in pixels.

**Returns:**
width in pixels

getZPosition

```java
public int getZPosition()
```

Gets the Z-position, or the elevation, of the item. The Z-position decides the stacking order of neighboring items.

**Returns:**
the Z-position of the item

setParent

```java
public void setParent(Object parent)
```

Sets the parent object of this CanvasItem. Typically the parent object would be Canvas or CustomItem. Setting the parameter to null removes the association to the parent. If setParent(null) is called for a CanvasItem yet not having any parent the call is silently ignored.

**Parameters:**
parent - the parent object

**Throws:**
IllegalArgumentException - if parent is not a valid object with which a CanvasItem can be associated, or if CanvasItem is already set to another parent

setPositionX

```java
public void setPositionX(int x)
```

Sets the x-coordinate of the rendering position of this CanvasItem. The origin is relative to the coordinate system of the parent.

The CanvasItem may be placed partially or fully off of the visible area of the parent by the setPosition method; in this case the CanvasItem is just not fully visible.

**Parameters:**
x - the x coordinate of the anchor point, in pixels

setPositionY

```java
public void setPositionY(int y)
```
Sets the y-coordinate of the rendering position of this `CanvasItem`. The origin is relative to the coordinate system of the parent.

The `CanvasItem` may be placed partially or fully off of the visible area of the parent by the `setPosition` method; in this case the `CanvasItem` is just not fully visible.

**Parameters:**
- `y` - the y coordinate of the anchor point, in pixels.

### setSize

```
public void setSize(int width,  
   int height)
```

Sets the size of this `CanvasItem` in pixels.

**Parameters:**
- `width` - width in pixels
- `height` - height in pixels

**Throws:**
- `IllegalArgumentException` - if the width or height is less than one pixel

### setVisible

```
public void setVisible(boolean visible)
```

Sets the visibility value of the `CanvasItem`. Initially the `CanvasItem` is not visible so it must be explicitly set to visible in order it to appear on the user interface.

If the `CanvasItem` is already visible, calling `setVisible(true)` does nothing. If the editor is already hidden calling `setVisible(false)` does nothing.

**Parameters:**
- `visible` - true to set visible, false to set invisible

**Throws:**
- `java.lang.IllegalStateException` - if the item is not added to any parent

### setZPosition

```
public void setZPosition(int z)
```

Sets the Z-position, or the elevation, of the item. The elevation decides the stacking order of neighboring items. An item of high Z-position will be drawn on top of an item with a lower Z-position if they share the same parent item.

Modifying Z-value of an item should not impact Z-position of other items, so there may be items with the same Z-position. Items that share the same Z-position will be drawn in an undefined order, although the order will stay the same for as long as the items live.

The Z-position does not affect the item's size in any way.

When items are added with `setParent` they will get a Z-position that is increased by 1 from the previously added item. However if `setZPosition` has been called before calling `setParent` then setting the parent does not change the already set Z-position of this `CanvasItem`.

**Parameters:**
- `z` - the Z-position of the item

**Throws:**
- `java.lang.IllegalArgumentException` - if `z < 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 font attribute applies to the text part and can be controlled by the application. 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:

```java
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 setFitPolicy(int) and getFitPolicy() methods control this
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 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 \texttt{setSelectedFlags} and \texttt{setSelectedIndex} methods. This state is available to the application through the \texttt{getSelectedFlags} and \texttt{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 \texttt{CommandListener} or a \texttt{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).
Disabling a Choice element automatically marks the state of the choice element as not-selected. Hence, a disabled choice element once enabled using `setEnabled(i, true)` will be in not-selected state (except for EXCLUSIVE/POPUP/IMPLICIT Choice, where at least one element MUST be selected, see details in next paragraph). Attempts to set the state of a disabled choice element using a `setSelectedIndex` or `setSelectedFlags` will have no effect on the state of the element.

In the case of EXCLUSIVE/IMPLICIT/POPUP Choice elements, if there were no enabled elements in the Choice and `setEnabled(i, true)` was called to enable element i, then this choice element at index i is marked selected automatically. If the currently selected Choice element is disabled using `setEnabled(i, false)`, the implementation MUST look for the next Choice element that is enabled in this Choice to mark as selected. If the end of the Choice is reached and no enabled element is found, the implementation MUST continue to look for an enabled element starting from index 0. It is possible that no enabled Choice elements exist in the Choice, and hence no element is marked selected in this EXCLUSIVE/IMPLICIT/POPUP Choice.

**Note:** Methods have been added to the Choice interface in version 2.0. Adding methods to interfaces is normally an incompatible change. However, Choice does not appear as a type in any field, method parameter, or method return value, and so it is not useful for an application to create a class that implements the Choice interface. Future versions of this specification may make additional changes to the Choice interface. In order to remain compatible with future versions of this specification, applications should avoid creating classes that implement the Choice interface.

**Since:** MIDP 1.0

---

**Field Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public static final EXCLUSIVE</code></td>
<td>EXCLUSIVE</td>
<td>is a choice having exactly one element selected at time. Value: 1</td>
</tr>
<tr>
<td><code>public static final IMPLICIT</code></td>
<td>IMPLICIT</td>
<td>is a choice in which the currently focused element is selected when a Command is initiated. Value: 3</td>
</tr>
<tr>
<td><code>public static final MULTIPLE</code></td>
<td>MULTIPLE</td>
<td>is a choice that can have arbitrary number of elements selected at a time. Value: 2</td>
</tr>
<tr>
<td><code>public static final POPUP</code></td>
<td>POPUP</td>
<td>is a choice having exactly one element selected at a time. Value: 4</td>
</tr>
<tr>
<td><code>public static final TEXT_WRAP_DEFAULT</code></td>
<td>TEXT_WRAP_DEFAULT</td>
<td>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. Value: 0</td>
</tr>
<tr>
<td><code>public static final TEXT_WRAP_OFF</code></td>
<td>TEXT_WRAP_OFF</td>
<td>Constant for hinting that text element contents should be limited to a single line. Value: 2</td>
</tr>
<tr>
<td><code>public static final TEXT_WRAP_ON</code></td>
<td>TEXT_WRAP_ON</td>
<td>Constant for hinting that text element contents should be wrapped to multiple lines if necessary to fit available content space. Value: 1</td>
</tr>
</tbody>
</table>
### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int append(String stringPart, Image imagePart)</code></td>
<td>Appends an element to the Choice.</td>
</tr>
<tr>
<td><code>void delete(int elementNum)</code></td>
<td>Deletes the element referenced by elementNum.</td>
</tr>
<tr>
<td><code>void deleteAll()</code></td>
<td>Deletes all elements from this Choice, leaving it with zero elements.</td>
</tr>
<tr>
<td><code>int getFitPolicy()</code></td>
<td>Gets the application's preferred policy for fitting Choice element contents to the available screen space.</td>
</tr>
<tr>
<td><code>javax.microedition.lcdui.Font getFont(int elementNum)</code></td>
<td>Gets the application's preferred font for rendering the specified element of this Choice.</td>
</tr>
<tr>
<td><code>javax.microedition.lcdui.Image getImage(int elementNum)</code></td>
<td>Gets the Image part of the element referenced by elementNum.</td>
</tr>
<tr>
<td><code>int getSelectedFlags(boolean[] selectedArray_return)</code></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><code>int getSelectedIndex()</code></td>
<td>Returns the index number of an element in the Choice that is selected.</td>
</tr>
<tr>
<td><code>java.lang.String getString(int elementNum)</code></td>
<td>Gets the String part of the element referenced by elementNum.</td>
</tr>
<tr>
<td><code>void insert(int elementNum, String stringPart, Image imagePart)</code></td>
<td>Inserts an element into the Choice just prior to the element specified.</td>
</tr>
<tr>
<td><code>boolean isEnabled(int elementNum)</code></td>
<td>Gets a boolean value indicating whether the specified Choice element is enabled or disabled.</td>
</tr>
<tr>
<td><code>boolean isSelected(int elementNum)</code></td>
<td>Gets a boolean value indicating whether this element is selected.</td>
</tr>
<tr>
<td><code>void set(int elementNum, String stringPart, Image imagePart)</code></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><code>void setSelectedFlags(boolean[] selectedArray)</code></td>
<td>Attempts to set the selected state of every element in the Choice.</td>
</tr>
<tr>
<td><code>void setEnabled(int elementNum, boolean isEnabled)</code></td>
<td>Sets the Choice element into enabled or disabled mode.</td>
</tr>
<tr>
<td><code>void setFitPolicy(int fitPolicy)</code></td>
<td>Sets the application's preferred policy for fitting Choice element contents to the available screen space.</td>
</tr>
<tr>
<td><code>void setFont(int elementNum, Font font)</code></td>
<td>Sets the application's preferred font for rendering the specified element of this Choice.</td>
</tr>
</tbody>
</table>

**Related Classes:**
- `javax.microedition.lcdui.Item`
void setSelectedIndex(int elementNum, boolean selected)
For MULTIPLE, this simply sets an individual element's selected state.

int size()
Gets the number of elements present.

**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: getFitPolicy(), 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: getFitPolicy(), 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: getFitPolicy(), setFitPolicy(int)
Since: MIDP 2.0

Methods

append

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

public void delete(int elementNum)
Delete 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, starting from zero

Throws:
- IndexOutOfBoundsException - if elementNum is invalid

### deleteAll

```java
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 TEXT_WRAP_DEFAULT, TEXT_WRAP_ON, or TEXT_WRAP_OFF

See Also: 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: 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.

The elementNum parameter must be within the range [0..size()-1], inclusive.

Parameters:
- elementNum - the index of the element to be queried, starting from zero
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: setSelectedFlags(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, starting from zero

Returns:
    the string part of the element

Throws:
    IndexOutOfBoundsException - if elementNum is invalid

See Also: 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, starting from zero
- 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

isEnabled

public boolean isEnabled(int elementNum)

Gets a boolean value indicating whether the specified Choice element is enabled or disabled. The elementNum parameter must be within the range [0..size()-1], inclusive.

Parameters:
- elementNum: the index of the element to be queried, starting from zero

Returns:
- state of the element: true if enabled, false if disabled

Throws:
- IndexOutOfBoundsException: if elementNum is invalid

Since: MIDP 3.0

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, starting from zero

Returns:
- selection state of the element

Throws:
- IndexOutOfBoundsException: if elementNum is invalid

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. 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, starting from zero
- `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`

### setEnabled

```java
public void setEnabled(int elementNum, boolean isEnabled)
```

Sets the **Choice** element into enabled or disabled mode. The set mode applies for the element of the **Choice** object specified by the `elementNum` parameter. A disabled **Choice** element SHOULD be visually different from an enabled **Choice** element.

The `elementNum` parameter must be within the range `[0..size()-1]`, inclusive.

**Parameters:**
- `elementNum` - the index of the element to which the mode change is applied, the index starts from zero
- `isEnabled` - true if enabled, false if disabled

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

**See Also:** `isEnabled(int)`

**Since:** MIDP 3.0

### 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 `TEXT_WRAP_DEFAULT`, `TEXT_WRAP_ON`, and `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:** `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.
javax.microedition.lcdui.Choice

**setSelectedFlags**

```java
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 collect the selection status

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

**See Also:** `getSelectedFlags(boolean[])`

**setSelectedIndex**

```java
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:** `getSelectedIndex()`

**size**

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

Gets the number of elements present.
javax.microedition.lcdui.Choice

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

ChoiceGroup

Declaration

public class ChoiceGroup extends Item implements Choice

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.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>deleteAll()</code></td>
<td>Deletes all elements from this ChoiceGroup.</td>
</tr>
<tr>
<td><code>getFitPolicy()</code></td>
<td>Gets the application's preferred policy for fitting Choice element contents to the available screen space.</td>
</tr>
<tr>
<td><code>getFont(int elementNum)</code></td>
<td>Gets the application's preferred font for rendering the specified element of this Choice.</td>
</tr>
<tr>
<td><code>getImage(int elementNum)</code></td>
<td>Gets the Image part of the element referenced by elementNum.</td>
</tr>
<tr>
<td><code>getSelectedFlags(boolean[] selectedArray_return)</code></td>
<td>Queries the state of a ChoiceGroup and returns the state of all elements in the boolean array selectedArray_return.</td>
</tr>
<tr>
<td><code>getSelectedIndex()</code></td>
<td>Returns the index number of an element in the ChoiceGroup that is selected.</td>
</tr>
<tr>
<td><code>getString(int elementNum)</code></td>
<td>Gets the String part of the element referenced by elementNum.</td>
</tr>
<tr>
<td><code>insert(int elementNum, String stringPart, Image imagePart)</code></td>
<td>Inserts an element into the ChoiceGroup just prior to the element specified.</td>
</tr>
<tr>
<td><code>isEnabled(int elementNum)</code></td>
<td>Gets a boolean value indicating whether the specified Choice element is enabled or disabled.</td>
</tr>
<tr>
<td><code>isSelected(int elementNum)</code></td>
<td>Gets a boolean value indicating whether this element is selected.</td>
</tr>
<tr>
<td><code>set(int elementNum, String stringPart, Image imagePart)</code></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><code>setEnabled(int elementNum, boolean isEnabled)</code></td>
<td>Sets the Choice element into enabled or disabled mode.</td>
</tr>
<tr>
<td><code>setFitPolicy(int fitPolicy)</code></td>
<td>Sets the application's preferred policy for fitting Choice element contents to the available screen space.</td>
</tr>
<tr>
<td><code>setFont(int elementNum, Font font)</code></td>
<td>Sets the application's preferred font for rendering the specified element of this Choice.</td>
</tr>
<tr>
<td><code>setSelectedFlags(boolean[] selectedArray)</code></td>
<td>Attempts to set the selected state of every element in the ChoiceGroup.</td>
</tr>
<tr>
<td><code>setSelectedIndex(int elementNum, boolean selected)</code></td>
<td>For ChoiceGroup objects of type MULTIPLE, this simply sets an individual element's selected state.</td>
</tr>
<tr>
<td><code>size()</code></td>
<td>Returns the number of elements in the ChoiceGroup.</td>
</tr>
</tbody>
</table>

Methods inherited from class `javax.microedition.lcdui.Item`
Methods inherited from class Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from interface javax.microedition.lcdui.Choice

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
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.

The elementNum parameter must be within the range [0..size()-1], inclusive.

Parameters:
elementNum - the index of the element to be deleted, starting from zero

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: 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:
- setFont(int, Font)

Since: MIDP 2.0

getSelectedFlags

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:
- setSelectedFlags(boolean[])

getSelectedIndex

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: setSelectedIndex(int, boolean)

**getString**

```java
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, starting from zero

Returns:
- the string part of the element

Throws:
- IndexOutOfBoundsException - if `elementNum` is invalid

See Also: 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.

The `elementNum` parameter must be within the range `[0..size()-1]`, inclusive.

Parameters:
- `elementNum`: the index of the element where insertion is to occur, starting from zero
- `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

**isEnabled**

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

Gets a boolean value indicating whether the specified Choice element is enabled or disabled.

The `elementNum` parameter must be within the range `[0..size()-1]`, inclusive.

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

Returns:
- state of the element: true if enabled, false if disabled

Throws:
- IndexOutOfBoundsException - if `elementNum` is invalid
Since: MIDP 3.0

**isSelected**

```java
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, starting from zero

**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.

The `elementNum` parameter must be within the range `[0..size()-1]`, inclusive.

**Parameters:**
- `elementNum` - the index of the element to be set, starting from zero
- `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`

**setEnabled**

```java
public void setEnabled(int elementNum, boolean isEnabled)
```

Sets the `Choice` element into enabled or disabled mode. The set mode applies for the element of the `Choice` object specified by the `elementNum` parameter. A disabled `Choice` element SHOULD be visually different from an enabled `Choice` element.

The `elementNum` parameter must be within the range `[0..size()-1]`, inclusive.

**Parameters:**
- `elementNum` - the index of the element to which the mode change is applied, the index starts from zero
- `isEnabled` - true if enabled, false if disabled

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

**See Also:** `isEnabled(int)`

**Since:** MIDP 3.0

**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:
   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: 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:
   IllegalArgumentExcepption - if selectedArray is shorter than the size of the ChoiceGroup
   NullPointerException - if the selectedArray is null

See Also: getSelectedFlags(boolean[])

setSelectedIndex

public void setSelectedIndex(int elementNum, boolean selected)

Mobile Information Device Profile v3.0 - JSR 271
Java Community Process - Final Release
© Copyright Motorola, Inc. 2002-2009. All Rights Reserved.
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 index of the element, starting from zero
- `selected` - the new state of the element (true=selected, false=not selected)

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

**See Also:** `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. If both short and long labels are too long to fit the command space, the short command label will be truncated to the available space.
From MIDP 3.0 the visible parts of a Command is mutable. This means that the labels may be set at any
time by the setLabel() and setLongLabel() methods. It is up to the implementation to act on the
change as soon as possible, at latest before the next invocation of the CommandListener for the
Command in question.

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.

**Exact placement of Commands**

MIDP 3.0 adds the possibility of the application using exact placement of Commands when used as softkeys by using arguments to the addCommand method. How this is done is explained in the `Displayable` class.

**Enabled**

The application uses the enabled value to specify whether a command is enabled or not. If a command is not enabled, it should be rendered differently, for instance grayed out, and it should not be available for selection by the user.

**Images**

The application may add an Image to the Command to be shown together with the labels. The device shows the image on a best effort basis, depending on what can be done on the particular display. For
example, a Command with an image is added to two different displays, one with high resolution and one with low, the implementation may decide either to show the image only on the high resolution display or ignore the image completely. Implementations may truncate or scale the icon image if it is larger than the size supported by device.

The Image may be mutable or immutable. If the Image is mutable, then a snapshot of its contents is taken before the Command() constructor or setImage() method returns. The snapshot is used whenever the contents of the Command are to be displayed. Even if the application subsequently draws into the Image, the snapshot is not modified until the next call to setImage(). If the Command is visible on the display then the display SHOULD be updated with the new snapshot as soon as it is feasible for the implementation to do so.

Applications can query the implementation's command icon size by calling Display.getBestImageWidth(int) and Display.getBestImageHeight(int) methods using the Display.COMMAND image type. The style and appearance of commands are platform-dependent. If the image is changed after the Command is first displayed, then it is up to the implementation to act on the change as soon as possible, at latest before the next invocation of the CommandListener for the Command in question.

Since: MIDP 3.0

<table>
<thead>
<tr>
<th>Field Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
## Constructor Summary

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

## Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int getCommandType()</code></td>
<td>Gets the type of the command.</td>
</tr>
<tr>
<td><code>boolean getEnabled()</code></td>
<td>Gets the enabled value of the command.</td>
</tr>
<tr>
<td><code>javax.microedition.lcdui.Font getFont()</code></td>
<td>Gets the application's preferred font for rendering this <code>Command</code>.</td>
</tr>
<tr>
<td><code>javax.microedition.lcdui.Image getImage()</code></td>
<td>Gets the image of the command.</td>
</tr>
<tr>
<td><code>java.lang.String getLabel()</code></td>
<td>Gets the short label of the command.</td>
</tr>
<tr>
<td><code>java.lang.String getLongLabel()</code></td>
<td>Gets the long label of the command.</td>
</tr>
<tr>
<td><code>int getPriority()</code></td>
<td>Gets the priority of the command.</td>
</tr>
<tr>
<td><code>void onParentEnabled(boolean enabled)</code></td>
<td>Informs the command of a change in its parent's enabled value.</td>
</tr>
<tr>
<td><code>void setEnabled(boolean enabled)</code></td>
<td>Sets the enabled value of the command.</td>
</tr>
<tr>
<td><code>void setFont(Font font)</code></td>
<td>Sets the application's preferred font for rendering this <code>Command</code>.</td>
</tr>
<tr>
<td><code>void setImage(Image image)</code></td>
<td>Sets the image of the command.</td>
</tr>
<tr>
<td><code>void setLabel(String label)</code></td>
<td>Sets the label of the command.</td>
</tr>
<tr>
<td><code>void setLongLabel(String longLabel)</code></td>
<td>Sets the long label of the command.</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: CANCEL, STOP

**CANCEL**

`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: BACK, STOP

**EXIT**

`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**

`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**

`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

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: CANCEL

SCREEN

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

STOP

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: BACK, CANCEL

Constructors

Command

public Command(String label,
    int commandType,
    int priority)
javax.microedition.lcdui.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
**getCommandType**

public int getCommandType()  

Gets the type of the command.  

**Returns:**  

type of the Command

**getFont**

public javax.microedition.lcdui.Font getFont()  

Gets the application's preferred font for rendering this Command. A Command's font is a hint, and the implementation may disregard the application's preferred font.  

**Returns:**  

font preferred font to use to render the command; null if the system is using the default font.

**See Also:** setFont(Font)

**Since:** MIDP 3.0

**getEnabled**

public boolean getEnabled()  

Gets the enabled value of the command.  

**Returns:**  

enabled value of the Command

**Since:** MIDP 3.0

**getImage**

public javax.microedition.lcdui.Image getImage()  

Gets the image of the command. The image may be null if the Command has no image.  

**Returns:**  

image of the Command

**Since:** MIDP 3.0

**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

getPriority

public int getPriority()

Gets the priority of the command.

Returns:
    priority of the Command

onParentEnabled

public void onParentEnabled(boolean enabled)

Informs the command of a change in its parent's enabled value. The application must itself decide how the Command should handle this information, the default is to do nothing.

Since: MIDP 3.0

setEnabled

public void setEnabled(boolean enabled)

Sets the enabled value of the command.

Since: MIDP 3.0

setFont

public void setFont(Font font)

Sets the application's preferred font for rendering this Command. A Command's 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 element.

Parameters:
    font - the preferred font to use to render the command

See Also: getFont()

Since: MIDP 3.0

setImage

public void setImage(Image image)

Sets the image of the command. If the image is mutable then a snapshot is created before this method returns.

For example, after painting into a mutable image contained by a Command, the application refreshes the snapshot by calling:

    command.setImage(command.getImage());

Parameters:
    image - the command's image; may be null.
Since: MIDP 3.0

setLabel

public void setLabel(String label)

Sets the label of the command.

Throws:

- NullPointerException - if label is null

Since: MIDP 3.0

setLongLabel

public void setLongLabel(String longLabel)

Sets the long label of the command.

Since: MIDP 3.0
javax.microedition.lcdui

CommandLayoutPolicy

Declaration

public interface CommandLayoutPolicy

Description

This interface is used to implement exact placement of commands.

Example:
javax.microedition.lcdui.CommandLayoutPolicy

class MyCanvas extends Canvas implements CommandLayoutPolicy {
    MyCanvas() {
        setCommandLayoutPolicy(this);
    }

    void sort(Command[] commands, int[] positions) {
        // sort the commands in the correct order depending on the positions available.
        // Implementation can use Display's getCommandPreferredPlacements to get the
        // recommended
        // placement for each Command.
    }

    public void onCommandLayout(Displayable displayable) {
        Display display = displayable.getCurrentDisplay();

        final int border = Display.SOFTKEY_BOTTOM;

        int[] positions = display.getExactPlacementPositions(border);

        int numOfPositions = positions.length;

        Command[] cmd = displayable.getCommands();

        sort(cmd, positions);

        if (cmd.length <= numOfPositions) {
            for (int i = 0; i < cmd.length; ++i) {
                displayable.setCommand(cmd[i], positions[i]);
            }
        } else {
            for (int i = 0; i < numOfPositions ; ++i) {
                displayable.setCommand(cmd[i], positions[i]);
            }
        }

        int[] menuSupportedPlmts = display.getMenuSupportedPlacements();
        if( menuSupportedPlmts != null ) {
            Menu options = new Menu("More", null, null);
            // first add the remaining commands in the Menu
            for (int i = numOfPositions; i < cmd.length; ++i) {
                options.append(cmd[i]);
            }
            // Get the first preferred placement of Menu
            int menuPlmt = menuSupportedPlmts[0];
            // check if this placement already occupied by Command
            if( displayable.getCommand(menuPlmt) != null ) {
                //add the existing Command in Menu, otherwise the existing
                // Command will be removed from the Displayable according to
                // setMenu() method
                //
                options.append(displayable.getCommand(menuPlmt));
            }
        }
    }
}
javax.microedition.lcdui.CommandLayoutPolicy

displayable.setMenu(options, menuPlmt);
}
}
}

Since: MIDP 3.0

Method Summary

<table>
<thead>
<tr>
<th>void</th>
<th>onCommandLayout(Displayable displayable)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Method called when a new layout is needed.</td>
</tr>
</tbody>
</table>

Methods

onCommandLayout

public void onCommandLayout (Displayable displayable)

Method called when a new layout is needed. Any Command or Menu not explicitly set at a placement (via Displayable.setCommand(Command, int), Displayable.setMenu(Menu, int), or Item.setCommand(Command, int)) in the CommandLayoutPolicy implementation will be ignored and not displayed.

Parameters:

displayable - The Displayable that holds the commands that should be updated. If displayable is a TabbedPane, it is the CommandLayoutPolicy implementation's responsibility to update commands depending on the active tab etc. If displayable is a Form, it is the CommandLayoutPolicy implementation's responsibility to update commands depending on the active item etc.

Since: MIDP 3.0
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>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandAction</td>
<td>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

Direct Known Subclasses:

javax.microedition.lcdui.IdleItem

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 content changes. This indicates to the implementation that it may need to perform its layout computation, which will call the content size methods to get new values based on the CustomItem's new contents.
Interaction Modes

The `CustomItem` class is intended to allow edit-in-place on many items, but it does not allow every conceivable interaction. Desire for flexibility has been balanced against a requirement that these APIs be simple enough to master easily, along with a need to allow for platform-specific variations in look-and-feel, all without sacrificing interoperability.

The general idea is that there are multiple interaction "modes" and that the Form implementation can convey which ones it supports. The `CustomItem` can then choose to support one or more interaction modes. There is no requirement for a `CustomItem` to implement all combinations of all interaction modes. Typically, a `CustomItem` will implement an approach (such as the separate editing screen technique discussed below) that works on all platforms, in addition to a highly interactive approach that relies on a particular interaction mode. At run time, the `CustomItem` code can query the system to determine whether this interaction mode is supported. If it is, the `CustomItem` can use it; otherwise, it will fall back to the approach that works on all platforms.

`CustomItem` can always use item commands to invoke a separate editing screen, although components with a small number of discrete states could simply respond by changing the state and then causing an `notifyStateChanged` notification. A technique for using a separate editing screen would be to load the value into another `Displayable` object (such as a List) and then to call `Display.setCurrent(Displayable)` on it. When the user issues a command (such as "OK") to indicate that editing of this value is complete, the listener can retrieve the value from that `Displayable` object and then call `Display.setCurrentItem(Item)` 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_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 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 actions such that all 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 actions in a `CustomItem` may differ from the mapping in a `Canvas`. See `Key Events` and `Actions` on class `Canvas` for further information about key codes and actions.

Pointer Input

The implementation may optionally support delivery of pointer events (such as taps with a stylus) to the `CustomItem`. The implementation indicates the level of support by setting the `POINTER_PRESS`, `POINTER_RELEASE`, and `POINTER_DRAG` bits in the value returned by `getInteractionModes`. Events
corresponding to these bits are delivered through calls to the `pointerPressed()`, `pointerReleased()`, and `pointerDragged()` methods, respectively. If an implementation supports `POINTER_RELEASE` events, it must also support `POINTER_PRESS` events. If an implementation supports `POINTER_DRAG` events, it must also support `POINTER_PRESS` and `POINTER_RELEASE` events. If supported, `POINTER_RELEASE` events will generally occur after a corresponding `POINTER_PRESS` event is received, and `POINTER_DRAG` events will generally occur between `POINTER_PRESS` and `POINTER_RELEASE` events. However, it is possible for the `CustomItem` to receive `POINTER_RELEASE` or `POINTER_DRAG` events without a corresponding `POINTER_PRESS` if the pointer is down when the `CustomItem` becomes visible.

The \((x,y)\) location of the pointer event is reported with every pointer event. This location is expressed in the coordinate system of the `CustomItem`, where \((0,0)\) is the upper-left corner of the `CustomItem`. Under certain circumstances, pointer events may occur outside the bounds of the item.

### Traversal

An implementation may support traversal *internal* to a `CustomItem`, that is, the implementation may temporarily delegate the responsibility for traversal to the item itself. Even if there is only one traversal location inside the `CustomItem`, the item may want to support the internal traversal protocol so that it can perform specialized highlighting, animation, etc. when the user has traversed into it.

The implementation indicates its support for traversal internal to a `CustomItem` by setting one or both of the `TRAVERSE_HORIZONTAL` or `TRAVERSE_VERTICAL` bits in the value returned by `getInteractionModes()` . If neither of these bits is set, the implementation is unwilling to let `CustomItems` traverse internally, or the implementation does not support traversal at all. If the implementation does support traversal but has declined to permit traversal internal to `CustomItems`, the implementation will supply its own highlighting outside the `CustomItem`'s content area.

The `CustomItem` need not support internal traversal at all. It can do this by returning `false` to the initial call to the `traverse` method. (This is the default behavior if this method hasn't been overridden by the `CustomItem`.) If this occurs, the system must arrange for the user to be able to traverse onto and past this item. The system must also arrange for proper scrolling to take place, particularly if the item exceeds the height of the screen, regardless of whether internal traversal is occurring.

An implementation may provide support for delivering keypad or pointer events to `CustomItems` even if it has declined to support delivering traversal events to `CustomItems`. If an implementation provides support for delivering keypad or pointer events to `CustomItems`, it must provide a means to do so for every `CustomItem`, even for those that have refused internal traversal by returning `false` to the initial `traverse()` call. This implies that such implementations must still support some notion of focus for an item, even if that item is not supporting internal traversal.

See the documentation for the `traverse` method for a full specification of the behavior and responsibilities required for the item to perform internal traversal.

### Item Appearance

The visual appearance of each item consists of a label (handled by the implementation) and its contents (handled by the subclass).

Labels are the responsibility of the implementation, not the item. The screen area that is allocated to the `CustomItem` for its contents is separate from the area that the implementation uses to display the `CustomItem`'s label. The implementation controls the rendering of the label and its layout with respect to the content area.

The `CustomItem` is responsible for painting its contents whenever the `paint` method is called.

The colors for foreground, background, highlighted foreground, highlighted background, border, and highlighted border should be retrieved from `Display.getColor(int)`. This will allow `CustomItems` to match the color scheme of other items provided with the device. The `CustomItem` is responsible for keeping track of its own highlighted and unhighlighted state.
The fonts used should be retrieved from `Font.getFont(String, int, int)`. This will allow them to match the fonts used by other items on the device for a consistent visual appearance.

CanvasItem subclasses, such as a TextEditor, may be drawn on Canvas or CustomItem. See CanvasItem and TextEditor for more details.

Since: MIDP 2.0

---

### Field Summary

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protected static final</td>
<td>KEY_PRESS Interaction mode bit indicating support for key pressed events. Value: 4</td>
</tr>
<tr>
<td>protected static final</td>
<td>KEY_RELEASE Interaction mode bit indicating support for key released events. Value: 8</td>
</tr>
<tr>
<td>protected static final</td>
<td>KEY_REPEAT Interaction mode bit indicating support for key repeated events. Value: 16</td>
</tr>
<tr>
<td>protected static final</td>
<td>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</td>
</tr>
<tr>
<td>protected static final</td>
<td>POINTER_DRAG Interaction mode bit indicating support for point dragged events. Value: 128</td>
</tr>
<tr>
<td>protected static final</td>
<td>POINTER_PRESS Interaction mode bit indicating support for point pressed events. Value: 32</td>
</tr>
<tr>
<td>protected static final</td>
<td>POINTER_RELEASE Interaction mode bit indicating support for point released events. Value: 64</td>
</tr>
<tr>
<td>protected static final</td>
<td>TRAVERSE_HORIZONTAL Interaction mode bit indicating support of horizontal traversal internal to the CustomItem. Value: 1</td>
</tr>
<tr>
<td>protected static final</td>
<td>TRAVERSE_VERTICAL Interaction mode bit indicating support for vertical traversal internal to the CustomItem. Value: 2</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_EXPAND, LAYOUT_VSHRINK, PLAIN

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

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int getGameAction(int keyCode)</td>
<td>Gets the action associated with the given key code of the device.</td>
</tr>
<tr>
<td>int getInteractionModes()</td>
<td>Gets the available interaction modes.</td>
</tr>
<tr>
<td>int getKeyCode(int action)</td>
<td>Gets a key code that corresponds to the specified action on the device.</td>
</tr>
<tr>
<td>abstract int getMinContentHeight()</td>
<td>Implemented by the subclass to return the minimum height of the content area, in pixels.</td>
</tr>
<tr>
<td>abstract int getMinContentWidth()</td>
<td>Implemented by the subclass to return the minimum width of the content area, in pixels.</td>
</tr>
<tr>
<td>abstract int 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 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 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 invalidate()</td>
<td>Signals that the CustomItem's size and traversal location need to be updated.</td>
</tr>
<tr>
<td>void keyPressed(int keyCode)</td>
<td>Called by the system when a key is pressed.</td>
</tr>
<tr>
<td>void keyReleased(int keyCode)</td>
<td>Called by the system when a key is released.</td>
</tr>
<tr>
<td>void keyRepeated(int keyCode)</td>
<td>Called by the system when a key is repeated.</td>
</tr>
<tr>
<td>abstract void 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 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 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>
</tbody>
</table>
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.

void repaint()
   Called by subclass code to request that the item be repainted.

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.

void setKeyListener(KeyListener listener)
   Sets a listener for key events to this CustomItem, replacing any previous KeyListener.

void setPaintMode(boolean opaque)
   Sets the paint mode for this CustomItem.

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.

void sizeChanged(int w, int h)
   Implemented by the subclass in order to handle size change events.

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.

void traverseOut()
   Called by the system when traversal has occurred out of the item.

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

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

Fields

KEY_PRESS
protected static final int KEY_PRESS
   Interaction mode bit indicating support for key pressed events.

   KEY_PRESS has the value 4.
   Constant value: 4
See Also: getInteractionModes(), keyPressed(int)

KEY_RELEASE
protected static final int KEY_RELEASE

Mobile Information Device Profile v3.0 - JSR 271
© Copyright Motorola, Inc. 2002-2009. All Rights Reserved.
Java Community Process - Final Release
Page 291 of 891
Interaction mode bit indicating support for key released events.

**KEY_RELEASE** has the value 8.
Constant value: 8
See Also: `getInteractionModes()`, `keyReleased(int)`

---

**KEY_REPEAT**

protected static final int **KEY_REPEAT**

Interaction mode bit indicating support for key repeated events.

**KEY_REPEAT** has the value 0x10.
Constant value: 16
See Also: `getInteractionModes()`, `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: `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: `getInteractionModes()`, `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: `getInteractionModes()`, `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: `getInteractionModes()`, `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: getInteractionModes(), 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: getInteractionModes(), traverse(int, int, int, int[])

Constructors

CustomItem

protected CustomItem(String label)

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

Parameters:
  label - the CustomItem's label

Methods

getGameAction

public int getGameAction(int keyCode)

Gets the action associated with the given key code of the device. Returns zero if no action is associated with this key code. See the Actions section of class Canvas for further discussion of actions.

The mapping of key codes to actions may differ between CustomItem and Canvas.

Parameters:
  keyCode - the key code

Returns:
  the 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.
getKeyCode

public int getKeyCode(int action)

Gets a key code that corresponds to the specified action on the device. The implementation is required to provide a mapping for every action, so this method will always return a valid key code for every action. There may be multiple keys associated with the same action; however, this method will return only one of them. Applications should translate the key code of every key event into a action using Canvas.getGameAction() and then interpret the resulting action, instead of generating a table of key codes at using this method during initialization. See the Actions section of class Canvas for further discussion of actions.

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

Parameters:
  action - the action

Returns:
a key code corresponding to this action

Throws:
IllegalArgumentException - if action is not a valid action

Since: MIDP 3.0

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

```java
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

```java
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

```java
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)
```

Supported on most terminals.

Closest to the bottom of the screen, this method is called whenever the user presses a key. The keyCode parameter is an integer specifying the key which was pressed. For Example, on most terminals the 'A' key has a keyCode of 0x41 (65 in decimal).
Called by the system when a key is pressed. The implementation indicates support for delivery of key press events by setting the KEY_PRESS bit in the value returned by the getInteractionModes method.

**Parameters:**
- keyCode - the key code of the key that has been pressed

**See Also:** getInteractionModes()

---

### 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:** 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:** getInteractionModes()

---

### paint

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

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, unless the paint mode has been set to transparent (see CustomItem.setPaintMode). 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 POINTER_DRAG bit in the value returned by the getInteractionModes method.

Parameters:
- \(x\) - the \(x\) coordinate of the pointer drag
- \(y\) - the \(x\) coordinate of the pointer drag

See Also: getInteractionModes()

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 POINTER_PRESS bit in the value returned by the getInteractionModes method.

Parameters:
- \(x\) - the \(x\) coordinate of the pointer down
- \(y\) - the \(y\) coordinate of the pointer down

See Also: getInteractionModes()

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 \(x\) coordinate of the pointer up

See Also: getInteractionModes()

repaint

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

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 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

setKeyListener

public void setKeyListener(KeyListener listener)

Sets a listener for key events to this CustomItem, replacing any previous KeyListener. A null reference is allowed and has the effect of removing any existing listener. The set of key delivery methods of CustomItem (keyPressed(), keyRepeated() and keyReleased()) MUST work even if a KeyListener has been set, however the application should use only one of the ways to listen to key events. The method call order of CustomItem methods and KeyListener methods is undefined.

Parameters:
   listener - the new listener, or null to remove a listener.

Since: MIDP 3.0

setPaintMode

public void setPaintMode(boolean opaque)
Sets the paint mode for this CustomItem.

If the paint mode is *opaque*, the implementation may assume that the every pixel in the clip region will be rendered by the `paint` method. Thus, the implementation is not required to clear or reset the pixels to a suitable state prior to calling the `paint` method since they will all be rendered by the CustomItem.

In this example, the opaque paint mode is used and the paint method renders a white background and black text, thus fully obscuring the device’s background wallpaper:

```java
paint(Graphics g) {
    g.setColor(0xFFFFFF);
    g.fillRect(0, 0, getWidth(), getHeight());
    g.setColor(0x000000);
    g.drawString("Some Text", 10, 10, Graphics.TOP + Graphics.LEFT);
}
```

If the paint mode is *transparent*, the implementation is responsible for appropriately filling the entire clip region prior to calling the `paint` method. Some devices may fill the pixels to a suitable background color, while others may fill them with a suitable background image. Hence, any pixels untouched by the `paint` method will be colored appropriately. In this mode, the `paint` method does not need to render every pixel within the clip region; it should render only those pixels within the clip region that it wishes to control the contents of. By rendering a subset of the pixels in this manner, the contents of the CustomItem will appear rendered on top of the background content provided by the implementation.

In this example, the transparent paint mode is used and the paint method renders only the black text, thus making it appear on top of the device's background wallpaper:

```java
paint(Graphics g) {
    g.setColor(0x000000);
    g.drawString("Some Text", 10, 10, Graphics.TOP + Graphics.LEFT);
}
```

The paint mode is opaque by default.

**Parameters:**
- `opaque` - true to set the paint mode to opaque, false to set it to transparent

**See Also:** `paint(Graphics, int, int)`

**Since:** MIDP 3.0

### 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

protected void sizeChanged(int w, int h)

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

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 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 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 actions are used to indicate the traversal direction, this does not imply that the
keys mapped to these actions are being used for traversal, nor 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: getInteractionModes(), traverseOut(), TRAVERSE_HORIZONTAL, TRAVERSE_VERTICAL

traverseOut

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: getInteractionModes(), traverse(int, int, int, int[]), TRAVERSE_HORIZONTAL, TRAVERSE_VERTICAL
javax.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 the date object may not contain same millisecond value when the value of the field is set and subsequently read back.

Since: MIDP 1.0

Field Summary

<table>
<thead>
<tr>
<th>Method Type</th>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final</td>
<td>DATE</td>
<td>Input mode for date information (day, month, year). Value: 1</td>
</tr>
<tr>
<td>public static final</td>
<td>DATE_TIME</td>
<td>Input mode for date (day, month, year) and time (minutes, hours) information. Value: 3</td>
</tr>
<tr>
<td>public static final</td>
<td>TIME</td>
<td>Input mode for time information (hours and minutes). Value: 2</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
javax.microedition.lcdui.DateField

<table>
<thead>
<tr>
<th>public</th>
<th>DateField(String label, int mode)</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>DateField(String label, int mode, TimeZone timeZone)</td>
</tr>
</tbody>
</table>

`DateField` creates a `DateField` object with the specified label and mode. It also 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>java.util.Date</th>
<th>getDate()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns date value of this field.</td>
</tr>
<tr>
<td>int</td>
<td>getInputModule()</td>
</tr>
<tr>
<td></td>
<td>Gets input mode for this date field.</td>
</tr>
<tr>
<td>void</td>
<td>setDate(Date date)</td>
</tr>
<tr>
<td></td>
<td>Sets a new value for this field.</td>
</tr>
<tr>
<td>void</td>
<td>setInputMode(int mode)</td>
</tr>
<tr>
<td></td>
<td>Set input mode for this date field.</td>
</tr>
</tbody>
</table>

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

`addCommand`, `getCommands`, `getLabel`, `getLayout`, `getLayoutHint`, `getMinimumHeight`, `getMinimumWidth`, `getPreferredSize`, `getPreferredSize`, `notifyStateChanged`, `removeCommand`, `setCommand`, `setDefaultCommand`, `setItemCommandListener`, `setLabel`, `setLayout`, `setLayoutHint`, `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`.
Constant value: 2

### Constructors

**DateField**

```java
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**

```java
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

**getDateTime**

```java
public java.util.Date getDateTime()
```

Returns the 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:** `setDateTime(Date)`

**getInputMode**

```java
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: 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: 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: getInputMode()
javax.microedition.lcdui

Display

Declaration

public class Display

Object

+-javax.microedition.lcdui.Display

Description

The Display class provides a MIDlet with access to the device's user interface hardware resources. It includes static methods for obtaining Display objects as well as instance methods to retrieve their properties and display user interface objects on them.

Display Resources

A MIDP device has one or more display resources for interacting with the user. Each resource includes a graphical display device and may also include keys, touchscreen input, etc.

A MIDlet gains access to a display resource using a Display object. Each Display object represents a specific MIDlet's use of a specific resource, rather than the resource itself. Hence, for a given display resource, each MIDlet has its own dedicated Display object for accessing it.

Display States

Multiple MIDlets may want to use the same display resource simultaneously. Some aspects of a display resource are exclusive in nature; that is, they can be made available to only one MIDlet at a time. For example, regular key events are typically provided to a single application. However, other resources may be non-exclusive, thereby allowing the resource to be simultaneously allocated to several MIDlets. For example, space on a large display can be split up into several windows, each of which is allocated to a specific application.

The nature of specific resources and the policies for sharing them between applications are largely platform dependent. However, the Display class defines three states that should be used in conjunction with the platform's policies for sharing display resources.

When a Display has foreground status, it is given the highest priority for accessing the corresponding display resource. Exclusive aspects of the resource are only available to the foreground Display, and it also has the highest priority to use non-exclusive aspects of the resource. For a given display resource, there can be no more than one foreground Display. If the resource is also used by the native UI of the device, there may be times when no Display is in the foreground.

When a Display has background status, all of its display resources are relinquished and unavailable to the MIDlet. For a given set of display resources, there can be any number of Displays that are in the background state. A Display is initially in this state and it cannot be changed unless a Displayable is shown on it.

When a Display is visible, it has access to at least one pixel on the display resource. However, it will not have access to exclusive aspects of the resource. The application may implicitly request that a Display be brought to the foreground or sent to the background by calling the Display's setCurrent method. However, these requests to change the Display state are not guaranteed and are subject to the device's application window management behavior.
Display Hardware States

The underlying hardware resource represented by a Display may not always be available. For example, the main display of a mobile phone is not usable while the flip is closed. Furthermore, some resources are dynamic (e.g. a Bluetooth enabled projector) and their availability may be temporary in nature.

Therefore, a Display object has a hardware state to indicate the state of its underlying resource. This state refers to the display resource itself, rather than a specific MIDlet's ability to use it. Thus, all Displays that refer to the same hardware have the same hardware state, regardless of which MIDlet they belong to.

The hardware life-cycle is defined with three states:

- **DISPLAY_HARDWARE_ENABLED** - The Display's hardware is active and potentially available for use by the MIDlet. Attempts to use the Display should be successful, but due to the asynchronous nature of the API and possible system use, immediate availability of the resource is not guaranteed.

- **DISPLAY_HARDWARE_DISABLED** - The Display's hardware is inactive and currently cannot be used to interact with the user. For example, the main display of a mobile phone is disabled while the flip is closed. When a display resource is disabled, all of the corresponding Displays are temporarily placed in the background state. While the hardware is disabled, calls to `setCurrent()`, `removeCurrent()`, and `setCurrentItem()` should still be honored, but the Display MUST remain in the background. When the hardware is enabled again, the Display state should be restored to the most recent foreground/background request made; if no requests were made while the hardware was disabled, then the restored state should simply be the Display's state immediately before the hardware became disabled.
• **DISPLAY_HARDWARE_ABSENT** - This state only applies to an Auxiliary display hardware resource and it indicates that the resource is no longer accessible by the device. Attempts to use an absent Display will result in an exception being thrown. Displays that represent absent hardware are automatically placed in the background state. MIDlets should not retain references to absent Displays since they cannot be re-used. If the same Auxiliary display hardware becomes accessible once more, a new Display object is created to access it.

## Obtaining Display Objects

For every MIDlet, there is always one *primary* Display object. The primary Display corresponds to the device's main display that is normally used to access its features. The application can get a reference to the primary Display 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.

In addition to the primary Display, one or more secondary Displays may also be available to a MIDlet. These Displays correspond to secondary display devices such as the second screen on the outside of a mobile phone's flip; they may also include auxiliary display devices that are available to the device via a suitable connection.

The `getDisplays(int)` method returns a list of Display objects corresponding to a given MIDlet. The primary Display object is always returned as the first element in the list, followed by any secondary Display objects that are also available.

## Display Capabilities

The primary Display must support all of the user interface features defined in the LCDUI package; however, secondary Displays might be less capable and may support only a subset of the features.

The capabilities of a given Display can be queried by calling the `getCapabilities()` method. This method returns an integer containing a bit field representation of the Display's capabilities. The bit field will contain a combination of the following values:

- **SUPPORTS_COMMANDS** : A Display must have this capability in order to present Commands
- **SUPPORTS_INPUT_EVENTS** : A Display must have this capability in order to receive raw input events (as provided to a Canvas or CustomItem)
- **SUPPORTS_FORMS** : A Display must have this capability in order to show Forms
- **SUPPORTS_TICKER** : A Display must have this capability in order to show Displayables that include a Ticker
- **SUPPORTS_TITLE** : A Display must have this capability in order to show Displayables that include a Title
- **SUPPORTS_ALERTS** : A Display must have this capability in order to show Alerts
- **SUPPORTS_LISTS** : A Display must have this capability in order to show Lists
- **SUPPORTS_TEXTBOXES** : A Display must have this capability in order to show TextBoxes
- **SUPPORTS_FILESELECTORS** : A Display must have this capability in order to show FileSelectors
- **SUPPORTS_TABBEDPANELS** : A Display must have this capability in order to show TabbedPanels
- **SUPPORTS_MENUS** : A Display must have this capability in order to show Menus
- **SUPPORTS_IDLEITEM** : A Display must have this capability in order to display content on the idle screen
- **SUPPORTS_ORIENTATION_PORTrait** : A Display must have this capability if the longer dimension of the screen is its height
- **SUPPORTS_ORIENTATION_LANDscape** : A Display must have this capability if the longer dimension of the screen is its width
- **SUPPORTS_ORIENTATION_PORTrait180** : A Display must have this capability if the longer dimension of the screen is its height and content is rotated 180 degrees.
- **SUPPORTS_ORIENTATION_LANDscape180** : A Display must have this capability if the longer dimension of the screen is its width and content is rotated 180 degrees.
Basic support for rendering a Canvas is required for all Displays and is assumed if none of these capabilities are available.

A given Display's capabilities are static; that is, they do not change over time or in response to the state of the device. If a device includes a hardware resource whose capabilities are dynamic, then it must be represented as multiple Displays, with each one having a static set of capabilities and being enabled and disabled based on the current state of the device.

For example, the following device has a single physical screen but it operates quite differently with the flip open versus closed. With the flip in the open position, the screen is 240 x 320 pixels and includes touchscreen support, and thus it has support for all Display capabilities. However, with the flip closed, the screen is much smaller and has no input capabilities other than a numeric keypad. In this mode, very few capabilities would be available, perhaps just Tickers and Titles.

From the MIDlet's perspective, this device would have two Display objects corresponding to the two flip positions. Display A would be 240 x 320 pixels and have support for all capabilities; it would be enabled only while the flip is open. Display B is smaller and can only support Tickers and Titles and has limited key events instead of pointer events; it would be enabled only while the flip is closed.

Displayables

The user interface objects that are shown on a Display are contained within a Displayable object. At any time the application may have at most one Displayable Object to be shown on a given Display. This Displayable is referred to as the current Displayable. A Displayable is also considered to be current on a given Display if it is to be shown pending the dismissal of an Alert, see the method setCurrent(Alert, Displayable)

If the device has multiple Displays, a Displayable is allowed to be used on multiple Displays, but it can only be current on one Display at any given time. Displayable subclasses may require specific Display capabilities in order to function correctly. A DisplayCapabilityException will be thrown if the specified Display does not meet the needs of the Displayable sub-class that is being passed into the setCurrent methods; details about the Display requirements for a specific Displayable can be found in the documentation for that class. In addition, changes to a Displayable that will require capabilities not provided by its Display will also result in an exception being thrown. For example, the use of Commands on a Displayable will require that its Display can support commands; if they are not supported, a DisplayCapabilityException is thrown.

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 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 detect 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.

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, or if the current Displayable has been removed by calling removeCurrent. The getCurrent() method will never return a reference to a Displayable object that was not passed in a prior call to setCurrent() 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.

Color Scheme Data

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.

For a given MIDlet, each `Display` object may have a different color scheme to reflect different display hardware capabilities. For example, a color scheme used on the main color display might not be suitable for a secondary grayscale display, so it may employ a unique color scheme to ensure sufficient contrast.

The color scheme values are subject to skinning and theme settings on the device including those provided via JSR 258. Hence, the `Display` objects for different MIDlets may have different color scheme values even if they correspond to the same display resource.

Adding content to the idle screen

In MIDP 3.0 a new `setIdleItem` method is added to allow the MIDlet to add or remove the content shown on the idle screen. A corresponding `getIdleItem` method is provided to query what content is currently on the idle screen. Not all `Display` objects will support items on idle screen. The `getCapabilities` method can be used to test for idle support on a `Display`.

Each MIDlet has its own instance of a `Display` class. This means that there can be more than one idle screen MIDlet shown on the idle at a time. However one instance of the `Display` class can contain only one item on the idle screen.

Colors on the idle screen

The `Display` class already provides a mechanism for MIDlets to query system colors, by means of color specifiers and the `getColor` method. This mechanism is expanded to include four new color specifiers for the idle screen.

Exact Placement of Commands

In MIDP 3.0 the method of using `type` and `priority` for Commands placement is supplemented with a method to use absolute placement. In general, an implementation can not know how many command placements that are available until a Displayable is assigned to a Display by a call to `setCurrent()`.

To implement exact placement of commands the MIDlet must implement the `CommandLayoutPolicy` interface. See `CommandLayoutPolicy` for an example implementation of a custom exact placement policy.

A `CommandLayoutPolicy` instance can be set either on a `Display` or on a `Displayable`. The platform chooses which `CommandLayoutPolicy` to use in the following priority order:
• 1) If the Displayable has a CommandLayoutPolicy set, that policy will be used.
• 2) Otherwise, if the Display has a CommandLayoutPolicy, that policy will be used.
• 3) If neither the Displayable nor the Display has a CommandLayoutPolicy, the platform default layout will be used.

The CommandLayoutPolicy interface contains a layout method (onCommandLayout) that will be called when commands needs to be placed. This is typically done when changing orientation, scrolling in forms, changing tabs in a TabbedPane, Displayable being set as current for the first time and when adding/removing commands from a Displayable/Item. It will also be called if the native implementation changes the number of available command placements of the current Displayable. CommandLayoutPolicy.onCommandLayout (Displayable) MUST NOT be called if the Displayable has not yet been attached to a Display (via a call to the setCurrent () method).

When one of the layout methods is called it is up to the CommandLayoutPolicy implementation to query the current Displayable about which command positions that are available at that time and which commands that should be placed.

In an implementation of the CommandLayoutPolicy it is possible to use the Display.getExactPlacementPositions(int), Displayable.setMenu(Menu, int), Displayable.setCommand(Command, int), Display.getMenuPreferredPlacements(), Display.getCommandPreferredPlacements(int) and Displayable.getCommands() to implement the desired command layout.

If the CommandLayoutPolicy implementation throws an exception the implementation will revert to the Display's policy (if any). If the Display policy also fails, or no Display policy exists, the platform default Command layout is used instead. The available placements are retrieved using getExactPlacementPositions(int). A placement is encoded as a base value for each border plus an index of the soft key along the border. The borders are labeled SOFTKEY_BOTTOM, SOFTKEY_LEFT, SOFTKEY_TOP, SOFTKEY_RIGHT and SOFTKEY_OFFSCREEN, each represented by a field (for example, Display.SOFTKEY_BOTTOM). The SOFTKEY_OFFSCREEN represents keys not on the screen. In the presence of multiple active screens, the Commands on the SOFTKEY_OFFSCREEN MUST be the same for all Displays. The index along a border are ordered from one to max number of commands on that border (no more than 15), counting from top to bottom or from left to right respectively. The indices are ordered according to:

<table>
<thead>
<tr>
<th>Border</th>
<th>Index order</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOFTKEY_TOP or SOFTKEY_BOTTOM (horizontal)</td>
<td>LEFT to RIGHT</td>
</tr>
<tr>
<td>SOFTKEY_LEFT or SOFTKEY_RIGHT (vertical)</td>
<td>TOP to BOTTOM</td>
</tr>
<tr>
<td>SOFTKEY_OFFSCREEN</td>
<td>Device dependent</td>
</tr>
</tbody>
</table>

The getExactPlacementPositions(int) method returns an integer array, each element containing a possible exact placement for the Display associated with the Displayable.

For example, consider a display that has four different placements where commands may be placed, three on the lower bottom border and one as a fixed key not directly tied to the screen. In this case:

```java
int[] bottomPlacements = getExactPlacementPositions(Display.SOFTKEY_BOTTOM);
int[] offscreenPlacements = getExactPlacementPositions(Display.SOFTKEY_OFFSCREEN);
```

... would return bottomPlacement = [801, 802, 803] and offscreenPlacements = [881]. To continue the example, there are four valid placements available to the application: 801, 802, 803, and 881, see the figure below:

The getExactPlacementPositions(int) method allows the application to fetch valid placements. The application may then use Display.getCommandPreferredPlacements method to inquire the placements, if any, associated with a particular command type.
For example, assume that `Command.BACK` type command in normally placed on the dedicated key (placement 881 in the example above) on a given device. This information can be revealed by calling:

```java
int[] backPlacements = display.getCommandPreferredPlacements(Command.BACK);
// backPlacements = [881]
```

Placing a `Command` object on the `SOFTKEY_OFFSCREEN` border places the object outside of the `Displayable`’s visible area (typically mapped to a certain key).

Note that the set of placements available on a particular `Displayable` are set by the device, and this set cannot be changed by the application. This means that the application developer can specify where a `Command` is placed from a list of placements, but the developer can not add more placements to the list.

For this reason, developers who implement their own `CommandLayoutPolicy` should to consider the case where the number of `Commands` exceeds the number of available placements. If such a condition exists, the `CommandLayoutPolicy` should use the `Menu` class to put some of the commands into menus. Any `Command` or `Menu` not explicitly set at a placement (via `Displayable.setCommand(Command, int)`, `Displayable.setMenu(Menu, int)`, or `Item.setCommand(Command, int)`) in the `CommandLayoutPolicy` implementation will be ignored and not displayed. On each callback of `CommandLayoutPolicy.onCommandLayout(Displayable)`, implementation MUST first disassociate `Command` and `Menu` with their respective previous placements. At the beginning of each callback of `CommandLayoutPolicy.onCommandLayout(Displayable)` method, `Displayable.getCommand(int)` or `Displayable.getMenu(int)` will return `null` for every placement.

**Since:** MIDP 1.0

---

### Field Summary

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>display</td>
<td>A <code>Displayable</code> object</td>
</tr>
<tr>
<td>backPlacements</td>
<td>An array of placements for the <code>Command.BACK</code> command</td>
</tr>
<tr>
<td>SOFTKEY_OFFSCREEN</td>
<td>A border placement</td>
</tr>
<tr>
<td>Command</td>
<td>A <code>Command</code> object</td>
</tr>
<tr>
<td>Menu</td>
<td>A <code>Menu</code> object</td>
</tr>
<tr>
<td>Item</td>
<td>An <code>Item</code> object</td>
</tr>
</tbody>
</table>

---

`javax.microedition.lcdui.Display`
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final ALERT</td>
<td>Image type for Alert image.</td>
<td>3</td>
</tr>
<tr>
<td>public static final CHOICE_GROUP_ELEMENT</td>
<td>Image type for ChoiceGroup element image.</td>
<td>2</td>
</tr>
<tr>
<td>public static final COLOR_BACKGROUND</td>
<td>A color specifier for use with <code>getColor</code>.</td>
<td>0</td>
</tr>
<tr>
<td>public static final COLOR_BORDER</td>
<td>A color specifier for use with <code>getColor</code>.</td>
<td>4</td>
</tr>
<tr>
<td>public static final COLOR_FOREGROUND</td>
<td>A color specifier for use with <code>getColor</code>.</td>
<td>1</td>
</tr>
<tr>
<td>public static final COLOR_HIGHLIGHTED_BACKGROUND</td>
<td>A color specifier for use with <code>getColor</code>.</td>
<td>2</td>
</tr>
<tr>
<td>public static final COLOR_HIGHLIGHTED_BORDER</td>
<td>A color specifier for use with <code>getColor</code>.</td>
<td>5</td>
</tr>
<tr>
<td>public static final COLOR_HIGHLIGHTED_FOREGROUND</td>
<td>A color specifier for use with <code>getColor</code>.</td>
<td>3</td>
</tr>
<tr>
<td>public static final COLOR_IDLE_BACKGROUND</td>
<td>Color specifier for the background of items on idle screen</td>
<td>6</td>
</tr>
<tr>
<td>public static final COLOR_IDLE_FOREGROUND</td>
<td>Color specifier for the foreground of items on idle screen</td>
<td>7</td>
</tr>
<tr>
<td>public static final COLOR_IDLE_HIGHLIGHTED_BACKGROUND</td>
<td>Color specifier for the background of focused items on idle screen</td>
<td>8</td>
</tr>
<tr>
<td>public static final COLOR_IDLE_HIGHLIGHTED_FOREGROUND</td>
<td>Color specifier for the foreground of focused items on idle screen</td>
<td>9</td>
</tr>
<tr>
<td>public static final COMMAND</td>
<td>Image type for images used in Commands.</td>
<td>5</td>
</tr>
<tr>
<td>public static final DISPLAY_HARDWARE_ABSENT</td>
<td>Hardware state indicating that the display hardware is no longer connected to the device and cannot be used.</td>
<td>2</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>DISPLAY_HARDWARE_DISABLED</td>
<td>Hardware state indicating that the display hardware is disabled and its contents are not visible to the user.</td>
<td>1</td>
</tr>
<tr>
<td>DISPLAY_HARDWARE_ENABLED</td>
<td>Hardware state indicating that the display hardware is enabled and available for use.</td>
<td>0</td>
</tr>
<tr>
<td>LIST_ELEMENT</td>
<td>Image type for List element image.</td>
<td>1</td>
</tr>
<tr>
<td>MENU</td>
<td>Image type for images used in Menus.</td>
<td>7</td>
</tr>
<tr>
<td>MODE_ACTIVE</td>
<td>Activity mode indicating that power-saving actions should be deferred for as long as possible to maximize the user’s visibility of the Display.</td>
<td>1</td>
</tr>
<tr>
<td>MODE_NORMAL</td>
<td>Activity mode indicating that normal power management behavior should be applied.</td>
<td>0</td>
</tr>
<tr>
<td>NOTIFICATION</td>
<td>Image type for Notification icon image.</td>
<td>6</td>
</tr>
<tr>
<td>ORIENTATION_LANDSCAPE</td>
<td>Orientation mode in which the longer dimension of the screen is its width.</td>
<td>2</td>
</tr>
<tr>
<td>ORIENTATION_LANDSCAPE_180</td>
<td>Orientation mode in which the longer dimension of the screen is its width and content is rotated 180 degrees.</td>
<td>8</td>
</tr>
<tr>
<td>ORIENTATION_PORTrait</td>
<td>Orientation mode in which the longer dimension of the screen is its height.</td>
<td>1</td>
</tr>
<tr>
<td>ORIENTATION_PORTrait_180</td>
<td>Orientation mode in which the longer dimension of the screen is its height and content is rotated 180 degrees.</td>
<td>4</td>
</tr>
<tr>
<td>SOFTKEY_BOTTOM</td>
<td>Specifies the bottom (or lower) horizontal border on the screen.</td>
<td>800</td>
</tr>
<tr>
<td>SOFTKEY_INDEX_MASK</td>
<td>Mask to isolate the index of the soft key.</td>
<td>15</td>
</tr>
<tr>
<td>Method</td>
<td>Constant Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>public static final</td>
<td>SOFTKEY_LEFT</td>
<td>Specifies the left vertical border on the screen.</td>
</tr>
<tr>
<td>public static final</td>
<td>SOFTKEY_OFFSCREEN</td>
<td>Specifies a placement not on a screen.</td>
</tr>
<tr>
<td>public static final</td>
<td>SOFTKEY_RIGHT</td>
<td>Specifies the right vertical border on the screen.</td>
</tr>
<tr>
<td>public static final</td>
<td>SOFTKEY_TOP</td>
<td>Specifies the top (or upper) horizontal border on the screen.</td>
</tr>
<tr>
<td>public static final</td>
<td>STATE_BACKGROUND</td>
<td>Background state in which the Display is invisible and has no access to user interface related resources.</td>
</tr>
<tr>
<td>public static final</td>
<td>STATE_FOREGROUND</td>
<td>Foreground state in which the Display is visible and has priority over user interface related resources.</td>
</tr>
<tr>
<td>public static final</td>
<td>STATE_VISIBLE</td>
<td>Visible state in which the Display is at least partially visible but has limited access to user interface related resources.</td>
</tr>
<tr>
<td>public static final</td>
<td>SUPPORTS_ALERTS</td>
<td>The capability of supporting Alerts.</td>
</tr>
<tr>
<td>public static final</td>
<td>SUPPORTS_COMMANDS</td>
<td>The capability of supporting Commands.</td>
</tr>
<tr>
<td>public static final</td>
<td>SUPPORTS_FILESELECTORS</td>
<td>The capability of supporting FileSelectors.</td>
</tr>
<tr>
<td>public static final</td>
<td>SUPPORTS_FORMS</td>
<td>The capability of supporting Forms.</td>
</tr>
<tr>
<td>public static final</td>
<td>SUPPORTS_IDLEITEM</td>
<td>The capability of supporting an idle screen.</td>
</tr>
<tr>
<td>public static final</td>
<td>SUPPORTS_INPUT_EVENTS</td>
<td>The capability of receiving raw input events from the user.</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 SUPPORTS_LISTS</td>
<td>The capability of supporting Lists.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value: 64</td>
<td></td>
</tr>
<tr>
<td>public static final SUPPORTS_MENUS</td>
<td>The capability of supporting Menus.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value: 1024</td>
<td></td>
</tr>
<tr>
<td>public static final SUPPORTS_ORIENTATION_LANDSCAPE</td>
<td>The capability of supporting an orientations of physical display in which the longer dimension of the screen is its width.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value: 8192</td>
<td></td>
</tr>
<tr>
<td>public static final SUPPORTS_ORIENTATION_LANDSCAPE180</td>
<td>The capability of supporting an orientations of physical display in which the longer dimension of the screen is its width and content is rotated 180 degrees.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value: 32768</td>
<td></td>
</tr>
<tr>
<td>public static final SUPPORTS_ORIENTATION портрет</td>
<td>The capability of supporting an orientations of physical display in which the longer dimension of the screen is its height.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value: 4096</td>
<td></td>
</tr>
<tr>
<td>public static final SUPPORTS_ORIENTATION портрет180</td>
<td>The capability of supporting an orientations of physical display in which the longer dimension of the screen is its height and content is rotated 180 degrees.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value: 16384</td>
<td></td>
</tr>
<tr>
<td>public static final SUPPORTS_TABBEDPANES</td>
<td>The capability of supporting TabbedPanes.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value: 256</td>
<td></td>
</tr>
<tr>
<td>public static final SUPPORTS_TEXTBOXES</td>
<td>The capability of supporting TextBoxes.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value: 128</td>
<td></td>
</tr>
<tr>
<td>public static final SUPPORTS_TICKER</td>
<td>The capability of showing Tickers to the user.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value: 8</td>
<td></td>
</tr>
<tr>
<td>public static final SUPPORTS_TITLE</td>
<td>The capability of showing Titles to the user.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value: 16</td>
<td></td>
</tr>
<tr>
<td>public static final TAB</td>
<td>Image type for TabbedPane icon image.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value: 4</td>
<td></td>
</tr>
</tbody>
</table>

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void addDisplayListener(DisplayListener l)</td>
<td>Adds a listener to receive Display notifications.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>callSerially(Runnable r)</code></td>
<td>Causes the <code>Runnable</code> object to have its <code>run()</code> method called later, serialized with the event stream, soon after completion of the repaint cycle.</td>
</tr>
<tr>
<td><code>flashBacklight(int duration)</code></td>
<td>Requests a flashing effect for the device's backlight.</td>
</tr>
<tr>
<td><code>getActivityMode()</code></td>
<td>Gets the activity mode of this Display.</td>
</tr>
<tr>
<td><code>getBestImageHeight(int imageType)</code></td>
<td>Returns the best image height for a given image type.</td>
</tr>
<tr>
<td><code>getBestImageWidth(int imageType)</code></td>
<td>Returns the best image width for a given image type.</td>
</tr>
<tr>
<td><code>getBorderStyle(boolean highlighted)</code></td>
<td>Returns the stroke style used for border drawing depending on the state of the component (highlighted/non-highlighted).</td>
</tr>
<tr>
<td><code>getCapabilities()</code></td>
<td>Gets the capabilities of this Display.</td>
</tr>
<tr>
<td><code>getColor(int colorSpecifier)</code></td>
<td>Returns one of the colors from the high level user interface color scheme, in the form 0xAARRGGBB based on the <code>colorSpecifier</code> passed in.</td>
</tr>
<tr>
<td><code>getCommandLayoutPolicy()</code></td>
<td>This method will return the current CommandLayoutPolicy for this Display.</td>
</tr>
<tr>
<td><code>getCommandPreferredPlacements(int commandType)</code></td>
<td>Returns array of integers containing the preferred placements normally associated with Command type <code>commandType</code>.</td>
</tr>
<tr>
<td><code>getCurrent()</code></td>
<td>Gets the current <code>Displayable</code> object for this Display.</td>
</tr>
<tr>
<td><code>getDisplay(MIDlet m)</code></td>
<td>Gets the primary Display object that is unique to this MIDlet.</td>
</tr>
<tr>
<td><code>getDisplays(int capabilities)</code></td>
<td>Gets a list of all the Displays for this MIDlet.</td>
</tr>
<tr>
<td><code>getDisplayState()</code></td>
<td>Gets the state of this Display.</td>
</tr>
<tr>
<td><code>getDotPitch()</code></td>
<td>Gets the dot pitch measurement of the Display.</td>
</tr>
<tr>
<td><code>getExactPlacementPositions(int border)</code></td>
<td>This will get the possible exact placement positions for the given border on this Display.</td>
</tr>
<tr>
<td><code>getHardwareState()</code></td>
<td>Gets the state of the hardware for this Display.</td>
</tr>
<tr>
<td><code>getHeight()</code></td>
<td>Gets the height in pixels of the Display.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>javax.microedition.lcdui.IdleItem.getIdleItem()</td>
<td>Returns the current IdleItem on the Display.</td>
</tr>
<tr>
<td>int[] getMenuPreferredPlacements()</td>
<td>Returns an array of integers containing the preferred placements normally associated with Menu.</td>
</tr>
<tr>
<td>int[] getMenuSupportedPlacements()</td>
<td>Returns an array of integers containing the placements that support Menus.</td>
</tr>
<tr>
<td>int getOrientation()</td>
<td>Gets the current orientation of this Display.</td>
</tr>
<tr>
<td>int getWidth()</td>
<td>Gets the width in pixels of the Display.</td>
</tr>
<tr>
<td>boolean hasPointerEvents()</td>
<td>Checks if the Display supports pointer press and release events.</td>
</tr>
<tr>
<td>boolean hasPointerMotionEvents()</td>
<td>Checks if the Display supports pointer motion events (pointer dragged).</td>
</tr>
<tr>
<td>boolean isBuiltIn()</td>
<td>Checks if this Display's hardware is Built-In or Auxiliary.</td>
</tr>
<tr>
<td>boolean isColor()</td>
<td>Gets information about color support of the Display.</td>
</tr>
<tr>
<td>int numAlphaLevels()</td>
<td>Gets the number of alpha transparency levels supported by this Display.</td>
</tr>
<tr>
<td>int numColors()</td>
<td>Gets the number of colors (if isColor() is true) or graylevels (if isColor() is false) that can be represented on the display.</td>
</tr>
<tr>
<td>void removeCurrent()</td>
<td>Removes the current Displayable from this Display.</td>
</tr>
<tr>
<td>static void removeDisplayListener(DisplayListener l)</td>
<td>Removes a Display listener.</td>
</tr>
<tr>
<td>void setActivityMode(int mode)</td>
<td>Sets the activity mode of this Display.</td>
</tr>
<tr>
<td>void setCommandLayoutPolicy(CommandLayoutPolicy policy)</td>
<td>This method will set a new CommandLayoutPolicy for this Display.</td>
</tr>
<tr>
<td>void setCurrent(Alert alert, Displayable nextDisplayable)</td>
<td>Requests that this Alert be made current, and that nextDisplayable be made current after the Alert is dismissed.</td>
</tr>
<tr>
<td>void setCurrent(Displayable nextDisplayable)</td>
<td>Requests that a different Displayable object be made visible on the display.</td>
</tr>
<tr>
<td>void setCurrentItem(Item item)</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void <a href="#setidleitem">setIdleItem</a> (<a href="#idleitem">IdleItem</a> idleItem)</td>
<td>Sets the specified <a href="#idleitem">IdleItem</a> to the <a href="#display">Display</a>.</td>
</tr>
<tr>
<td>void <a href="#setpreferredorientation">setPreferredOrientation</a> (int orientation)</td>
<td>Sets the preferred orientation for this <a href="#display">Display</a>.</td>
</tr>
<tr>
<td>boolean <a href="#vibrate">vibrate</a> (int duration)</td>
<td>Requests operation of the device's vibrator.</td>
</tr>
</tbody>
</table>

**Methods inherited from class `Object`**

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

### Fields

**ALERT**

public static final int `ALERT`

- Image type for Alert image.
- The value of `ALERT` is 3.
- Constant value: 3

**See Also:** [getBestImageWidth(int)](#getbestimagewidth), [getBestImageHeight(int)](#getbestimageheight)

**Since:** MIDP 2.0

**CHOICE_GROUP_ELEMENT**

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:** [getBestImageWidth(int)](#getbestimagewidth), [getBestImageHeight(int)](#getbestimageheight)

**Since:** MIDP 2.0

**COLOR_BACKGROUND**

public static final int `COLOR_BACKGROUND`

- A color specifier for use with [getColor](#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:** [getColor(int)](#getcolor)

**Since:** MIDP 2.0

**COLOR_BORDER**

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

**COLOR_IDLE_BACKGROUND**

```java
public static final int COLOR_IDLE_BACKGROUND
```

Color specifier for the background of items on idle screen

`COLOR_IDLE_BACKGROUND` can be used with `getColor` method to determine the background color of unfocused items on the idle screen. On some platforms this may be a transparent color.

`COLOR_IDLE_BACKGROUND` has the value 6.
Constant value: 6

Since: MIDP 3.0

**COLOR_IDLE_FOREGROUND**

```java
public static final int COLOR_IDLE_FOREGROUND
```

Color specifier for the foreground of items on idle screen

`COLOR_IDLE_FOREGROUND` can be used with `getColor` to determine the foreground color of unfocused items on the idle screen.

`COLOR_IDLE_FOREGROUND` has the value 7.
Constant value: 7

Since: MIDP 3.0

**COLOR_IDLE_HIGHLIGHTED_BACKGROUND**

```java
public static final int COLOR_IDLE_HIGHLIGHTED_BACKGROUND
```

Color specifier for the background of focused items on idle screen

`COLOR_IDLE_HIGHLIGHTED_BACKGROUND` can be used with `getColor` to determine the background color of focused items on the idle screen. This color should be used to render the background of items on the idle screen. On some platforms this may be a transparent color.

`COLOR_IDLE_HIGHLIGHTED_BACKGROUND` has the value 8.
Constant value: 8

Since: MIDP 3.0

**COLOR_IDLE_HIGHLIGHTED_FOREGROUND**

```java
public static final int COLOR_IDLE_HIGHLIGHTED_FOREGROUND
```

Color specifier for the foreground of focused items on idle screen

`COLOR_IDLE_HIGHLIGHTED_FOREGROUND` can be used with `getColor` to determine the foreground color of focused items on the idle screen (IdleItem).

`COLOR_IDLE_HIGHLIGHTED_FOREGROUND` has the value 9.
Constant value: 9
Since: MIDP 3.0

**COMMAND**

public static final int COMMAND

Image type for images used in Commands.
Constant value: 5

See Also: getBestImageWidth(int), getBestImageHeight(int)

Since: MIDP 3.0

**DISPLAY_HARDWARE_ABSENT**

public static final int DISPLAY_HARDWARE_ABSENT

Hardware state indicating that the display hardware is no longer connected to the device and cannot be used. This state only applies to auxiliary display hardware.
Constant value: 2

Since: MIDP 3.0

**DISPLAY_HARDWARE_DISABLED**

public static final int DISPLAY_HARDWARE_DISABLED

Hardware state indicating that the display hardware is disabled and its contents are not visible to the user.
Constant value: 1

Since: MIDP 3.0

**DISPLAY_HARDWARE_ENABLED**

public static final int DISPLAY_HARDWARE_ENABLED

Hardware state indicating that the display hardware is enabled and available for use.
Constant value: 0

Since: MIDP 3.0

**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: getBestImageWidth(int), getBestImageHeight(int)

Since: MIDP 2.0

**MENU**

public static final int MENU

Image type for images used in Menus.
Constant value: 7

See Also: getBestImageWidth(int), getBestImageHeight(int)

Since: MIDP 3.0
MODE_ACTIVE

public static final int MODE_ACTIVE

Activity mode indicating that power-saving actions should be deferred for as long as possible to maximize the user’s visibility of the Display.

Value 1 is assigned to MODE_ACTIVE.
Constant value: 1
Since: MIDP 3.0

MODE_NORMAL

public static final int MODE_NORMAL

Activity mode indicating that normal power management behavior should be applied.

Value 0 is assigned to MODE_NORMAL.
Constant value: 0
Since: MIDP 3.0

NOTIFICATION

public static final int NOTIFICATION

Image type for Notification icon image.
Constant value: 6
See Also: getBestImageWidth(int), getBestImageHeight(int)
Since: MIDP 3.0

ORIENTATION_LANDSCAPE

public static final int ORIENTATION_LANDSCAPE

Orientation mode in which the longer dimension of the screen is its width.
Constant value: 2
Since: MIDP 3.0

ORIENTATION_LANDSCAPE_180

public static final int ORIENTATION_LANDSCAPE_180

Orientation mode in which the longer dimension of the screen is its width and content is rotated 180 degrees.
Constant value: 8
Since: MIDP 3.0

ORIENTATION_PORTrait

public static final int ORIENTATION_PORTrait

Orientation mode in which the longer dimension of the screen is its height.
Constant value: 1
Since: MIDP 3.0
public static final int ORIENTATION_PORTRAIT_180

Orientation mode in which the longer dimension of the screen is its height and content is rotated 180 degrees.
Constant value: 4
Since: MIDP 3.0

public static final int SOFTKEY_BOTTOM

Specifies the bottom (or lower) horizontal border on the screen.

Value 800 is assigned to SOFTKEY_BOTTOM. The values 801..815 are reserved for soft keys on the bottom of the screen.
Constant value: 800
Since: MIDP 3.0

public static final int SOFTKEY_INDEX_MASK

Mask to isolate the index of the soft key. Only 15 soft keys can defined on each border. For example:

```java
int action = getGameAction(keycode);
int index = action & SOFTKEY_INDEX_MASK;
int border = action - index;
```

Value 15 is assigned to SOFTKEY_INDEX_MASK.
Constant value: 15
Since: MIDP 3.0

public static final int SOFTKEY_LEFT

Specifies the left vertical border on the screen.

Value 820 is assigned to SOFTKEY_LEFT. The values 821..835 are reserved for soft keys on the left of the screen.
Constant value: 820
Since: MIDP 3.0

public static final int SOFTKEY_OFFSCREEN

Specifies a placement not on a screen. For example, a dedicated key attached to a particular soft button.

Value 880 is assigned to SOFTKEY_OFFSCREEN. The values 881..895 are reserved for soft keys off the screen.
Constant value: 880
Since: MIDP 3.0

SOFTKEY_RIGHT

public static final int SOFTKEY_RIGHT

Specifies the right vertical border on the screen.

Value 860 is assigned to SOFTKEY_RIGHT. The values 861..875 are reserved for soft keys on the right of the screen.

Constant value: 860

Since: MIDP 3.0

SOFTKEY_TOP

public static final int SOFTKEY_TOP

Specifies the top (or upper) horizontal border on the screen.

Value 840 is assigned to SOFTKEY_TOP. The values 841..855 are reserved for soft keys on the top of the screen.

Constant value: 840

Since: MIDP 3.0

STATE_BACKGROUND

public static final int STATE_BACKGROUND

Background state in which the Display is invisible and has no access to user interface related resources.

Constant value: 0

Since: MIDP 3.0

STATE_FOREGROUND

public static final int STATE_FOREGROUND

Foreground state in which the Display is visible and has priority over user interface related resources.

Constant value: 2

Since: MIDP 3.0

STATE_VISIBLE

public static final int STATE_VISIBLE

Visible state in which the Display is at least partially visible but has limited access to user interface related resources.

Constant value: 1

Since: MIDP 3.0

SUPPORTS_ALERTS

public static final int SUPPORTS_ALERTS

The capability of supporting Alerts. The Display must be able to show the Alert and, if necessary, allow the user to respond to it.

Constant value: 32
Since: MIDP 3.0

SUPPORTS_COMMANDS

```java
public static final int SUPPORTS_COMMANDS
```

The capability of supporting Commands. The user is able to view the Commands and invoke them.
Constant value: 2

Since: MIDP 3.0

SUPPORTS_FILESELECTORS

```java
public static final int SUPPORTS_FILESELECTORS
```

The capability of supporting FileSelectors.
Constant value: 512

Since: MIDP 3.0

SUPPORTS_FORMS

```java
public static final int SUPPORTS_FORMS
```

The capability of supporting Forms. The Display must be able to support all features of the Form class and all Item subclasses including CustomItem. In addition to rendering the Form, the Display must also allow the user to interact with the appropriate Items.
Constant value: 4

Since: MIDP 3.0

SUPPORTS_IDLEITEM

```java
public static final int SUPPORTS_IDLEITEM
```

The capability of supporting an idle screen.
Constant value: 2048

Since: MIDP 3.0

SUPPORTS_INPUT_EVENTS

```java
public static final int SUPPORTS_INPUT_EVENTS
```

The capability of receiving raw input events from the user. The events may be in the form of keys events or pointer events.
Constant value: 1

Since: MIDP 3.0

SUPPORTS_LISTS

```java
public static final int SUPPORTS_LISTS
```

The capability of supporting Lists. The Display must be able to show a List, allow the user to scroll through its contents, and select an item in the List.
Constant value: 64

Since: MIDP 3.0

SUPPORTS_MENUS

```java
public static final int SUPPORTS_MENUS
```

The capability of supporting Menus.
Constant value: **1024**

**Since:** MIDP 3.0

---

**SUPPORTS_ORIENTATION_LANDSCAPE**

```java
public static final int SUPPORTS_ORIENTATION_LANDSCAPE
```

The capability of supporting an orientation of physical display in which the longer dimension of the screen is its width.

**Constant value:** **8192**

**Since:** MIDP 3.0

---

**SUPPORTS_ORIENTATION_LANDSCAPE180**

```java
public static final int SUPPORTS_ORIENTATION_LANDSCAPE180
```

The capability of supporting an orientation of physical display in which the longer dimension of the screen is its width and content is rotated 180 degrees.

**Constant value:** **32768**

**Since:** MIDP 3.0

---

**SUPPORTS_ORIENTATION_PORTRAIT**

```java
public static final int SUPPORTS_ORIENTATION_PORTRAIT
```

The capability of supporting an orientation of physical display in which the longer dimension of the screen is its height.

**Constant value:** **4096**

**Since:** MIDP 3.0

---

**SUPPORTS_ORIENTATION_PORTRAIT180**

```java
public static final int SUPPORTS_ORIENTATION_PORTRAIT180
```

The capability of supporting an orientation of physical display in which the longer dimension of the screen is its height and content is rotated 180 degrees.

**Constant value:** **16384**

**Since:** MIDP 3.0

---

**SUPPORTS_TABBEDPANES**

```java
public static final int SUPPORTS_TABBEDPANES
```

The capability of supporting TabbedPanels.

**Constant value:** **256**

**Since:** MIDP 3.0

---

**SUPPORTS_TEXTBOXES**

```java
public static final int SUPPORTS_TEXTBOXES
```

The capability of supporting TextBoxes. The Display must be able to show the TextBox and allow the user to enter or edit its contents.

**Constant value:** **128**

**Since:** MIDP 3.0
**SUPPORTS_TICKER**

public static final int SUPPORTS_TICKER

The capability of showing Tickers to the user.
Constant value: 8

**Since:** MIDP 3.0

---

**SUPPORTS_TITLE**

public static final int SUPPORTS_TITLE

The capability of showing Titles to the user.
Constant value: 16

**Since:** MIDP 3.0

---

**TAB**

public static final int TAB

Image type for TabbedPane icon image.
Constant value: 4

**See Also:** getBestImageWidth(int), getBestImageHeight(int)

**Since:** MIDP 3.0

### Methods

**addDisplayListener**

public static void addDisplayListener(DisplayListener l)

Adds a listener to receive Display notifications. The listener will be notified of changes to all Displays. Multiple listeners may be added, though the order in which they are notified of a given event is implementation dependent.

**Parameters:**
- l - the listener to receive events about the MIDlet's Displays

**Throws:**
- NullPointerException - if the listener is null

**Since:** MIDP 3.0

---

**callSerially**

public 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. As noted in the Event Handling section of the package summary, the methods that deliver event notifications to the application are all called serially. The call to r.run() will be serialized along with the event calls into the application. The run() method will be called exactly once for each call to callSerially(). Calls to run() will occur in the order in which they were requested by calls to callSerially().

If the current Displayable is a Canvas that has a repaint pending at the time of a call to callSerially(), the paint() method of the Canvas will be called and will return, and a buffer switch will occur (if double buffering is in effect), before the run() method of the Runnable is called. If the current Displayable contains one or more CustomItems that have repaints pending at the time of a call to callSerially(), the paint() methods of the CustomItems will be called and will return before the run() method of the Runnable is called. Calls to the run() method will occur in a timely fashion, but they are not guaranteed to occur immediately after the repaint cycle finishes, or even before the next event is delivered.

The callSerially() method may be called from any thread. The call to the run() method will occur independently of the call to callSerially(). In particular, callSerially() will never block waiting for r.run() to return.

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

The callSerially() facility may be used by applications to run an animation that is properly synchronized with the repaint cycle. A typical application will set up a frame to be displayed and then call repaint(). The application must then wait until the frame is actually displayed, after which the setup for the next frame may occur. The call to run() notifies the application that the previous frame has finished painting. The example below shows callSerially() being used for this purpose.

```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:
   r - instance of interface Runnable to be called

```
flashBacklight

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.

When the flashing effect is complete, the backlight MUST be restored to the state it was in
immediately before calling this method. BACKLIGHT system events MUST NOT be posted as a
result of the flashing effect.

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

```
getActivityMode

public int getActivityMode()
```

Gets the activity mode of this Display. The mode is an attribute of the Display and does not reflect the
current operating state of the underlying display resource. The activity mode is set to
MODE_NORMAL by default.

Returns:
   The activity mode for this Display (MODE_NORMAL or MODE_ACTIVE)

Since: MIDP 3.0

```
getBestImageHeight

public int getBestImageHeight(int imageType)
```

Returns the best image height for a given image type. The image type must be one of
LIST_ELEMENT, CHOICE_GROUP_ELEMENT, ALERT, TAB, COMMAND, NOTIFICATION, or MENU.

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
getBestImageWidth

```java
public int getBestImageWidth(int imageType)

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

**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:**
java.lang.IllegalArgumentException - if imageType is illegal

**Since:** MIDP 2.0

getBorderStyle

```java
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

getCapabilities

```java
public int getCapabilities()

Gets the capabilities of this Display. The returned integer value may be any combination of
SUPPORTS_INPUT_EVENTS, SUPPORTS_COMMANDS, SUPPORTS/forms, SUPPORTS/TICKER,
SUPPORTS_TITLE, SUPPORTS_ALERTS, SUPPORTS/LISTS, SUPPORTS/TEXTBOXES,
SUPPORTS/TABBEDPANES, SUPPORTS/FILESELECTORS, SUPPORTS/orientation/PORTRAIT,
SUPPORTS/orientation/LANDSCAPE, SUPPORTS/orientation.PORTRAIT180,
SUPPORTS/orientation/LANDSCAPE180, SUPPORTS/MENUS, SUPPORTS/IDLEITEM.

If 0 is returned, the Display has only basic capabilities for displaying graphical content in a Canvas.
```

**Returns:**
the capabilities of this Display

**Since:** MIDP 3.0

getColor

```java
public int getColor(int colorSpecifier)

Returns one of the colors from the high level user interface color scheme, in the form 0xARRGBB
based on the colorSpecifier passed in.
```

**Parameters:**
**getCommandLayoutPolicy**

```java
public javax.microedition.lcdui.CommandLayoutPolicy getCommandLayoutPolicy()
```

This method will return the current CommandLayoutPolicy for this Display.

**Returns:**
- The current CommandLayoutPolicy

**Since:** MIDP 3.0

**getCommandPreferredPlacements**

```java
public int[] getCommandPreferredPlacements(int commandType)
```

Returns array of integers containing the preferred placements normally associated with Command type commandType.

If the method is supplied with an invalid commandType, an IllegalArgumentException will be thrown.

**Parameters:**
- commandType - the command type to fetch placements for

**Returns:**
- the preferred placements for the given commandType on this Display, or null if no placements are associated to the commandType.

**Throws:**
- IllegalArgumentException - If the commandType is invalid.

**See Also:** Command

**Since:** MIDP 3.0

**getCurrent**

```java
public javax.microedition.lcdui.Displayable getCurrent()
```

Gets the current Displayable object for this Display. The Displayable object returned may not actually be visible on the device if the Display is in the background. 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 if a Displayable has yet to be shown on this Display or if removeCurrent has been called.

**Returns:**
- the MIDlet's current Displayable object

**See Also:** setCurrent (Displayable)

**getDisplay**

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

**Related Classes:**
- javax.microedition.lcdui.Displayable
- javax.microedition.lcdui.Command

**See Also:**
- Displayable
- Command

**Since:** MIDP 2.0

**colorSpecifier**

```java
javax.microedition.lcdui.Display colorSpecifier - the predefined color specifier; must be one of COLOR_BACKGROUND,
COLOR_FOREGROUND, COLOR_HIGHLIGHTED_BACKGROUND, COLOR_HIGHLIGHTED_FOREGROUND,
COLOR_BORDER, COLOR_HIGHLIGHTED_BORDER, COLOR_IDLE_BACKGROUND,
COLOR_IDLE_FOREGROUND, COLOR_IDLE_HIGHLIGHTED_BACKGROUND or
COLOR_IDLE_HIGHLIGHTED_FOREGROUND
```

**Returns:**
- color in the form of 0x00RRGGBB

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

**Since:** MIDP 2.0
getPrimaryDisplay

java.microedition.lcdui.Display

getPrimaryDisplay

javax.microedition.lcdui.Display

Get the primary Display object that is unique to this MIDlet.

Parameters:
- m - MIDlet of the application

Returns:
the primary display object that the application can use for its user interface

Throws:
- NullPointerException - if m is null

getDisplays

public static javax.microedition.lcdui.Display[] getDisplays(int capabilities)

Gets a list of all the Displays for this MIDlet. The returned array will be at least one element long; the first element (index 0) contains a reference to the primary Display which can also be obtained using getDisplay(MIDlet).

The developer can use the capabilities parameter to filter the list of Displays to only those that include specific capabilities. If 0 is used, all Displays are returned regardless of their capabilities. Since the Primary Display must support all capabilities, it is always returned as the first element regardless of the capabilities requested.

The capabilities parameter may include any combination of SUPPORTS_INPUT_EVENTS, SUPPORTS_COMMANDS, SUPPORTS_FORMS, SUPPORTS_TICKER, SUPPORTS_TITLE, SUPPORTS_ALERTS, SUPPORTS_LISTS, SUPPORTS_TEXTBOXES, SUPPORTS_TABBEDPANES, SUPPORTS_FILESELECTORS, SUPPORTS_MENUS, SUPPORTS_IDLEITEM SUPPORTS_ORIENTATION_PORTRAIT, SUPPORTS_ORIENTATION_LANDSCAPE, SUPPORTS_ORIENTATION_PORTRAIT180, SUPPORTS_ORIENTATION_LANDSCAPE180.

The hardware state of the returned Displays should be DISPLAY_HARDWARE_ENABLED or DISPLAY_HARDWARE_DISABLED. Auxiliary Displays that have become absent should not be returned in the list; however, a Display may become absent at any time and so the state of a returned Display object cannot be guaranteed.

Parameters:
- capabilities - a bit field representing the desired Display capabilities

Returns:
An array of Displays containing at least 1 element

Throws:
- IllegalArgumentException - if the capabilities are not valid values (see above)

Since: MIDP 3.0

getDisplayState

public int getDisplayState()

Gets the state of this Display. The state can be STATE_FOREGROUND, STATE_VISIBLE, or STATE_BACKGROUND.

Returns:
the current state of this Display

Since: MIDP 3.0

gdotPitch

public int getDotPitch()
Gets the dot pitch measurement of the Display. This measurement corresponds to the physical spacing of pixels on the display device. Applications can use this information to determine how pixel-based dimensions correspond to physical dimensions on a particular Display. For example, if a Display has a dot pitch of 128 micrometers, then a 100 x 200 pixel Image will be 1.28 cm x 2.56 cm when shown on that Display. The displays on MIDP devices are assumed to have pixels that are approximately square in shape, so the single value applies to both the vertical and horizontal directions. If a display's pixels are not exactly square, the reported dot pitch should be the average of the horizontal and vertical dot pitch values.

**Returns:**
the dot pitch of the Display, in micrometers

**Since:** MIDP 3.0

---

`getExactPlacementPositions`

```java
public int[] getExactPlacementPositions(int border)
```

This will get the possible exact placement positions for the given border on this Display. The method will return an array of available positions for the given border. The border parameter is dependent on the current orientation of the Display. This means that if the display orientation changes, the positions need to be fetched again for the same border id. Each index in the array contains a placement along the border. For horizontal borders (SOFTKEY_BOTTOM, SOFTKEY_TOP) the order of the elements is from left to right. For vertical (SOFTKEY_LEFT, SOFTKEY_RIGHT), the order is from top to bottom. The first possible placement on a border is always 1 plus the border.

Running the code:

```java
int[] pos = getExactPlacementPositions(Display.SOFTKEY_BOTTOM);
```

may return the following:

- `pos[0] = 801`
- `pos[1] = 803`

In this example, the first and third place along the Display.SOFTKEY_BOTTOM are available.

**Parameters:**
- `border` - Any of Display.SOFTKEY_LEFT, Display.SOFTKEY_BOTTOM, Display.SOFTKEY_TOP, Display.SOFTKEY_RIGHT or Display.SOFTKEY_OFFSCREEN

**Returns:**
an array with placement positions or null if none are available.

**Throws:**
`IllegalArgumentException` - if `border` is not a valid border.

**Since:** MIDP 3.0

---

`getHardwareState`

```java
public int getHardwareState()
```

Gets the state of the hardware for this Display. The state can be `DISPLAY_HARDWARE_ENABLED`, `DISPLAY_HARDWARE_DISABLED`, or `DISPLAY_HARDWARE_ABSENT` (applicable to Auxiliary display hardware only).

**Returns:**
the current state of the hardware

**Since:** MIDP 3.0

---

`getHeight`

```java
public int getHeight()
```

**Since:** MIDP 3.0
**getIdleItem**

```java
public javax.microedition.lcdui.IdleItem getIdleItem()
```

Returns the current IdleItem on the Display.

**Returns:**
the IdleItem currently on the display, null if there is no IdleItem set to the display

**Since:** MIDP 3.0

---

**getMenuPreferredPlacements**

```java
public int[] getMenuPreferredPlacements()
```

Returns an array of integers containing the preferred placements normally associated with Menu.

**Returns:**
the preferred placements for Menu on this Display, or null if no preferred placements are associated with Menu.

**See Also:** Menu

**Since:** MIDP 3.0

---

**getMenuSupportedPlacements**

```java
public int[] getMenuSupportedPlacements()
```

Returns an array of integers containing the placements that support Menus.

**Returns:**
the placements that support Menu on this Display, or null if no placements support Menus on this Display.

**See Also:** Menu

**Since:** MIDP 3.0

---

**getOrientation**

```java
public int getOrientation()
```

Gets the current orientation of this Display. The orientation may be ORIENTATION_PORTRAIT, ORIENTATION_LANDSCAPE, ORIENTATION_PORTRAIT_180, or ORIENTATION_LANDSCAPE_180.

**Returns:**
the current orientation

**Since:** MIDP 3.0

---

**getWidth**

```java
public int getWidth()
```

Gets the width in pixels of the Display. The available width reflects the maximum usable size of a Displayable shown on the Display as obtained using a Canvas in full-screen mode having no Commands, Menus, Title, or Ticker. The available width may be smaller than the width of the physical hardware.
Returns:
maximum width of the displayable area

Since: MIDP 3.0

hasPointerEvents

public boolean hasPointerEvents()

Checks if the Display supports pointer press and release events.

Returns:
true if the Display supports pointer events.

Since: MIDP 3.0

hasPointerMotionEvents

public boolean hasPointerMotionEvents()

Checks if the Display supports pointer motion events (pointer dragged). Applications may use this method to determine if this Display is capable of supporting motion events.

Returns:
true if the Display supports pointer motion events.

Since: MIDP 3.0

isBuiltIn

public boolean isBuiltIn()

Checks if this Display's hardware is Built-In or Auxiliary. Built-in display hardware is an integral part of the device and is never absent. Auxiliary hardware may become absent if the connection to it is lost.

Returns:
true if the Display is for Built-In display hardware, false if it is for Auxiliary display hardware

Since: MIDP 3.0

isColor

public boolean isColor()

Gets information about color support of the Display.

Returns:
true if the Display supports color, false otherwise

numAlphaLevels

public int numAlphaLevels()

Gets the number of alpha transparency levels supported by this Display. 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 display.

Note that the number of colors for a black and white display is 2.

Returns:
number of colors

removeCurrent

public void removeCurrent()

Removes the current Displayable from this Display. The Display is moved to the background state if it is not in that state already.

The value returned by getCurrent() is null after removeCurrent has been called.

See Also: setCurrent (Displayable)
Since: MIDP 3.0

removeDisplayListener

public static void removeDisplayListener(DisplayListener l)

Removes a Display listener. The listener will no longer be notified of changes to the Displays.

Parameters:
l - the listener to be removed

Throws:
NullPointerException - if the listener is null
IllegalStateException - if the listener is not currently a listener
Since: MIDP 3.0

setActivityMode

public void setActivityMode(int mode)

Sets the activity mode of this Display. The activity mode of the Display influences how power saving features are applied to the underlying display resource. If the mode is MODE_NORMAL, the device should apply normal power management policies and may take actions such as turning off the backlight or invoking a screensaver to reduce power consumption. If the mode is MODE_ACTIVE, the device must adjust its power management behavior and defer such actions so that regular operation is maintained for as long as possible. If the display device is already in a power-saving state, calling this method with MODE_ACTIVE should cause the display to return to its regular operating state. The current activity mode of the Display is honored only if it has foreground status. To avoid unpredictable behavior, screensaver MIDlets should not use MODE_ACTIVE.

Parameters:
mode - The desired activity mode for this Display (MODE_NORMAL or MODE_ACTIVE)

Throws:
IllegalArgumentException - if the value of mode is not MODE_NORMAL or MODE_ACTIVE
Since: MIDP 3.0

setCommandLayoutPolicy

public void setCommandLayoutPolicy(CommandLayoutPolicy policy)

This method will set a new CommandLayoutPolicy for this Display.
**Parameters:**

- `policy` - The new CommandLayoutPolicy that should be used. If the parameter is null then default policy for the handset is used.

**Since:** MIDP 3.0

```java
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
- `IllegalArgumentException` - if this Display's hardware is absent
- `IllegalStateException` - if the Alert or the Displayable is current on another Display
- `IllegalStateException` - if the Displayable is currently on a tab of a TabbedPane and the TabbedPane is current on another Display
- `DisplayCapabilityException` - if the Displayable is incompatible with this Display

**See Also:** Alert, 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

```java
    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 to place the Display in the background. This implies that a call to `getCurrent()` after a `setCurrent(null)` will return the current `Displayable`, and not `null`. Similarly, if the Display is in the background, passing a non-null reference to `setCurrent()` may be interpreted by the application management software as a request to move the Display 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

```java
    d.setCurrent(d.getCurrent());
```

generally will have no effect other than requesting that the Display 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.

The properties of the `Displayable` with regards to dimensions and capabilities are set at the time `setCurrent()` is called.
**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. The Display will be moved to the foreground state if needed.

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`
- `IllegalStateException` - if the item's container is current on another Display
- `IllegalStateException` - if the item's container is currently on a tab of a `TabbedPane` and the `TabbedPane` is current on another Display
- `DisplayCapabilityException` - if the item's container is incompatible with this Display
- `NullPointerException` - if `item` is null

**Since:** MIDP 2.0

---

**setIdleItem**

```java
public void setIdleItem(IdleItem idleItem)
```

Sets the specified `IdleItem` to the `Display`. Setting new `IdleItem` removes the old one from the `Display`. Passing `null` as the parameter removes the currently set `IdleItem` from the `Display`.

Calling this method does not guarantee that an `IdleItem` will become part of the idle screen for a display. This method merely requests that the system places the `IdleItem` to the display. When the `IdleItem` has been added to the display `IdleItem.addedToDisplay` method is called to inform the `IdleItem` that it is actually part of the idle screen.

**Parameters:**
- `idleItem` - the `IdleItem` to be set to the display

**Throws:**
- `DisplayCapabilityException` - If idle screen access is not supported on this display.
- `IllegalStateException` - If the item is already owned by a Form or `Display`.

**Since:** MIDP 3.0
setPreferredOrientation

public void setPreferredOrientation(int orientation)

Sets the preferred orientation for this Display.

The orientation must one of ORIENTATION_PORTRAIT, ORIENTATION_LANDSCAPE, ORIENTATION_PORTRAIT_180, or ORIENTATION_LANDSCAPE_180 This call may not have an effect on the actual orientation of the display if the specified orientation is not supported on the device. The getCapabilities method can be used to check if the Display support a particular orientation. The application may use getOrientation() to detect the actual orientation of the display after this call.

Parameters:
orientation - the new preferred orientation for this Display

Throws:
IllegalArgumentException - if the orientation value is invalid

See Also: getOrientation()
Since: MIDP 3.0

vibrate

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.

Exact placement of Commands

In MIDP3.0 the method of using type and priority for Commands placement is supplemented with a method to use absolute placement using the optional placement value. The available placements are retrieved from Display.getExactPlacements. A placement is encoded as a base value for each border plus an index of the soft key along the border. For more details see Display.

Since: MIDP 3.0

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void addCommand(Command cmd)</td>
<td>Adds a command to the Displayable.</td>
</tr>
<tr>
<td>javax.microedition.lcdui.Command getCommand(int placement)</td>
<td>Gets the Command at the given placement.</td>
</tr>
<tr>
<td>javax.microedition.lcdui.CommandLayoutPolicy getCommandLayoutPolicy()</td>
<td>This method will return the current CommandLayoutPolicy for this Displayable.</td>
</tr>
</tbody>
</table>
### javax.microedition.lcdui.Command[]

**getCommands()**

*Gets the set of commands that has been added to the Displayable.*

### javax.microedition.lcdui.Display

**getCurrentDisplay()**

*Returns the Display associated with this Displayable.*

### abstract int

**getHeight()**

*Gets the height of the Displayable, in pixels.*

### javax.microedition.lcdui.Menu

**getMenu(int placement)**

*Gets the Menu at the given placement.*

### javax.microedition.lcdui.Ticker

**getTicker()**

*Gets the ticker used by this Displayable.*

### java.lang.String

**getTitle()**

*Gets the title of the Displayable.*

### abstract int

**getWidth()**

*Gets the width of the Displayable, in pixels.*

### void

**invalidateCommandLayout()**

*Request an update of the layout policy.*

### boolean

**isShown()**

*Checks if the Displayable is visible on the display.*

### void

**removeCommand(Command cmd)**

*Removes a command from the Displayable.*

**removeCommandOrMenu(int placement)**

*Removes the Command or Menu present at the given exact placement.*

**setCommand(Command cmd, int placement)**

*Adds or sets a Command to the Displayable at the given placement.*

**setCommandLayoutPolicy(CommandLayoutPolicy policy)**

*This method will set a new CommandLayoutPolicy for this Displayable.*

**setCommandListener(CommandListener l)**

*Sets a listener for Commands to this Displayable, replacing any previous CommandListener.*

**setMenu(Menu menu, int placement)**

*Adds or sets a menu to the Displayable at the given placement.*

**setTicker(Ticker ticker)**

*Sets a ticker for use with this Displayable, replacing any previous ticker.*

**setTitle(String s)**

*Sets the title of the Displayable.*

**sizeChanged(int w, int h)**

*The implementation calls this method when the available area of the Displayable has been changed.*

### Methods inherited from class object

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

addCommand

```java
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 on 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
- `DisplayCapabilityException` - if the Displayable is set to a Display that does not support Commands

getCommand

```java
public javax.microedition.lcdui.Command getCommand(int placement)
```

Gets the Command at the given placement. If the placement is empty or contains a Menu, null is returned.

**Parameters:**
- `placement` - the placement to query

**Returns:**
- Command instance located at the given placement, or null if no Command exists at the given placement.

**Throws:**
- `IllegalArgumentException` - if the provided placement is not valid.

**Since:** MIDP 3.0

getCommandLayoutPolicy

```java
public javax.microedition.lcdui.CommandLayoutPolicy getCommandLayoutPolicy()
```

This method will return the current CommandLayoutPolicy for this Displayable.

**Returns:**
- The current CommandLayoutPolicy

**Since:** MIDP 3.0

getCommands

```java
public javax.microedition.lcdui.Command[] getCommands()
```

Gets the set of commands that has been added to the Displayable.

**Returns:**
- An array containing all commands added to the displayable

**Since:** MIDP 3.0

getCurrentDisplay

```java
public javax.microedition.lcdui.Display getCurrentDisplay()
```
javax.microedition.lcdui.Displayable

Returns the Display associated with this Displayable. Returns null if this Displayable is not bound to any Display.

Returns:
Display associated with this Displayable. null if this Displayable is not bound to a Display.

Since: MIDP 3.0

**getHeight**

public abstract int getHeight()

Gets the height of the Displayable, in pixels. The returned value indicates the height of the Displayable's content area that is available in its entirety for the developer to use. The content area specifically excludes any portions of the screen that are not directly useable by the developer such as areas occupied by status bars, commands, tickers, titles, and other on-screen artifacts (e.g. an on-screen keypad to support game actions on touch-screen devices). The returned value reflects the size and layout of the default Display if the Displayable is not currently shown on a Display.

Returns:
height of the area available to the application

Since: MIDP 2.0

**getMenu**

public javax.microedition.lcdui.Menu getMenu(int placement)

Gets the Menu at the given placement. If the placement is empty or contains a Command, null is returned.

Parameters:
placement - the placement to query

Returns:
Menu instance located at the given placement, or null if no Menu exists at the given placement.

Throws:
IllegalArgumentException - if the provided placement is not valid.

Since: MIDP 3.0

**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: 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: setTitle(String)
getWidth
public abstract int getWidth()

Gets the width of the Displayable, in pixels. The returned value indicates the width of the Displayable's content area that is available in its entirety for the developer to use. The content area specifically excludes any portions of the screen that are not directly useable by the developer such as areas occupied by status bars, commands, tickers, titles, and other on-screen artifacts (e.g. an on-screen keypad to support game actions on touch-screen devices). The returned value reflects the size and layout of the default Display if the Displayable is not currently shown on a Display.

Returns:
width of the area available to the application

invalidateCommandLayout
public void invalidateCommandLayout()

Request an update of the layout policy. If this displayable is not current on its display, or doesn't have focus the request will be ignored.

isShown
public boolean isShown()

Checks if the Displayable is visible on the display. A Displayable is considered to be visible if at least one of its pixels is visible to the user. In order for a Displayable to be visible, all of the following must be true: the Display must not be in the background, the Displayable must be the Display's current screen, and at least one pixel of the screen's display resource must be allocated to the Display.

Returns:
true if the Displayable is currently at all visible

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

removeCommandOrMenu
public void removeCommandOrMenu(int placement)

Removes the Command or Menu present at the given exact placement. This method should only be called from inside an implementation of the CommandLayoutPolicy interface.

Parameters:
placement - the placement of the Command or Menu to be removed

Throws:
IllegalArgumentException - if the provided placement is not valid
removeClass

public void removeClass(String class)

Removes the specified class from the set of classes that are currently being used to represent this object.

Parameters:

class - the class to be removed

Throws:

IllegalArgumentException - if class is null

Since: MIDP 3.0

removeClassLayoutPolicy

public void removeClassLayoutPolicy(ClassLayoutPolicy policy)

This method will remove the specified ClassLayoutPolicy for this Displayable.

Parameters:

policy - the ClassLayoutPolicy to be removed

Since: MIDP 3.0

removeClassListener

public void removeClassListener(ClassListener l)

Removes the specified ClassListener from the list of listeners for this Displayable.

Parameters:

l - the ClassListener to be removed

Since: MIDP 3.0

removeClasses

public void removeClasses()

Removes all classes from this object.

Since: MIDP 3.0

setCommand

public void setCommand(Command cmd, int placement)

Adds or sets a Command to the Displayable at the given placement. If the Command already exists on another placement of this Displayable, it will be moved to the new placement. Just before the Command is added, any Menu or Command already occupying the given placement will be removed from the associated Item or Displayable. This method should only be called from inside an implementation of the CommandLayoutPolicy interface. 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
placement - the placement of the Command to be added

Throws:

NullPointerException - if cmd is null
DisplayCapabilityException - if the Displayable is set to a Display that does not support Commands
IllegalArgumentException - if the provided placement is not valid
IllegalStateException - If this method is not called from a CommandLayoutPolicy.onCommandLayout() implementation.
IllegalStateException - If this Displayable is neither the same instance passed into the CommandLayoutPolicy.onCommandLayout() method nor the current tab of a TabbedPane

Since: MIDP 3.0

setCommandLayoutPolicy

public void setCommandLayoutPolicy(CommandLayoutPolicy policy)

This method will set a new CommandLayoutPolicy for this Displayable.

Parameters:

policy - The new CommandLayoutPolicy that should be used. If the parameter is null then default policy for the Display is used.

Since: MIDP 3.0

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

setMenu

public void setMenu(Menu menu, int placement)

Adds or sets a menu to the Displayable at the given placement. If the menu already exists on another placement of this Displayable, it will be moved to the new placement. Just before the Menu is added, any Menu or Command already occupying the given placement will be removed from the associated Item or Displayable. This method should only be called from inside an implementation of the CommandLayoutPolicy interface. 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:

menu - the Menu to be added
placement - the placement of the Menu to be added

Throws:

NullPointerException - if menu is null
DisplayCapabilityException - if the Displayable is set to a Display that does not support Menus
IllegalArgumentException - if the provided placement is not valid
IllegalStateException - If this method is not called from a CommandLayoutPolicy.onCommandLayout() implementation.
IllegalStateException - If this Displayable is neither the same instance passed into the CommandLayoutPolicy.onCommandLayout() method nor the current tab of a TabbedPane

Since: MIDP 3.0
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

Throws:

DisplayCapabilityException - if the Displayable is set to a Display that does not support Tickers

See Also: 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

Throws:

DisplayCapabilityException - if the Displayable is set to a Display that does not support Titles

See Also: getTitle()

Since: MIDP 2.0
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 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
DisplayCapabilityException

Declaration

public class DisplayCapabilityException extends RuntimeException

Object
\|-- Throwable
   \|-- Exception
      \|-- RuntimeException
         \|-- javax.microedition.lcdui.DisplayCapabilityException

Description

Indicates that a Display’s capabilities are insufficient for the requested operation.

This exception may be thrown in two types of calls. First, it is thrown by the Display on a setCurrent() if
the Displayable is incompatible with the Display’s capabilities. Second, it is thrown by a Displayable’s
methods if the relevant feature is not supported by the Displayable’s current Display.

Since: MIDP 3.0
See Also: Displayable, Display

Constructor Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>DisplayCapabilityException()</td>
<td>Constructs an exception with no specified detail message.</td>
</tr>
<tr>
<td>public</td>
<td>DisplayCapabilityException(String s)</td>
<td>Constructs an exception with the specified detail message.</td>
</tr>
</tbody>
</table>

Methods inherited from class Throwable

getMessage, printStackTrace, printStackTrace, toString

Methods inherited from class Object

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

Constructors

DisplayCapabilityException

public DisplayCapabilityException()  Constructs an exception with no specified detail message.

DisplayCapabilityException

public DisplayCapabilityException(String s)
javax.microedition.lcdui.DisplayCapabilityException

Constructs an exception with the specified detail message.

**Parameters:**
- `s` - the detail message
javax.microedition.lcdui

DisplayListener

Declaration

public interface DisplayListener

Description

The DisplayListener interface defines a series of methods that are called in response to Display events. Examples of Display events include the addition of new Displays and changes to the state of an existing Display. A DisplayListener may be added using the Display.addDisplayListener method.

Since: MIDP 3.0

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>displayAdded(Display d)</td>
<td>This method is called when a new Display becomes available for the MIDlet.</td>
</tr>
<tr>
<td>void</td>
<td>displayStateChanged(Display d, int newState)</td>
<td>This method is called when the state of a Display changes.</td>
</tr>
<tr>
<td>void</td>
<td>hardwareStateChanged(Display d, int newState)</td>
<td>This method is called when the state of a Display's hardware changes.</td>
</tr>
<tr>
<td>void</td>
<td>orientationChanged(Display d, int newOrientation)</td>
<td>This method is called when a Display's orientation changes.</td>
</tr>
<tr>
<td>void</td>
<td>sizeChanged(Display d, int w, int h)</td>
<td>The implementation calls this method when the dimensions of a Display have been changed, i.e.</td>
</tr>
</tbody>
</table>

Methods

displayAdded

public void displayAdded(Display d)

This method is called when a new Display becomes available for the MIDlet. A new Display is added when the device establishes a connection with new auxiliary display hardware. The hardwareStateChanged method can be used to detect subsequent changes to the state of the Display's hardware.

Parameters:
  d - the new Display that was added

displayStateChanged

public void displayStateChanged(Display d, int newState)

This method is called when the state of a Display changes.

Parameters:
  d - the Display whose state changed
javax.microedition.lcdui.DisplayListener

newState - the new state of the Display (STATE_BACKGROUND, STATE_FOREGROUND, or STATE_VISIBLE)

**hardwareStateChanged**

public void **hardwareStateChanged**(Display d, int newState)

This method is called when the state of a Display's hardware changes.

**Parameters:**
- d - the Display
- newState - the new hardware state of the Display (Display.DISPLAY_HARDWARE_ENABLED, Display.DISPLAY_HARDWARE_DISABLED, or Display.DISPLAY_HARDWARE_ABSENT)

**orientationChanged**

public void **orientationChanged**(Display d, int newOrientation)

This method is called when a Display's orientation changes. Such a change may be caused by a change to the MIDlet's preferred orientation or by the user via a change to the device's settings. If currently shown on the Display, the Displayable's sizeChanged method may also be called in response to an orientation change.

**Parameters:**
- d - the Display
- newOrientation - the new orientation of the Display (Display.ORIENTATION_PORTRAIT, Display.ORIENTATION_LANDSCAPE, Display.ORIENTATION_PORTRAIT_180, or Display.ORIENTATION_LANDSCAPE_180)

**sizeChanged**

public void **sizeChanged**(Display d, int w, int h)

The implementation calls this method when the dimensions of a Display have been changed, i.e. as a result of a change in screen orientation or resizing of a virtual display such as a window.

If the size of a Display changes while its contents are visible, sizeChanged will be called. If the size of a Display changes while its contents are not visible, calls to sizeChanged may be deferred. If the size had changed while the Display contents were not visible, sizeChanged will be called at least once at the time the contents of the Display become visible once again.

**Parameters:**
- d - the Display
- w - the new width in pixels of the available area
- h - the new height in pixels of the available area

**Since:** MIDP 3.0
javax.microedition.lcdui

FileSelector

**Declaration**

```java
public class FileSelector extends Screen
```

**Object**

```
+--javax.microedition.lcdui.Displayable
   +--javax.microedition.lcdui.Screen
      +--javax.microedition.lcdui.FileSelector
```

**Description**

The `FileSelector` class allows the user to select a file from file system to be loaded, saved, or to select a directory. The FileSelector maintains the confidentiality of the filesystem by revealing only the URL of the file or directory explicitly selected by the user. The application only receives information and access to the file permitted by the user. The directory and file names passed to and returned by the FileSelector are formatted as fully qualified absolute file URLs [RFC3986].

FileSelector has two fixed, built-in Commands: **OK_COMMAND** and **CANCEL_COMMAND**. It is not possible to add any other commands or menus to or remove commands or menus from FileSelector.

When the user selects a file or directory with a FileSelector and pressed OK, the CommandListener is called with an **OK_COMMAND** event. Similarly, if the user dismisses the FileSelector by selecting Cancel, the command listener is called with a **CANCEL_COMMAND** event. These two operations, OK and Cancel, MUST be accessible to the user in any FileSelector. If the screen is dismissed without any user action, the listener MUST NOT be called.

When the user selects OK, the `getURL` method MUST return the URL of the user selected file or directory depending on the mode. The application can call `Connector.open` with this URL and access will be subject to the security model and policy. The application can also call the `FileSelector.open` method and access the content via the `Connection` it returns. Since the user has explicitly given the MIDlet access to the file, no additional security prompts are required. Implementations MAY combine security prompts for access to the file with the FileSelector. In **LOAD** and **SAVE** mode, the `getURL` method MUST return **null** unless the user has selected the file and pressed OK; whereas in **DIRECTORY_SELECT** and **DIRECTORY_CREATE** mode, the `getURL` method MUST return **null** unless the user has selected any directory and pressed OK. After selecting Cancel the `getURL` method must return **null**. It is up to the application to change the current displayable in response to the OK or Cancel command.

**FileChooser** has four modes, **LOAD**, **SAVE**, **DIRECTORY_SELECT**, and **DIRECTORY_CREATE**. The mode affects the behavior of the dialog:

- In the **LOAD** mode, the implementation must ensure that a file returned from FileSelector exists.
- In the **SAVE** mode the user MUST be able to select an existing file or create a new file name.
- In the **DIRECTORY_SELECT** mode the user must be able to select an existing directory.
- In the **DIRECTORY_CREATE** mode the user must be able to create a new named directory.

If the implementation allows the user to type a file name or directory name and the requested action is not successful, then the implementation MUST show an implementation specific warning with the appropriate reason. Reasons may include, but are not limited, to the following:

- The file or directory does not exist in **LOAD** and **DIRECTORY_SELECT** mode
- The application does not have permission to create file or directory in the specified directory
- The application is not allowed to access the file or directory
The FileSelector's CommandListener MUST not be called if the requested action is not completed successfully.

Example Using FileSelector

This example shows the use of the FileSelector to select an image based on the file suffix. The open method is used to get a StreamConnection to the contents and then to create an Image from the stream. The Image is added to an Alert asking the user to confirm the image. This part of the example can be used on any implementation of this specification that has a file system.

The second part of the example, to delete the image, requires the JSR-75 FileConnection API to be implemented by the device. When the delete command is selected then the FileConnection.delete method is called.
public class FileSelectDelete implements CommandListener {

    Display display;
    String title;
    FileSelector fs;
    Displayable oldCurrent;
    String[] filterStrings = {"png", "gif", "jpeg"};

    Command DELETE_COMMAND = new Command("Delete", Command.SCREEN, 2);
    Command CANCEL_COMMAND = new Command("Cancel", Command.SCREEN, 1);

    /**
     * Delete a user selected file; with a confirmation.
     * /
    void deleteImage(Display display, String title) {
        this.display = display;
        this.title = title;
        oldCurrent = display.getCurrent();
        fs = new FileSelector(title, FileSelector.SAVE);
        fs.setFilterExtensions(filterStrings);
        fs.setCommandListener(this);
        display.setCurrent(fs);
    }

    /**
     * Handle the OK or Cancel command from the user.
     * In all cases, an Alert is displayed and when dismissed,
     * the previous displayable is restored.
     * The errors for incorrect file path or type put up alerts.
     * If it was ok, then try to display the selected file as an image
     * and provide commands to Delete and cancel the delete.
     * Use the FileConnection to delete the file.
     * *
     * @param c The command selected by the user
     * @param d the Displayable containing the command (FileSelector)
     * /
    public void commandAction(Command c, Displayable d) {
        Alert alert = null;
        if (c == FileSelector.OK_COMMAND) {
            // Invoked when user selects OK from FileSelector
            StreamConnection conn = null;
            try {
                conn = (StreamConnection) fs.open(Connector.READ, false);
                Image image = Image.createImage(conn.openInputStream());
                alert = new Alert(title, "Delete Image?", image,
                    AlertType.WARNING);
                alert.addCommand(DELETE_COMMAND);
                alert.addCommand(CANCEL_COMMAND);
            } catch (IOException ioe1) {
                alert = new Alert("Image Delete",
                    AlertType.ERROR);
                alert.addCommand(DELETE_COMMAND);
            } catch (IOException ioe2) {
                alert = new Alert("Image Delete",
                    AlertType.ERROR);
                alert.addCommand(DELETE_COMMAND);
            }
        } else if (c == FileSelector.CANCEL_COMMAND) {
            display.setCurrent(oldCurrent);
        } else {
            alert = new Alert("File Select Delete",
                AlertType.ERROR);
            alert.addCommand(DELETE_COMMAND);
            alert.addCommand(CANCEL_COMMAND);
        }
        alert.show();
    }
}
"Selected file cannot be displayed as an image.",
null, AlertType.ERROR);
} finally {
    try {if (conn != null) conn.close();} catch (IOException ioe2) {}}  
}  
else if (c == FileSelector.CANCEL_COMMAND) {
    // Invoked when the user cancels a file selection
    alert = new Alert(title, "Image selection canceled.", null,
    AlertType.INFO);
} else if (c == CANCEL_COMMAND) {
    // Invoked when the user has cancelled the delete of the image
    alert = new Alert(title, "Image delete canceled.", null,
    AlertType.INFO);
} else if (c == DELETE_COMMAND) {
    // Invoked when the user has confirmed the delete of the image
    String fileurl = fs.getURL();
    FileConnection fc = null;
    try {
        fc = (FileConnection)Connector.open(fileurl, Connector.WRITE,
false);
        fc.delete();
        alert = new Alert(title,
            "Image delete confirmed.", null, AlertType.CONFIRMATION);
    } catch (IOException ioe) {
        alert = new Alert(title,
            "Unable to delete image.", null, AlertType.ERROR);
    } finally {
        try {if (fc != null) fc.close();} catch (IOException ioe2) {}}
    
    alert.setCommandListener(this);
    display.setCurrent(alert, oldCurrent);
}

Since: MIDP 3.0

<table>
<thead>
<tr>
<th>Field Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td>A Command delivered to a listener to indicate that user has dismissed the FileSelector without selecting any file.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td>This constant value indicates that the purpose of the file selector is to choose the name for a directory the application will create.</td>
</tr>
<tr>
<td>Value: 3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>public static final</strong></td>
</tr>
<tr>
<td>This constant value indicates that the purpose of the file selector is to locate a directory.</td>
</tr>
<tr>
<td>Value: 2</td>
</tr>
<tr>
<td>Method Description</td>
</tr>
<tr>
<td>--------------------</td>
</tr>
<tr>
<td>LOAD</td>
</tr>
<tr>
<td>OK_COMMAND</td>
</tr>
<tr>
<td>SAVE</td>
</tr>
</tbody>
</table>

**Constructor Summary**

```
public FileSelector(String title, int mode)
```

Creates a FileSelector with the specified title and mode.

**Method Summary**

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

FileSelector has two fixed, built-in Commands: OK_COMMAND and CANCEL_COMMAND.

```
public java.lang.String[] getFilterExtensions()
```

Gets the file extensions set for this file selector.

```
public int getHeight()
```

Gets the height in pixels of the displayable area in FileSelector that is used to render file or directory information without scrolling.

```
public int getMode()
```

Sets whether this file selector is for loading from a file, for saving to a file, or for finding a directory.

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

Gets the URL of the user selected directory and file.

```
public int getWidth()
```

Gets the width in pixels of the displayable area in FileSelector that is used to render file or directory information without scrolling.

```
public javax.microedition.io.StreamConnection open(int mode, boolean timeouts)
```

Creates and opens a StreamConnection to the URL in getURL.

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

FileSelector does not allow Commands to be removed.

```
public void removeCommandOrMenu(int placement)
```

FileSelector does not allow Commands or Menus to be removed.

```
public void setCommand(Command cmd, int placement)
```

FileSelector has two fixed, built-in Commands: OK_COMMAND and CANCEL_COMMAND.

```
public void setCommandLayoutPolicy(CommandLayoutPolicy policy)
```

FileSelector does not allow setCommandLayoutPolicy() to be called.
### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void setFilterExtensions(String[] extensions)</td>
<td>Sets the file extensions for this file selector to show only the files with the set extensions.</td>
</tr>
<tr>
<td>void setMenu(Menu menu, int placement)</td>
<td>FileSelector does not allow menus to be added.</td>
</tr>
<tr>
<td>void setMode(int mode)</td>
<td>Sets the mode of the file selector to one of LOAD, SAVE, DIRECTORY_SELECT or DIRECTORY_CREATE.</td>
</tr>
<tr>
<td>void setURL(String URL)</td>
<td>Sets the selected directory and file for this file selector according to the given URL.</td>
</tr>
</tbody>
</table>

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

- getCommand
- getCommandLayoutPolicy
- getCommands
- getCurrentDisplay
- getMenu
- getTicker
- getTitle
- invalidateCommandLayout
- isShown
- setCommandListener
- setTicker
- setTitle
- sizeChanged

Methods inherited from class `Object`

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

### Fields

**CANCEL_COMMAND**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final javax.microedition.lcdui.Command CANCEL_COMMAND</td>
<td>A Command delivered to a listener to indicate that user has dismissed the FileSelector without selecting any file. The field values of CANCEL_COMMAND are as follows:</td>
</tr>
</tbody>
</table>

- label = "" (an empty string)
- type = Command.CANCEL
- priority = 1

The implementation displays CANCEL_COMMAND to the user using an implementation-specific label. Setting the long or short labels MUST have no effect on the label presented to the user.

**DIRECTORY_CREATE**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final int DIRECTORY_CREATE</td>
<td>This constant value indicates that the purpose of the file selector is to choose the name for a directory the application will create.</td>
</tr>
</tbody>
</table>

Value 3 is assigned to DIRECTORY. Constant value: 3

**DIRECTORY_SELECT**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final int DIRECTORY_SELECT</td>
<td></td>
</tr>
</tbody>
</table>
This constant value indicates that the purpose of the file selector is to locate a directory.

Value 2 is assigned to DIRECTORY.
Constant value: 2

---

**LOAD**

```java
public static final int LOAD
```

This constant value indicates that the purpose of the file selector is to locate a file from which to read.

Value 0 is assigned to LOAD.
Constant value: 0

---

**OKCOMMAND**

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

The OK Command delivered to a command listener indicates that a file has been selected from FileSelector. The field values of OK_COMMAND are as follows:

- label = "" (the empty string)
- type = Command.OK
- priority = 0

The implementation displays OK_COMMAND to the user using an implementation-specific label. Setting the long or short labels MUST have no effect on the label presented to the user.

---

**SAVE**

```java
public static final int SAVE
```

This constant value indicates that the purpose of the file selector is to locate a file to which to write. The file selector must also allow the user to create a new file name.

Value 1 is assigned to SAVE.
Constant value: 1

---

**Constructors**

**FileChooser**

```java
public FileSelector(String title,
                    int mode)
```

Creates a FileSelector with the specified title and mode. If the value of mode is LOAD, then the file selector is for finding a file to be read. If the value of mode is SAVE, the file selector is for finding a file to write. If the value of mode is DIRECTORY_SELECT, then the file selector is for finding a directory. If the value of the mode is DIRECTORY_CREATE, then the file selector is for creating new directories.

**Parameters:**
- title - The title of the dialog, or null if no title
- mode - The mode of the dialog

**Throws:**
- UnsupportedOperationException - if the underlying system does not have a file system to support the use of this class
- IllegalArgumentException - if the mode is other than LOAD, SAVE, DIRECTORY_SELECT or DIRECTORY_CREATE.
See Also: LOAD, SAVE, DIRECTORY_SELECT, DIRECTORY_CREATE

Methods

**addCommand**

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

FileSelector has two fixed, built-in Commands: OK_COMMAND and CANCEL_COMMAND. New Commands are not allowed on FileSelector, so this method will always throw IllegalStateException whenever it is called.

**Parameters:**
- `cmd` - the Command is ignored

**Throws:**
- `java.lang.IllegalStateException` - always

**getFilterExtensions**

```java
public java.lang.String[] getFilterExtensions()
```

Gets the file extensions set for this file selector. The extension strings are file system specific: for example "txt" or "xml". The extension strings are file system specific: for example "txt" or "xml".

**Returns:**
- the file extensions for this file selector; if no filters are in effect, an empty String array MUST be returned.

**getHeight**

```java
public int getHeight()
```

Gets the height in pixels of the displayable area in FileSelector that is used to render file or directory information without scrolling. This displayable area only includes the area which displays the file or directory information. This area excludes any Title, Ticker, scrollbar, or commands that do not actually render the file or directory information.

**Returns:**
- height in pixels of the area available to display the given URL information

**getMode**

```java
public int getMode()
```

Sets whether this file selector is for loading from a file, for saving to a file, or for finding a directory.

**Returns:**
- the mode of this FileSelector, one of: FileSelector.LOAD, FileSelector.SAVE, FileSelector.DIRECTORY_SELECT of FileSelector.DIRECTORY_CREATE.

**See Also:** LOAD, SAVE, DIRECTORY_SELECT, DIRECTORY_CREATE, setMode(int)

**getURL**

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

Gets the URL of the user selected directory and file. If the user selected CANCEL_COMMAND, null is returned. The URLs passed to and returned by the FileSelector are formatted as fully qualified absolute file URLs [RFC3986].

**Returns:**
Since: MIDP 3.0

getWidth

public int getWidth()

Gets the width in pixels of the displayable area in FileSelector that is used to render file or directory information without scrolling. This displayable area only includes the area which displays the file or directory information. This area excludes any Title, Ticker, scrollbar, or commands that do not actually render the file or directory information.

Returns: width in pixels of the area available to display the given URL information

open

public javax.microedition.io.StreamConnection open(int mode, boolean timeouts)
throws javax.microedition.io.ConnectionNotFoundException, java.io.IOException

Creates and opens a StreamConnection to the URL in getURL. This method is similar to Connector.open(getURL(), mode, timeouts). The application should use this method to access the content of the URL to avoid redundant access checks.

If JSR-75 is present, the StreamConnection returned SHOULD implement the javax.microedition.file.FileConnection. Applications can use the instanceof operator to determine if a FileConnection was returned.

Parameters:
mode - the access mode
timeouts - a flag to indicate that the caller wants timeout exceptions

Returns:
a StreamConnection supporting the requested mode.

Throws:
ConnectionNotFoundException - If the target of the name cannot be found, or if the requested protocol type is not supported.
IOException - If some other kind of I/O error occurs.
SecurityException - If access to the content is prohibited.

removeCommand

public void removeCommand(Command cmd)

FileChooser does not allow Commands to be removed.

Parameters:
cmd - the Command is ignored

Throws:
java.lang.IllegalStateException - always

removeCommandOrMenu

public void removeCommandOrMenu(int placement)

FileChooser does not allow Commands or Menus to be removed.
setCommand

public void setCommand(Command cmd, int placement)

FileSelector has two fixed, built-in Commands: OK_COMMAND and CANCEL_COMMAND. New Commands are not allowed on FileSelector, so this method will always throw IllegalStateException whenever it is called.

Parameters:
    cmd - the Command is ignored
    placement - the placement is ignored

Throws:
    java.lang.IllegalStateException - always

setCommandLayoutPolicy

public void setCommandLayoutPolicy(CommandLayoutPolicy policy)

FileSelector does not allow setCommandLayoutPolicy() to be called. Applications are not allowed to add/set Commands or Menus in FileSelector

Parameters:
    policy - the CommandLayoutPolicy is ignored

Throws:
    IllegalStateException - always

Since: MIDP 3.0

setFilterExtensions

public void setFilterExtensions(String[] extensions)

Sets the file extensions for this file selector to show only the files with the set extensions. The extension strings are file system specific. For example, "txt" or "xml".

Parameters:
    extensions - the file extensions being set; if null or an empty array then no filtering will be done.

setMenu

public void setMenu(Menu menu, int placement)

FileSelector does not allow menus to be added.

Parameters:
    menu - the Menu ignored
    placement - the placement is ignored

Throws:
    java.lang.IllegalStateException - always
setMode

public void setMode(int mode)

Sets the mode of the file selector to one of LOAD, SAVE, DIRECTORY_SELECT or DIRECTORY_CREATE.

Parameters:

mode - the mode for this file selector, one of FileDialog.LOAD, FileSelector.SAVE, FileSelector.DIRECTORY_SELECT or FileSelector.DIRECTORY_CREATE.

Throws:

java.lang.IllegalArgumentException - if an illegal file selector mode is used

See Also: LOAD, SAVE, DIRECTORY_SELECT, DIRECTORY_CREATE, getMode()

setURL

public void setURL(String URL)
throws java.io.IOException

Sets the selected directory and file for this file selector according to the given URL. This file becomes the default file, if it is set before the file selector is first set to visible. Specifying a hidden file maps to the behavior of the filesystem and native platform; for example if hidden files are shown because of a system wide user preference, then they are shown via FileSelector as well.

If supported by the filesystem, specifying a file that does not exist will result in creation of a new file by that name.

The directory and file names passed to and returned by the FileSelector are formatted as fully qualified absolute file URLs [RFC3986].

Parameters:

URL - the file being set; if null, then the default directory is the URL of the MIDlet suite's private storage directory.

Throws:

IllegalArgumentException - if the URL is not a valid absolute file URL or is empty.
IOException - if the mode is LOAD and the URL refers to a file that does not exist.
IOException - if the mode is SAVE and the URL refers to a directory that does not exist or in which the application is not permitted to create files or refers to a file which exists as a directory.
IOException - if the mode is DIRECTORY_SELECT and the URL refers to a directory that does not exist.
IOException - if the mode is DIRECTORY_CREATE and the URL refers to a directory that does not exist or in which the application is not permitted to create directories or refers to a directory which exists as a file.
SecurityException - if the URL references a directory or file that the MIDlet is not allowed to access.
Font

Declaration

public final class Font

Object

Description

The Font class represents fonts, which are used to render text in a visible way. A font provides the information needed to map sequences of characters to sequences of glyphs and to render sequences of glyphs on Graphics and Component objects.

Characters and Glyphs

A character is a symbol that represents a unit of text content - an item such as a letter, a digit, or punctuation represented in an abstract way by its character code. For example, 'g', LATIN SMALL LETTER G, is a character.

A glyph is a shape that is used to render a character, and represents a unit of text display. Each character may be represented using one of many possible glyphs that differ in appearance and style. Collections of glyphs designed in the same style are called typefaces, and are represented by different fonts.

A glyph may represent a single character or a sequence of characters. In simple writing systems, such as Latin, typically one glyph represents one character. In general, however, characters and glyphs do not have one-to-one correspondence. For example, the character 'á' LATIN SMALL LETTER A WITH ACUTE, can be represented by two glyphs: one for 'a' and one for '. On the other hand, the two-character string "fi" can be represented by a single glyph, a "fi" ligature. In complex writing systems, such as Arabic or the South and South-East Asian writing systems, the relationship between characters and glyphs can be more complicated and involve context-dependent selection of glyphs as well as glyph reordering. A font encapsulates the collection of glyphs needed to render a selected set of characters as well as the tables needed to map sequences of characters to corresponding sequences of glyphs.

Each glyph has a set of metrics that summarize the geometry of the glyph. The methods to get the metrics are: getHeight, getAscent, getMaxAscent, getDescent, getMaxDescent, and getBaselinePosition, getLeading. The metrics are illustrated below:

Physical and Logical Fonts

MIDP distinguishes between two kinds of fonts: physical fonts and logical fonts.

Physical fonts are the actual font files containing glyph data and tables to map from character sequences to glyph sequences, using a font technology such as OpenType, TrueType or PostScript Type 1. All implementations MUST support OpenType fonts with TrueType outlines. Implementations SHOULD support TrueType hinting and MAY support advanced typographic functions. Support for other font technologies is optional and implementation dependent. Physical fonts use distinct (and, sometimes, trademarked) names such as Helvetica, Palatino, HonMincho, or any other font names. Typically, each physical font supports only a limited character set, for example, only Latin characters or only Japanese and Basic Latin. The set of available physical fonts varies between configurations. Applications that require specific fonts can bundle them and instantiate them using the getFont or createFont methods.

Logical fonts are the five font families defined by the Java platform which must be supported by any Java runtime environment: Serif, SansSerif, Monospaced, Dialog, and DialogInput. These logical fonts are not actual font libraries. Instead, the logical font names are mapped to physical fonts by the Java runtime.
environment. The mapping is implementation- and (usually) locale-dependent, so the look and the metrics provided by them vary. Typically, each logical font name maps to several physical fonts in order to cover a large range of characters.

**Font Faces and Names**

A font (typeface) can have many styles and/or weights, such as heavy, medium, oblique, gothic and regular. All of these faces have similar typographic design.

There are three different names that application can get from a `Font` object. The *logical font name* is simply the name that was used to construct the font. The *font face name*, or just *font name* for short, is the name of a particular font that specifies its family and style, like "Helvetica Bold". The *family name* is the name of the font family that determines the typographic design across several faces, like "Helvetica".

The `Font` class represents an instance of a font face from a collection of font faces that are present in the system resources of the host system. As examples, Arial Bold and Courier Bold Italic are font faces. There can be several `Font` objects associated with a font face, each differing in size, style and font features.

A `Font`'s attributes are style, size (or `pixelSize`) and font name (or `fontSpecifier`). Values for *size* attribute may be specified as either integer or symbolic constant; values for *pixelSize* shall always be specified as positive, non-zero integer. Values for *style* attribute 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 symbolic 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.

Font Size

Most users are familiar with the idea of using point size to specify the size of glyphs in a font, which is based on typographic points, approximately 1/72 of an inch. The use of point size allows creating text documents that preserve the absolute size of the text, regardless of whether a document is reproduced in print or displayed on a computer screen. However, in some circumstances, a relative size of different objects of multimedia content (e.g. graphics, images and text) needs to be preserved in order to assure the content layout and appearance, especially if scalable media content is intended to be displayed on screens with different sizes and resolutions.

Mobile devices have a wide variety of display sizes and resolutions. MIDlets can query the display size and resolution of a particular device, and can calculate the required size of graphics, text and image objects. For this reason, the size of MIDP Font objects is always calculated and defined as device-specific, in pixels. Implementations SHOULD map the values of symbolic constants SIZE_LARGE, SIZE_MEDIUM and SIZE_SMALL to actual pixel sizes, based on the primary device screen size and resolution. The default size of a new Font object (when it's not specified) is SIZE_MEDIUM.

Since: MIDP 1.0

<table>
<thead>
<tr>
<th>Field Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>The &quot;monospace&quot; font face.</td>
</tr>
<tr>
<td>Value: 32</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>The &quot;proportional&quot; font face.</td>
</tr>
<tr>
<td>Value: 64</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>The &quot;system&quot; font face.</td>
</tr>
<tr>
<td>Value: 0</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>Font specifier for focused text on the idle screen.</td>
</tr>
<tr>
<td>Value: 3</td>
</tr>
<tr>
<td>public static final</td>
</tr>
<tr>
<td>Font specifier for unfocused text on the idle screen.</td>
</tr>
<tr>
<td>Value: 2</td>
</tr>
</tbody>
</table>
public static final FONT_INPUT_TEXT
Font specifier used by the implementation to draw text input by a user.
Value: 1

public static final FONT_STATIC_TEXT
Default font specifier used to draw Item and Screen contents.
Value: 0

public static final SIZE_LARGE
The "large" system-dependent font size.
Value: 16

public static final SIZE_MEDIUM
The "medium" system-dependent font size.
Value: 0

public static final SIZE_SMALL
The "small" system-dependent font size.
Value: 8

public static final STYLE_BOLD
The bold style constant.
Value: 1

public static final STYLE_ITALIC
The italicized style constant.
Value: 2

public static final STYLE_PLAIN
The plain style constant.
Value: 0

public static final STYLE_UNDERLINED
The underlined style constant.
Value: 4

Method Summary

int charsWidth(char[] ch, int offset, int length)
Returns the advance width for showing a substring of the array ch in this Font; starting at the specified offset and for the specified number of characters (length).

int charWidth(char ch)
Gets the advance width of the specified character in this Font.

static javax.microedition.lcdui.Font createFont(InputStream fontData)
Returns a new Font using the specified font input data.

javax.microedition.lcdui.Font deriveFont(int pixelSize)
Creates a new Font object by replicating this Font object and applying a new size to it.

javax.microedition.lcdui.Font deriveFont(int style, int pixelSize)
Creates a new Font object by replicating this Font object and applying a new style and size.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean equals(Object obj)</td>
<td>Compares this Font object to the specified Object.</td>
</tr>
<tr>
<td>int getAscent()</td>
<td>Gets the font ascent of this Font object.</td>
</tr>
<tr>
<td>static javax.microedition.lcdui.Font[]</td>
<td>Returns an array of Font where each Font object represents a physical font available in the system.</td>
</tr>
<tr>
<td>static javax.microedition.lcdui.Font[]</td>
<td>Returns an array of Font objects where each object represents a physical font having specified style.</td>
</tr>
<tr>
<td>static javax.microedition.lcdui.Font[]</td>
<td>Returns an array of Font objects where each object represents a physical font having specified face, style and pixelSize.</td>
</tr>
<tr>
<td>int getBaselinePosition()</td>
<td>Gets the distance in pixels from the top of the text to the text's baseline, which is defined by the MaxAscent value of a font.</td>
</tr>
<tr>
<td>static javax.microedition.lcdui.Font[]</td>
<td>Gets the default font of the system.</td>
</tr>
<tr>
<td>int getDescent()</td>
<td>Gets the font descent of this Font object.</td>
</tr>
<tr>
<td>int getFace()</td>
<td>Gets the face of the font.</td>
</tr>
<tr>
<td>java.lang.String getFamily()</td>
<td>Gets the family name of this Font.</td>
</tr>
<tr>
<td>static javax.microedition.lcdui.Font[]</td>
<td>Gets the Font used by the high level user interface for the fontSpecifier passed in.</td>
</tr>
<tr>
<td>static javax.microedition.lcdui.Font[]</td>
<td>Obtains an object representing a font (including custom fonts that are either packaged with a MIDlet or downloaded at run-time, if applicable) having the specified face, style and size.</td>
</tr>
<tr>
<td>static javax.microedition.lcdui.Font[]</td>
<td>Returns a new Font object from the font specified by name, style and size in pixels.</td>
</tr>
<tr>
<td>java.lang.String getFontName()</td>
<td>Gets the font face name of this Font.</td>
</tr>
<tr>
<td>int getHeight()</td>
<td>Gets the standard height of a line of text in this font.</td>
</tr>
<tr>
<td>int getLeading()</td>
<td>Gets the standard leading, in pixels, of this Font object.</td>
</tr>
<tr>
<td>int getMaxAscent()</td>
<td>Gets the maximum ascent of this Font.</td>
</tr>
<tr>
<td>int getMaxDescent()</td>
<td>Gets the maximum descent of this Font.</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getName()</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>int</td>
<td>getPixelSize()</td>
</tr>
<tr>
<td>static int</td>
<td>getPixelSize(String name)</td>
</tr>
<tr>
<td>int</td>
<td>getSize()</td>
</tr>
<tr>
<td>int</td>
<td>getStyle()</td>
</tr>
<tr>
<td>static int</td>
<td>getStyle(String name)</td>
</tr>
<tr>
<td>int</td>
<td>hashCode()</td>
</tr>
<tr>
<td>boolean</td>
<td>isBold()</td>
</tr>
<tr>
<td>boolean</td>
<td>isItalic()</td>
</tr>
<tr>
<td>boolean</td>
<td>isPlain()</td>
</tr>
<tr>
<td>boolean</td>
<td>isUnderlined()</td>
</tr>
<tr>
<td>int</td>
<td>stringWidth(String str)</td>
</tr>
<tr>
<td>int</td>
<td>substringWidth(String str, int offset, int len)</td>
</tr>
</tbody>
</table>

**Methods inherited from class Object**

getClass, notify, notifyAll, toString, wait, wait, wait

**Fields**

**FACE_MONOSPACEx**

public static final int FACE_MONOSPACEx

The "monospace" font face.

Value 32 is assigned to FACE_MONOSPACEx.

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_IDLE_HIGHLIGHTED_TEXT

public static final int FONT_IDLE_HIGHLIGHTED_TEXT

Font specifier for focused text on the idle screen.

FONT_IDLE_HIGHLIGHTED_TEXT can be is used with getFont to retrieve the appropriate Font for focused text on the IdleItem on the idle screen.

FONT_IDLE_HIGHLIGHTED_TEXT has the value 3.
Constant value: 3

Since: MIDP 3.0

FONT_IDLE_TEXT

public static final int FONT_IDLE_TEXT

Font specifier for unfocused text on the idle screen.

FONT_IDLE_TEXT can be is used with getFont to retrieve the appropriate Font for unfocused text on the IdleItem on the idle screen.

FONT_IDLE_TEXT has the value 2.
Constant value: 2

Since: MIDP 3.0

FONT_INPUT_TEXT

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: getFont(int)

Since: MIDP 2.0

FONT_STATIC_TEXT

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:** `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. Implementations SHOULD map this value to a particular font size in pixels, depending on device screen resolution and size.

Constant value: 16

---

**SIZE_MEDIUM**

```java
public static final int SIZE_MEDIUM
```

The "medium" system-dependent font size.

Value 0 is assigned to SIZE_MEDIUM. Implementations SHOULD map this value to a particular font size in pixels, depending on device screen resolution and size.

Constant value: 0

---

**SIZE_SMALL**

```java
public static final int SIZE_SMALL
```

The "small" system-dependent font size.

Value 8 is assigned to SIZE_SMALL. Implementations SHOULD map this value to a particular font size in pixels, depending on device screen resolution and size.

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 (except STYLE_PLAIN) for mixed styles.

Value 1 is assigned to STYLE_BOLD.

Constant value: 1

---

**STYLE_ITALIC**

```java
public static final int STYLE_ITALIC
```

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

Value 2 is assigned to STYLE_ITALIC.

Constant value: 2

---

**STYLE_PLAIN**

```java
public static final int STYLE_PLAIN
```

The plain style constant.

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 for showing a substring of the array `ch` in this `Font`; starting at the specified offset and for the specified number of characters (length). The advance width is the horizontal distance that would be occupied if the characters were to be drawn using this `Font`, including inter-character spacing following the characters necessary for proper positioning of subsequent text.

The `offset` and `length` parameters must specify a valid range of characters within the character array `ch`. The `offset` parameter must be within the range `[0..(ch.length)]`, inclusive. The `length` parameter must be a non-negative integer such that `(offset + length) <= ch.length`.

**Parameters:**
- `ch` - the array of characters
- `offset` - the index of the first character to measure
- `length` - the number of characters to be measured

**Returns:**
- the width of the character range

**Throws:**
- `ArrayIndexOutOfBoundsException` - if `offset` and `length` specify an invalid range
- `NullPointerException` - if `ch` is null

**See Also:** `substringWidth(String, int, int)`

#### 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:**
- `ch` - the character to be measured

**Returns:**
- the total advance width (a non-negative value)

#### createFont

**public static javax.microedition.lcdui.Font createFont(InputStream fontData)**

Throws `java.io.IOException`
Returns a new Font using the specified font input data. The new Font is created with a default size SIZE_MEDIUM and style STYLE_PLAIN. This base font can then be used with the deriveFont methods in this class to derive new Font objects with varying sizes and styles. This method does not close the InputStream.

Font data can be made persistent. When the createFont method is called (using either downloaded font data, or the data from the Record Store as an InputStream), the input font data shall be validated before the font is created. Implementations MUST insure that the availability and use of fonts created using createFont method is limited to the execution environment of a MIDlet that instantiated the font. The created fonts can be discovered using getAvailableFonts methods and can also be queried by getStyle method. After createFont is called, a new Font may also be instantiated using getFont or deriveFont methods.

Parameters:
- fontData - an InputStream object representing the input data for the font.

Returns:
- a new Font created from the specified font type and input data.

Throws:
- FontFormatException - if the fontData contains invalid font information, or if a font format is not supported.
- IOException - if the fontData cannot be completely read.

Since: MIDP 3.0

---

deriveFont

define the deriveFont

public javax.microedition.lcdui.Font deriveFont(int pixelSize)

Creates a new Font object by replicating this Font object and applying a new size to it.

Parameters:
- pixelSize - a positive integer value representing the size of this Font in pixels. When value of this argument is equal to zero, the Font object will have the default size SIZE_MEDIUM.

Returns:
- a new Font object.

Throws:
- IllegalArgumentException - if pixelSize is not a legal value (e.g. when the current font object is a non-scalable bitmap font)

Since: MIDP 3.0

---

deriveFont

public javax.microedition.lcdui.Font deriveFont(int style,
int pixelSize)

Creates a new Font object by replicating this Font object and applying a new style and size.

Parameters:
- style - the style constant for the Font. The style argument is an integer bitmask that may be a combination of STYLE_PLAIN and STYLE_UNDERLINED, or a bitwise union of STYLE_BOLD and/or STYLE_ITALIC, and a combination of them with STYLE_UNDERLINED (for example, STYLE_ITALIC or STYLE_ITALIC | STYLE_UNDERLINED | STYLE_BOLD | STYLE_ITALIC).
- pixelSize - a positive integer value representing the size of this Font in pixels. When value of this argument is equal to zero, the Font object will have the default size SIZE_MEDIUM.

Returns:
- a new Font object.

Throws:
- IllegalArgumentException - if style or pixelSize are not legal values (e.g. when the current font object is a non-scalable bitmap font)

Since: MIDP 3.0
equals

public boolean equals(Object obj)

Compares this Font object to the specified Object.

The equality of two Font objects is determined based on the comparison of their parameters such as font face name, style, pixel size, ascent and descent. However, since two Font objects may be instantiated from different font resources (either downloaded or resident) they may still differ, e.g. in character set coverage.

Overrides:

equals in class Object

Parameters:

obj - the Object to compare.

Returns:

true if the argument is a Font object having the same name, style, pixelSize, ascent and descent as this object; false otherwise.

Since: MIDP 3.0

getAscent

public int getAscent()

Gets the font ascent of this Font object. The font ascent is the distance from the font's baseline to the top of most alphanumeric characters. Some characters in the Font might extend above the font ascent line.

Returns:

the font ascent of the Font in pixels.

See Also: getMaxAscent()

Since: MIDP 3.0

getAvailableFonts

public static javax.microedition.lcdui.Font[] getAvailableFonts()

Returns an array of Font where each Font object represents a physical font available in the system. Font object will be created for each font face, even if they belong to the same font family. For example, if four different fonts representing faces "Arial", "Arial Italic", "Arial Bold" and "Arial Bold Italic" are available, the array returned by getAvailableFonts will have four different Font objects.

Fonts that are packaged with a MIDlet and declared in its manifest will be instantiated and included in the Font array. Fonts that are downloaded at run-time or stored in the Record Store will be included only after they are instantiated using createFont method.

Returns:

an array of Font representing all available physical fonts. Font objects are created with the default size.

See Also: getAvailableFonts(int), getAvailableFonts(int, int, int)

Since: MIDP 3.0
getAvailableFonts

public static javax.microedition.lcdui.Font[] getAvailableFonts(int style)

Returns an array of Font objects where each object represents a physical font having specified style. For example, if three different font families "Arial", "Courier New" and "Times New Roman" are available, and if style = STYLE_ITALIC - the Font[] will have three objects representing font faces "Arial Italic", "Courier New Italic" and "Times New Roman Italic".

Fonts that are packaged with a MIDlet and declared in its manifest will be instantiated and included in the Font array. Fonts that are downloaded at run-time or stored in the Record Store will be included only after they are instantiated using createFont method.

Parameters:
style - STYLE_PLAIN, STYLE_BOLD or STYLE_ITALIC, or a combination of STYLE_BOLD and STYLE_ITALIC.

Returns:
an array of Font representing available physical fonts having specified style. Font objects are created with the default size.

See Also: getAvailableFonts(), getAvailableFonts(int, int, int)
Since: MIDP 3.0

getAvailableFonts

public static javax.microedition.lcdui.Font[] getAvailableFonts(int face, int style, int pixelSize)

Returns an array of Font objects where each object represents a physical font having specified face, style and pixelSize. For example, if three different font families "Arial", "Courier New" and "Times New Roman" are available, and if face = FACE_PROPORTIONAL and style = STYLE_PLAIN - the Font[] will have two objects representing proportional font faces "Arial" and "Times New Roman" in the specified pixelSize.

Fonts that are packaged with a MIDlet and declared in its manifest will be instantiated and included in the Font array. Fonts that are downloaded at run-time or stored in the Record Store will be included only after they are instantiated using createFont method.

Parameters:
face - one of FACE_SYSTEM, FACE_MONOSPACe, or FACE_PROPORTIONAL
style - STYLE_PLAIN, STYLE_BOLD or STYLE_ITALIC, or a combination of STYLE_BOLD and STYLE_ITALIC
pixelSize - a positive integer value representing the size of this Font in pixels. When value of this argument is equal to zero, the Font object will have the default size SIZE_MEDIUM.

Returns:
an array of Font representing available fonts having specified face, style and size.

See Also: getAvailableFonts(), getAvailableFonts(int)
Since: MIDP 3.0

getBaselinePosition

public int getBaselinePosition()

Gets the distance in pixels from the top of the text to the text's baseline, which is defined by the MaxAscent value of a font.

Returns:
the distance in pixels from the top of the text to the text's baseline

See Also: getMaxAscent(), getMaxDescent()
get​​DefaultFont

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

Gets the default font of the system.

**Returns:**
the default font

getDescent

public int getDescent()

Gets the font descent of this Font object. The font descent is the distance from the font's baseline to the bottom of most alphanumeric characters. Some characters in the Font might extend below the font descent line.

**Returns:**
the font descent of the Font in pixels.

**See Also:** getMaxDescent()

**Since:** MIDP 3.0

getFace

public int getFace()

Gets the face of the font.

When this method is called with a Font object that was created from font data packaged with a MIDlet or downloaded at run-time and instantiated using createFont method, the call may return either FACE_PROPORTIONAL or FACE_MONOSPACE values, based on the information encoded in a font data.

**Returns:**
one of FACE_SYSTEM, FACE_PROPORTIONAL, FACE_MONOSPACE

getFamily

public java.lang.String getFamily()

Gets the family name of this Font.

The family name of a font is font specific. Two fonts such as Helvetica Italic and Helvetica Bold have the same family name, Helvetica, whereas their font face names are Helvetica Bold and Helvetica Italic.

Use getName() to get the logical name of the font. Use getFontName() to get the font face name of the font.

**Returns:**
String that is the family name of this Font.

**See Also:** getName(), getFontName()

**Since:** MIDP 3.0

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

Obtains an object representing a font (including custom fonts that are either packaged with a MIDlet or downloaded at run-time, if applicable) 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_MONOSPACING, or FACE_PROPORTIONAL
style - the style constant for the Font. The style argument is an integer bitmask that may be a combination of STYLE_PLAIN and STYLE_UNDERLINED, or a bitwise union of STYLE_BOLD and/or STYLE_ITALIC, and a combination of them with STYLE_UNDERLINED (for example, STYLE_ITALIC or STYLE_ITALIC | STYLE_UNDERLINED | STYLE_BOLD | STYLE_ITALIC).
size - one of SIZE_SMALL, SIZE_MEDIUM, or SIZE_LARGE.

Returns:
font instance of the nearest font found.

Throws:
IllegalArgumentException - if face, style, or size are not legal values

getFont

public static javax.microedition.lcdui.Font getFont(String name, int style, int pixelSize)

Returns a new Font object from the font specified by name, style and size in pixels.

The font name can be a font face name or a font family name. It is used together with the style to find an appropriate font face. When a font family name is specified, the style argument is used to select the most appropriate face from the family. When a font face name is specified, the face's style and the style argument are merged to locate the best matching font from the same family. For example if face name "Arial Bold" is specified with style Font.ITALIC, the font system looks for a face in the "Arial" family that is bold and italic, and may associate the font instance with the physical font face "Arial Bold Italic". The style argument is merged with the specified face's style, not added or subtracted. This means, specifying a bold face and a bold style does not double-embolden the font, and specifying a bold face and a plain style does not lighten the font. If the font face name specifies a particular style that is not the same as style argument, the font face name definition takes priority. For example, if name = "Arial Italic", and style = STYLE_PLAIN, the Font object will reference "Arial Italic".

Fonts that are packaged with a MIDlet and declared in its Manifest can be instantiated using this method. Fonts that are downloaded at run-time or stored in the Record Store may be referenced using this method, but only after they are instantiated using createFont method.

If no face for the requested style can be found, the font system may apply algorithmic styling to achieve the desired style. For example, if ITALIC is requested, but no italic face is available, glyphs from the plain face may be algorithmically obliqued (slanted).

Font name lookup is case insensitive, using the case folding rules of the US locale.

Parameters:
javax.microedition.lcdui.Font

name - the font name. This can be a font face name or a font family name, and may represent either a logical font or a physical font. The family names for logical fonts are: Dialog, DialogInput, Monospaced, Serif, or SansSerif. If name is null, the logical font name of the new Font as returned by getName() is set to the name “Default”.

style - the style constant for the Font. The style argument is an integer bitmask that may be a combination of STYLE_PLAIN and STYLE_UNDERLINED, or a bitwise union of STYLE_BOLD and/or STYLE_ITALIC, and a combination of them with STYLE_UNDERLINED (for example, STYLE_ITALIC or STYLE_ITALIC | STYLE_UNDERLINED or STYLE_BOLD | STYLE_ITALIC).

pixelSize - a positive integer value representing the size of this Font in pixels. When value of this argument is equal to zero, the Font object will have the default size SIZE_MEDIUM.

Returns:
font that corresponds to the passed in font name, style and size attributes.

Throws:
IllegalArgumentException - if the font with the specified name is not found, or if style or pixelSize are not legal values.

Since: MIDP 3.0

getFontName

public java.lang.String getFontName()

Gets the font face name of this Font. For example, Helvetica Bold could be returned as a font face name.

Use getFamily() to get the family name of the font. Use getName() to get the logical name of the font.

Returns:
String representing the font face name of this Font.

See Also: getFamily(), getName()

getHeight

public int getHeight()

Gets the standard height of a line of text in this font. This is the distance in pixels between two baselines of adjacent lines of text and is equal to the sum of the leading + ascent + descent. Due to rounding this may not be the same as getAscent() + getDescent() + getLeading(). There is no guarantee that lines of text spaced at this distance are disjoint; such lines may overlap if some characters overshoot either the standard ascent or the standard descent metric.

Returns:
standard height of a line of text in this font (a non-negative value), in pixels.

See Also: getLeading(), getAscent(), getDescent()

getLeading

public int getLeading()

Gets the standard leading, in pixels, of this Font object. The standard leading, or interline spacing, is the logical amount of space to be reserved between the descent of one line of text and the ascent of the next line. The height metric is calculated to include this extra space.

Returns:
standard leading of the Font in pixels.

See Also: getHeight(), getAscent(), getDescent()

Since: MIDP 3.0

gsetMaxAscent

public int getMaxAscent()
java.microedition.lcdui.Font

Gets the **maximum ascent** of this Font. No character extends further above the font's maximum ascent line.

**Returns:**
the maximum ascent of the Font in pixels.

**See Also:** `getAscent()`

**Since:** MIDP 3.0

---

`getMaxDescent`  
`public int getMaxDescent()`  

Gets the **maximum descent** of this Font. No character extends further below the font's maximum descent line.

**Returns:**
the font descent of the Font in pixels.

**See Also:** `getDescent()`

**Since:** MIDP 3.0

---

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

Gets the logical name of this Font.

Use `getFamily()` to get the family name of the font. Use `getFontName()` to get the font face name of the font.

**Returns:**
String representing the logical name of this Font.

**See Also:** `getFamily()`, `getFontName()`

---

`getPixelSize`  
`public int getPixelSize()`  

Gets the **pixelSize** of this Font. This pixelSize defines the height of the em box - a distance (in pixels) between the baselines of two unadjusted, adjacent lines of text in a single spaced text document.

**Returns:**
the size of this Font, in pixels.

**Since:** MIDP 3.0

---

`getPixelSize`  
`public static int getPixelSize(String name)`  

Gets the **pixelSize** supported by the font specified by name, in pixels.

If font is scalable, this method should return the value of 'zero'. If a font is the bitmap font, this pixelSize defines the height of the em box - a distance (in pixels) between the baselines of two unadjusted, adjacent lines of text in a single spaced text document.

**Parameters:**
name - the font name. This can be a font face name or a font family name, and may represent either a logical font or a physical font. The family names for logical fonts are: Dialog, DialogInput, Monospaced, Serif, or SansSerif. If name is null, the return value will indicate the size of the default system font.

**Returns:**
the size supported by the specified font, in pixels.
java.microedition.lcdui.Font

Throws:
IllegalArgument Exception - if the font with the specified name is not found.

Since: MIDP 3.0

getSize

public int getSize()

Gets the size of this Font. For a font object that was created using custom size in pixels, the return value indicates the range of a pixel size of a font.

- SIZE_SMALL is returned if the size of a font (in pixels) is smaller or equal to the system-dependent SIZE_SMALL;
- SIZE_MEDIUM is returned if the size of a font (in pixels) is larger then system-dependent SIZE_SMALL but smaller then system-dependent SIZE_LARGE;
- SIZE_LARGE is returned if the size of a font (in pixels) is equal or larger then system-dependent SIZE_LARGE;

Returns:
one of SIZE_SMALL, SIZE_MEDIUM or SIZE_LARGE.

getStyle

public int getStyle()

Gets the style of this Font. The value can be STYLE.PLAIN, STYLE.BOLD, STYLE.ITALIC, or STYLE.BOLD | STYLE.ITALIC, or a combination of them OR'ed with STYLE.UNDERLINED.

Returns:
style of this Font.

See Also: isPlain(), isBold(), isItalic(), isUnderlined()

getStyle

public static int getStyle(String name)

Gets the style of the font specified by name. The return value can be STYLE.PLAIN, STYLE.BOLD, STYLE.ITALIC, or STYLE_BOLD | STYLE.ITALIC.

Parameters:
name - the font name. This can be a font face name or a font family name, and may represent either a logical font or a physical font. The family names for logical fonts are: Dialog, DialogInput, Monospaced, Serif, or SansSerif. If name is null, the return value will indicate the style of the default system font.

Returns:
style of the specified font.

Throws:
IllegalArgument Exception - if the font with the specified name is not found.

Since: MIDP 3.0

hashCode

public int hashCode()

Gets the hash code value for this object. The hashcode must meet the general requirements of java.lang.Object.hashCode.
isBold

public boolean isBold()

Returns true if the font is bold.
Indicates whether or not this Font object's style is STYLE_BOLD.

Returns:
true if this Font is bold, false otherwise.
See Also: getStyle()

isItalic

public boolean isItalic()

Returns true if the font is italic.
Indicates whether or not this Font object's style is STYLE_ITALIC.

Returns:
true if this Font is italic, false otherwise.
See Also: getStyle()

isPlain

public boolean isPlain()

Returns true if the font is plain.
Indicates whether or not this Font object's style is STYLE_PLAIN.

Returns:
true if this Font is plain, false otherwise.
See Also: getStyle()

isUnderlined

public boolean isUnderlined()

Returns true if the font is underlined.
Indicates whether or not this Font object's style is STYLE_UNDERLINED.

Returns:
true if this Font is underlined.
See Also: 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. Note that the advance of a String is not necessarily the sum of the advance widths of its characters.

Parameters:
str - the String to be measured
Returns:
the total advance width

Throws:
NullPointerException - if str is null

See Also: charsWidth(char[], int, int), substringWidth(String, int, int)

__substringWidth__

```java
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. Note that the advance of a String is not necessarily the sum of the advance
widths of its characters.

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

See Also: charsWidth(char[], int, int)
javax.microedition.lcdui

FontFormatException

Declaration

public class FontFormatException extends RuntimeException

Object
  |-- Throwable
  |    |-- Exception
  |    |    |-- RuntimeException
  |    |    |    |-- javax.microedition.lcdui.FontFormatException

Description

Indicates that a font format is not supported, or that font data is invalid or is not conformant with the specified font format (OpenType with TrueType outlines).

This exception may be thrown by Font.createFont if the supplied font data is not valid.

Since: MIDP 3.0

See Also: Font

Constructor Summary

public FontFormatException()
Constructs an exception with no specified detail message.

public FontFormatException(String s)
Constructs an exception with the specified detail message.

Methods inherited from class Throwable

getMessage, printStackTrace, printStackTrace, toString

Methods inherited from class Object

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

Constructors

FontFormatException

public FontFormatException()

Constructs an exception with no specified detail message.

FontFormatException

public FontFormatException(String s)

Constructs an exception with the specified detail message.
javax.microedition.lcdui.FontFormatException

Parameters:
  s - the detail message
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 container(Form, Alert, or Display). If the application attempts to place an item into a Form, and the item is already owned by this or another container, an IllegalStateException is thrown. The application must remove the item from the current container 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.

Labels that contain line breaks may be truncated at the line break and cause the rest of the label not to be shown.

Layout

The default layout policy for Forms is the flow based defined in MIDP 2.0. However, the application can replace the default with an application provided layout policy instance. Layout policy can be delegated to instances of FormLayoutPolicy. The current layout policy is available from getLayoutPolicy. The method setLayoutPolicy changes the current layout policy. The layout policy may be changed at any time to a FormLayoutPolicy instance that was created for the specific Form. Each subclass provides a specific layout algorithm.

When the application sets a layout policy instance, the form delegates to the FormLayoutPolicy instance to determine the placement of Items. A particular layout policy implementation may be the cause of errors which the Form must tolerate. After a call to doLayout all of the Items within the x, y, viewPortWidth and viewPortHeight have been given positions. Every Item in the Form SHOULD NOT overlap with any other Item in the form. If any Item does not meet these requirements then the layout is in error and its appearance is unpredictable. Calls to the FormLayoutPolicy.doLayout and FormLayoutPolicy.getTraverse methods may throw runtime exceptions that must be caught and handled. The exceptions MUST be handled by reverting to the default layout policy as if setLayoutPolicy(null) had been called.
The Form must not compensate for bad or unusable Item placement produced by the doLayout method and MUST display the items that are visible on the form and clip them if necessary to the form and the screen. The Form does not need to compensate for a misbehaving FormLayoutPolicy.getTraverse method. If the method returns an unusable Item or null the implementation should refuse to change the focus. In the case of erroneous values or exceptions the Form MUST display any attached commands and enable the user to navigate away from the Form. The application is misbehaving and the user must be allowed to terminate it.

Default Layout

The default layout policy in Forms 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. The Form scrolls to keep the currently selected Item visible.

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 layout 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 this row to occupy the full width of the row.

If there are no stretchable items on this row, and there is still space available on this row, the Items are packed as tightly as possible and are placed on the row according to the alignment value shared by the Items on this row. (Since changing the current alignment causes a row break, all Items on the same row must share the same alignment value.) If the alignment value is LAYOUT_LEFT, the Items are positioned at the left end of the row and the remaining space is placed at the right end of the row. If the alignment value is LAYOUT_RIGHT, the Items are positioned at the right end of the row and the remaining space is placed at the left end of the row. If the alignment value is LAYOUT_CENTER, the Items are positioned in the middle of the row such that the remaining space on the row is divided evenly between the left and right ends of the row.

Given the set of items on a particular row, the heights of these Items are inspected. For each Item, the height that is used is the preferred height, unless the Item has the LAYOUT_VSHRINK directive, in which case the Item's minimum height is used. The height of the tallest Item determines the height of the row. Items that have the LAYOUT_VSHRINK directive are expanded to their preferred height or to the height of the row, whichever is smaller. Items that are still shorter than the row height and that have the
calls to the listener methods. The system can also notify the
and itemTraversedOut methods MUST be nested between the
calls to the traverse methods of the listener. The itemTraversedOut method MUST be called before another Item's itemTraversedIn method is called. When the Item is a CustomItem the calls to the traverse and traverseOut methods MUST be nested between the calls to the listener methods itemTraversedIn and itemTraversedOut. The system can also notify the

StringItems are treated specially in the above algorithm. If the contents of a StringItem (its string value, exclusive of its label) contain a newline character ("\n"), the string should be split at that point and the remainder laid out starting on the next row.

If one or both dimensions of the preferred size of a StringItem have been locked, the StringItem is wrapped to fit that width and height and is treated as a rectangle whose minimum and preferred width and height are the width and height of this rectangle. In this case, the LAYOUT_SHRINK, LAYOUT_EXPAND, and LAYOUT_VEXPAND directives are ignored.

If both dimensions of the preferred size of a StringItem are unlocked, the text from the StringItem may be wrapped across multiple rows. At the point in the layout algorithm where the width of the Item is compared to the remaining space on the row, as much text is taken from the beginning of the StringItem as will fit onto the current row. The contents of this row are then positioned according to the current alignment value. The remainder of the text in the StringItem is line-wrapped to the full width of as many new rows as are necessary to accommodate the text. Each full row is positioned according to the current alignment value. The last line of the text might leave space available on its row. If there is no row break following this StringItem, subsequent Items are packed into the remaining space and the contents of the row are positioned according to the current alignment value. This rule has the effect of displaying the contents of a StringItem as a paragraph of text set flush-left, flush-right, or centered, depending upon whether the current alignment value is LAYOUT_LEFT, LAYOUT_RIGHT, or LAYOUT_CENTER, respectively. The preferred width and height of a StringItem wrapped across multiple rows, as reported by the Item.getPreferredWidth and Item.getPreferredHeight methods, describe the width and height of the bounding rectangle of the wrapped text.

ImageItems are also treated specially by the above algorithm. The foregoing rules concerning the horizontal alignment value and the LAYOUT_LEFT, LAYOUT_RIGHT, and LAYOUT_CENTER directives, apply to ImageItems only when the LAYOUT_2 directive is also present on that item. If the LAYOUT_2 directive is not present on an ImageItem, the behavior of the LAYOUT_LEFT, LAYOUT_RIGHT, and LAYOUT_CENTER directives is implementation-specific.

A Form's layout is recomputed automatically as necessary. This may occur because of a change in an Item's size caused by a change in its contents or because of a request by the application to change the Item's preferred size. It may also occur if an Item's layout directives are changed by the application. The application does not need to perform any specific action to cause the Form's layout to be updated.

Line Breaks and Wrapping

For all cases where text is wrapped, line breaks must occur at each newline character ('
' = '\u000a'). 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.

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). The Form MUST notify the application when the user traverses between Items contained within the Form if a ItemTraversallListener has been set using set.ItemTraversallListener() method. Notifications are accomplished by calling the itemTraversedIn() and itemTraversedOut() methods of the listener. The ItemTraversedIn method MUST be called before traversal into an Item occurs. The ItemTraversedOut method MUST be called after traversal out of an Item occurs. When ItemTraversedIn has been called on an Item; the item's ItemTraversedOut MUST be called before another Item's ItemTraversedIn method is called. When the Item is a CustomItem the calls to the traverse and traverseOut methods MUST be nested between the calls to the listener methods itemTraversedIn and itemTraversedOut. The system can also notify 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 setItemStateChangedListener() method.

As with other Displayable objects, a Form can declare commands and declare a command listener with the setCommandListener() method. CommandListener objects are distinct from ItemStateListener and ItemTraverseListener 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

<table>
<thead>
<tr>
<th>Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public Form(String title)</code></td>
</tr>
<tr>
<td>Creates a new, empty Form</td>
</tr>
<tr>
<td><code>public Form(String title, Item[] items)</code></td>
</tr>
<tr>
<td>Creates a new Form with the specified contents.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int append(Image img)</code></td>
</tr>
<tr>
<td>Adds an item consisting of one Image to the Form.</td>
</tr>
<tr>
<td><code>int append(Item item)</code></td>
</tr>
<tr>
<td>Adds an Item into the Form.</td>
</tr>
<tr>
<td><code>int append(String str)</code></td>
</tr>
<tr>
<td>Adds an item consisting of one String to the Form.</td>
</tr>
<tr>
<td><code>void delete(int itemNum)</code></td>
</tr>
<tr>
<td>Deletes the Item referenced by itemNum.</td>
</tr>
<tr>
<td><code>void deleteAll()</code></td>
</tr>
<tr>
<td>Deletes all the items from this Form, leaving it with zero items.</td>
</tr>
<tr>
<td><code>javax.microedition.lcdui.Item get(int itemNum)</code></td>
</tr>
<tr>
<td>Gets the item at given position.</td>
</tr>
<tr>
<td><code>javax.microedition.lcdui.Item getCurrent()</code></td>
</tr>
<tr>
<td>Gets the currently focused Item in the Form.</td>
</tr>
<tr>
<td><code>int getHeight()</code></td>
</tr>
<tr>
<td>Returns the height in pixels of the displayable area available for Items.</td>
</tr>
<tr>
<td><code>javax.microedition.lcdui.FormLayoutPolicy getLayoutPolicy()</code></td>
</tr>
<tr>
<td>Gets the FormLayoutPolicy for this form.</td>
</tr>
<tr>
<td>Method</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td><code>int getWidth()</code></td>
</tr>
<tr>
<td><code>void insert(int itemNum, Item item)</code></td>
</tr>
<tr>
<td><code>void set(int itemNum, Item item)</code></td>
</tr>
<tr>
<td><code>void setItemStateListener(ItemStateListener iListener)</code></td>
</tr>
<tr>
<td><code>void setItemTraversalListener(ItemTraversalListener iListener)</code></td>
</tr>
<tr>
<td><code>void setLayoutPolicy(FormLayoutPolicy layoutPolicy)</code></td>
</tr>
<tr>
<td><code>int size()</code></td>
</tr>
</tbody>
</table>

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

- `addCommand`, `getCommand`, `getCommandLayoutPolicy`, `getCommands`, `getCurrentDisplay`, `getMenu`, `getTicker`, `getTitle`, `invalidateCommandLayout`, `isShown`, `removeCommand`, `removeCommandOrMenu`, `setCommand`, `setCommandLayoutPolicy`, `setCommandListener`, `setMenu`, `setTicker`, `setTitle`, `sizeChanged`

Methods inherited from class `Object`:

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

Constructors

```java
public Form(String title)

Creates a new, empty Form.

Parameters:
- `title` - the Form's title, or `null` for no title
```

```java
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
- `DisplayCapabilityException` - if the `Item` has Commands and this `Form` is currently being displayed on a `Display` that does not support Commands

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

---

### getCurrent

```java
public javax.microedition.lcdui.Item getCurrent()
```

Gets the currently focused Item in the Form. An Item has focus when it is selected as the active recipient of regular key events.

**Returns:**
the currently focused Item or null if there is no focused Item in the Form

---

### getHeight

```java
public int getHeight()
```

Returns the height in pixels of the displayable area available for Items. This value is the height available to items in the Form that can be shown without scrolling. The height may change depending on the presence of a horizontal scrollbar. If the Form is bound to a Display through a call to setCurrent, the height returned MUST NOT be larger that Display.getHeight. If a Form in a TabbedPane is brought into focus, then the Height returned MUST NOT be larger than TabbedPane.getHeight.

**Returns:**
the height of the area available for Items in pixels

**Since:** MIDP 2.0

---

### getLayoutPolicy

```java
public javax.microedition.lcdui.FormLayoutPolicy getLayoutPolicy()
```

Gets the FormLayoutPolicy for this form. Returns the last value set in a call to setLayoutPolicy or null if it was not set.

**Returns:**
the FormLayoutPolicy for this form; is null when the default layout policy is in use.
getWidth

public int getWidth()

Returns the width in pixels of the displayable area available for Items. This value is the width available to items in the Form that can be shown without scrolling. The width may change depending on the presence of a vertical scrollbar. If the Form is bound to a Display through a call to setCurrent, the width returned MUST NOT be larger than Display.getWidth. If a Form in a TabbedPane is brought into focus, then the width returned MUST NOT be larger than TabbedPane.getWidth.

Returns:
the width of the area available for Items in pixels

Since: MIDP 2.0

insert

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()], 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
DisplayCapabilityException - if the Item has Commands and this Form is currently being displayed on a Display that does not support Commands

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, it simply removes the previous ItemStateListener.

Parameters:
iListener - the new listener, or null to remove it

setItemTraversalListener

public void setItemTraversalListener(ItemTraversalListener iListener)

Sets the ItemTraversalListener for the Form, replacing any previous ItemTraversalListener. If iListener is null, it simply removes the previous ItemTraversalListener.

Parameters:
iListener - the new listener, or null to remove it

Since: MIDP 3.0

setLayoutPolicy

public void setLayoutPolicy(FormLayoutPolicy layoutPolicy)

Sets the FormLayoutPolicy for this form. The layout policy takes effect as soon as it is feasible for the implementation to do so. The FormLayoutPolicy instance must have been created to refer to this Form. The layout policy can be set to null to use the default layout policy.

Applications should avoid changing the layout while the form is visible. It may cause user confusion and loss of visual context. The current Item in the Form must not change. If the Form is currently being displayed the current Item must remain visible.

Parameters:
layoutPolicy - the new FormLayoutPolicy for this Form; may be null to select the default layout policy.

Throws:
IllegalArgumentException - if layout policy is not null and FormLayoutPolicy.getForm() does not refer to this Form.

See Also: getLayoutPolicy()

Since: MIDP 3.0

size

public int size()

Gets the number of items in the Form.

Returns:
the number of items
javax.microedition.lcdui.FormLayoutPolicy

Declaration

public abstract class FormLayoutPolicy

Object

+---javax.microedition.lcdui.FormLayoutPolicy

Direct Known Subclasses:
  javax.microedition.lcdui.TableLayoutPolicy

Description

FormLayoutPolicy is subclassed to provide custom layout algorithms. Form delegates layout functions to an instance of FormLayoutPolicy. Each FormLayoutPolicy is bound to a single Form when it is created.

The Items in the Form are accessed via the Form instance using the Form.size and Form.get methods. Each Item has methods to access layout directives and to the contents. Each FormLayoutPolicy subclass defines the effect of each layout directive.

The layout algorithm is implemented by subclasses in the doLayout method. The doLayout method is called when the layout of items is invalid and needs to be updated. The position and size of each Item is made available through the getX, getY, getWidth, getHeight methods. The x, y, width, and height are set with the setPosition and setSize methods. Setting and updating the position and size of each Item is the responsibility of the FormLayoutPolicy subclass. The values are initially zero, the values are only modified by the layout policy and otherwise remain unmodified. The methods to get and set Item position, size, or validity must not be called outside of the scope of a call to the doLayout or getTraverse methods.

When the contents or size of Items change it maybe necessary to update the layout. When items change in a way that might require the layout to be updated the Items MUST be marked invalid by calling setValid(item, false). When a valid layout is needed, for example to display the Form, the doLayout method is called to compute and update, as necessary, the position and size of invalid Items. Only the doLayout method should set the valid item to true.

When a Form is active on a display, updates to the Items or Form take place as soon as it is feasible for the implementation to do so. The doLayout method is called to recompute the layout when there are invalid Items and the display needs to be updated. Calls to doLayout may result from calls to Display.setCurrent, Form.delete, Form.insert, Form.append, Form.set, Displayable.setTitle, Displayable.setTicker, Item methods for setting labels, commands, preferred size, contents, etc.

After doLayout returns, the caller can assume that all of the Items within the viewport are valid and have valid positions and sizes and may display them as necessary; though the actual valid flags may not be set to true. Items that extend beyond the viewport MUST be clipped when rendered.

If the doLayout or traverse methods throw runtime exceptions the Form MUST set the layout policy to null and revert the layout to the default layout policy.

Since: MIDP 3.0
See Also: Form

Field Summary

<table>
<thead>
<tr>
<th>public static final</th>
<th>DIRECTION_LTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indicates left-to-right mode for layout direction.</td>
<td></td>
</tr>
<tr>
<td>Value: 0</td>
<td></td>
</tr>
</tbody>
</table>
### Constructor Summary

```java
protected FormLayoutPolicy(Form form)
Creates a new instance of FormLayoutPolicy and binds it to the Form.
```

### Method Summary

#### Abstract Method

```java
abstract void doLayout(int viewportX, int viewportY, int viewportWidth, int viewportHeight, int[] totalSize)
Compute or update the position and size of Items in the Form within the viewport.
```

```java
javax.microedition.lcdui.Form getForm()
Gets the Form for which this FormLayoutPolicy is handling the layout.
```

```java
int getHeight(Item item)
Gets the height of the Item.
```

```java
static int getLayoutDirection()
Gets the layout direction determined by the platform, either DIRECTION_LTR or DIRECTION_RTL.
```

```java
abstract javax.microedition.lcdui.Item getTraverse(Item item, int dir)
Gets the Item logically adjacent to an existing Item in the traversal direction.
```

```java
int getWidth(Item item)
Gets the current width of the Item.
```

```java
int getX(Item item)
Gets the x position of the Item.
```

```java
int getY(Item item)
Gets the y position of the Item.
```

```java
boolean isValid(Item item)
Returns true if the position and of the Item are valid.
```

```java
void setPosition(Item item, int x, int y)
Sets the position of the Item.
```

```java
void setSize(Item item, int width, int height)
Sets the width and height of the Item.
```

```java
void setValid(Item item, boolean valid)
Sets the validity of the Item.
```

### Methods inherited from class Object

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

### Fields

---

**javax.microedition.lcdui.FormLayoutPolicy**

```
public static final DIRECTION_RTL
Indicates right-to-left mode for layout direction.
Value: 1
```

---

**Constructor Summary**

```java
protected FormLayoutPolicy(Form form)
Creates a new instance of FormLayoutPolicy and binds it to the Form.
```
**DIRECTION_LTR**

```java
public static final int DIRECTION_LTR
```

Indicates left-to-right mode for layout direction.

Value 0 is assigned to **DIRECTION_LTR**.

Constant value: 0

Since: MIDP 3.0

---

**DIRECTION_RTL**

```java
public static final int DIRECTION_RTL
```

Indicates right-to-left mode for layout direction.

Value 1 is assigned to **DIRECTION_RTL**.

Constant value: 1

Since: MIDP 3.0

### Constructors

**FormLayoutPolicy**

```java
protected FormLayoutPolicy (Form form)
```

Creates a new instance of FormLayoutPolicy and binds it to the Form.

**Parameters:**

- `form` - the Form bound to this layout.

**Throws:**

- `NullPointerException` - if `form` is null.

### Methods

**doLayout**

```java
protected abstract void doLayout (int viewportX,
                                 int viewportY,
                                 int viewportWidth,
                                 int viewportHeight,
                                 int[] totalSize)
```

Compute or update the position and size of Items in the Form within the viewport. The viewport is defined by a width and height and its x and y position within the form. The implementation of this method is expected to set the position and size of Items so that they do not overlap, if not, the layout is in error and its appearance may be unpredictable. The Items SHOULD be placed within the viewport but some visual arrangements cannot by their nature be fit within a specified fixed area, for example, a spreadsheet with a large number of rows or columns.

The initial position and size of each Item is zero until updated by the layout algorithm. The values are only modified by the layout algorithm and will be valid from the previous layout. Any Item that has changed is invalid and its position and size may need to be updated and set as valid. The size of the screen available to display the items of the Form is provided as a viewport width and height. To support scrolling, this method must return the total width and height required for the complete form, including all items. The implementation can use the returned total width and height to provide feedback if scrolling is supported. Only items within the viewport need to be made valid by each doLayout method call.

The doLayout method should not make any changes to Item contents or invalidate Items. Changes might affect the layout and can cause an unstable layout.
getForm

protected final javax.microedition.lcdui.Form getForm()

Gets the Form for which this FormLayoutPolicy is handling the layout.

Returns:
The Form for which this instance is handling the layout.

getHeigh

protected final int getHeight(Item item)

Gets the height of the Item.

Parameters:
item - The item from which to get the height.

Returns:
The height of the Item.

Throws:
NullPointerException - if item is null.
IllegalArgumentException - if item is not an Item in this layout's form.
IllegalStateException - if called outside of the scope of a call to getTraverse or doLayout.

getLayoutDirection

public final static int getLayoutDirection()

Gets the layout direction determined by the platform, either DIRECTION_LTR or DIRECTION_RTL.
The layout direction should be based on the current language convention in use.

Returns:
the layout direction DIRECTION_LTR or DIRECTION_RTL.

Since: MIDP 3.0

getTraverse

protected abstract javax.microedition.lcdui.Item getTraverse(Item item, int dir)
Gets the Item logically adjacent to an existing Item in the traversal direction. The `getTraverse` method is overridden by subclasses so that the next item in any direction is consistent with the layout and the user concept of the next Item in each direction taking into account wrapping between left/right and top/bottom and progression, if any, to the next line. For example, if the current item is the last item in a row then the next Item to the right is most likely the first Item in the next row.

The traversal direction must be one of four directions `Canvas.UP`, `Canvas.DOWN`, `Canvas.LEFT`, or `Canvas.RIGHT`.

**Parameters:**
- `item` - a current Item
- `dir` - the traversal direction from the item to the adjacent item

**Returns:**
- the item in the traversal direction requested; may be null.

**Throws:**
- `NullPointerException` - if `item` is null.
- `IllegalArgumentException` - if `dir` is not one of `Canvas.UP`, `Canvas.DOWN`, `Canvas.LEFT`, or `Canvas.RIGHT`.
- `IllegalArgumentException` - if `item` is not an Item in this layout's form.

---

### get Width

```java
protected final int getWidth(Item item)
```

Gets the current width of the Item.

**Parameters:**
- `item` - The item for which to get the width.

**Returns:**
- The width of the Item.

**Throws:**
- `NullPointerException` - if `item` is null.
- `IllegalArgumentException` - if `item` is not an Item in this layout's form.
- `IllegalStateException` - if called outside of the scope of a call to `getTraverse` or `doLayout`.

---

### get X

```java
protected final int getX(Item item)
```

Gets the x position of the Item.

**Parameters:**
- `item` - The item for which to get the x position.

**Returns:**
- The x position of the item.

**Throws:**
- `NullPointerException` - if `item` is null.
- `IllegalArgumentException` - if `item` is not an Item in this layout's form.
- `IllegalStateException` - if called outside of the scope of a call to `getTraverse` or `doLayout`.

---

### get Y

```java
protected final int getY(Item item)
```

Gets the y position of the Item.

**Parameters:**
- `item` - The item for which to get the y position.

**Returns:**
- The y position of the Item.
Throws:
NullPointerException - if item is null.
IllegalArgumentException - if item is not an Item in this layout's form.
IllegalStateException - if called outside of the scope of a call to getTraverse or doLayout.

isValid

protected final boolean isValid(Item item)

Returns true if the position and of the Item are valid.

Parameters:
item - The item for which to get the validity.

Returns:
true if the position and size are valid; false otherwise.

Throws:
NullPointerException - if item is null.
IllegalArgumentException - if item is not an Item in this layout's form.
IllegalStateException - if called outside of the scope of a call to getTraverse or doLayout.

setPosition

protected final void setPosition(Item item, int x, int y)

Sets the position of the Item.

Parameters:
item - The item to be positioned at x and y.
x - The new x position of the Item; MUST be greater than or equal to zero.
y - The new y position of the Item; MUST be greater than or equal to zero.

Throws:
NullPointerException - if item is null.
IllegalStateException - if called outside of the scope of a call to getTraverse or doLayout.
IllegalArgumentException - if x or y is less than zero.

setSize

protected final void setSize(Item item, int width, int height)

Sets the width and height of the Item.

Parameters:
item - The Item for which to set the width and height; MUST NOT be null.
width - The new width of the Item.
height - The new height of the Item.

Throws:
NullPointerException - if item is null.
IllegalStateException - if called outside of the scope of a call to getTraverse or doLayout.
IllegalArgumentExcedtion - if width is less than the Item's Item.getMinimumWidth or height is less than the Item.getMinimumHeight
IllegalArgumentException - if item is not an Item in this layout's form.

setValid

protected final void setValid(Item item, boolean valid)
Sets the validity of the Item.

**Parameters:**
- item - The Item for which to set the validity.
- valid - true if the position and size are valid; false otherwise.

**Throws:**
- NullPointerException - if item is null.
- IllegalArgumentException - if item is not an Item in this layout's form.
- IllegalStateException - if called outside of the scope of a call to `getTraverse` or `doLayout`.
Gauge

Declaration

```java
public class Gauge extends Item
```

Description

Implements a graphical display, such as a bar graph, of an integer value. The Gauge contains a current value that lies between zero and the maximum value, inclusive. The application can control the current value and maximum value. The range of values specified by the application may be larger than the number of distinct visual states possible on the device, so more than one value may have the same visual representation.

For example, consider a Gauge object that has a range of values from zero to 99, running on a device that displays the Gauge's approximate value using a set of one to ten bars. The device might show one bar for values zero through nine, two bars for values ten through 19, three bars for values 20 through 29, and so forth.

A Gauge may be interactive or non-interactive. Applications may set or retrieve the Gauge's value at any time regardless of the interaction mode. The implementation may change the visual appearance of the bar graph depending on whether the object is created in interactive mode.

In interactive mode, the user is allowed to modify the value. The user will always have the means to change the value up or down by one and may also have the means to change the value in greater increments. The user is prohibited from moving the value outside the established range. The expected behavior is that the application sets the initial value and then allows the user to modify the value thereafter. However, the application is not prohibited from modifying the value even while the user is interacting with it.

In many cases the only means for the user to modify the value will be to press a button to increase or decrease the value by one unit at a time. Therefore, applications should specify a range of no more than a few dozen values.

In non-interactive mode, the user is prohibited from modifying the value. Non-interactive mode is used to provide feedback to the user on the state of a long-running operation. One expected use of the non-interactive mode is as a "progress indicator" or "activity indicator" to give the user some feedback during a long-running operation. The application may update the value periodically using the setValue() method.

A non-interactive Gauge can have a definite or indefinite range. If a Gauge has definite range, it will have an integer value between zero and the maximum value set by the application, inclusive. The implementation will provide a graphical representation of this value such as described above.

A non-interactive Gauge that has indefinite range will exist in one of four states: continuous-idle, incremental-idle, continuous-running, or incremental-updating. These states are intended to indicate to the user that some level of activity is occurring. With incremental-updating, progress can be indicated to the user even though there is no known endpoint to the activity. With continuous-running, there is no progress that gets reported to the user and there is no known endpoint; continuous-running is merely a busy state indicator. The implementation should use a graphical display that shows this appropriately. The implementation may use different graphics for indefinite continuous gauges and indefinite incremental gauges. Because of this, separate idle states exist for each mode. For example, the implementation might show an hourglass or spinning watch in the continuous-running state, but show an animation with different states, like a beach ball or candy-striped bar, in the incremental-updating state.
In the continuous-idle or incremental-idle state, the `Gauge` indicates that no activity is occurring. In the incremental-updating state, the `Gauge` indicates activity, but its graphical representation should be updated only when the application requests an update with a call to `setValue()`. In the continuous-running state, the `Gauge` indicates activity by showing an animation that runs continuously, without update requests from the application.

The values `CONTINUOUS_IDLE`, `INCREMENTAL_IDLE`, `CONTINUOUS_RUNNING`, and `INCREMENTAL_UPDATING` have their special meaning only when the `Gauge` is non-interactive and has been set to have indefinite range. They are treated as ordinary values if the `Gauge` is interactive or if it has been set to have a definite range.

An application using the `Gauge` as a progress indicator should typically also attach a `STOP` 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>Public static final</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final</td>
<td>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</td>
<td>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</td>
<td>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</td>
<td>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>
</tbody>
</table>
**javax.microedition.lcdui.Gauge**

```java
public static final INDEFINITE
A special value used for the maximum value in order to indicate that the Gauge has indefinite range.
Value: -1
```

**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 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.
```

**Method Summary**

```java
int getIncrementValue()
Gets the increment value of this Gauge object.

int getMaxValue()
Gets the maximum value of this Gauge object.

int getMinValue()
Gets the minimum value of this Gauge object.

int getValue()
Gets the current value of this Gauge object.

boolean isInteractive()
Tells whether the user is allowed to change the value of the Gauge.

void setIncrementValue(int incrementValue)
Sets the increment value of this Gauge object.

void setMaxValue(int maxValue)
Sets the maximum value of this Gauge object.

void setMinValue(int minValue)
Sets the minimum value of this Gauge object.

void setValue(int value)
Sets the current value of this Gauge object.
```

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

addCommand, getCommands, getLabel, getLayout, getLayoutHint, getMinimumHeight, getMinimumWidth, getPreferredHeight, getPreferredWidth, notifyStateChanged, removeCommand, setCommand, setDefaultCommand, setItemCommandListener, setLabel, setLayout, setLayoutHint, 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

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

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

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

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: INDEFINITE, CONTINUOUS_IDLE, INCREMENTAL_IDLE, CONTINUOUS_RUNNING, INCREMENTAL_UPDATING

Methods

getIncrementValue

public int getIncrementValue()

Gets the increment value of this Gauge object.

If the setIncrementValue method has not been called the default increment value 1 MUST be returned for the Gauge object.
javax.microedition.lcdui.Gauge

Returns:
the increment value of the Gauge.

See Also: setIncrementValue(int)

Since: MIDP 3.0

getMaxValue

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: INDEFINITE,setMaxValue(int)

getMinValue

public int getMinValue()

Gets the minimum value of this Gauge object.

If the setMinValue method has not been called the default minimum value 0 MUST be returned for
the Gauge object.

Returns:
the minimum value of the Gauge.

See Also: setMinValue(int)

Since: MIDP 3.0

getValue

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: CONTINUOUS_IDLE, INCREMENTAL_IDLE, CONTINUOUS_RUNNING, INCREMENTAL_UPDATING,
setMinValue(0)

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

setIncrementValue

public void setIncrementValue(int incrementValue)
Sets the increment value of this Gauge object.

The increment value for the gauge must be greater than or equal to 1 and less than or equal to the maximum value, otherwise an exception is thrown. If this method has not been called the default increment value for the Gauge object is 1.

Parameters:
incrementValue - the new increment value

Throws:
IllegalArgumentException - if incrementValue is invalid

See Also: getIncrementValue()
Since: MIDP 3.0

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: INDEFINITE, getMaxValue()
See Also: `getMinValue()`
Since: MIDP 3.0

---

**setValue**

```java
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: `CONTINUOUS_IDLE`, `INCREMENTAL_IDLE`, `CONTINUOUS_RUNNING`, `INCREMENTAL_UPDATING`, `getValue()`
javax.microedition.lcdui.Graphics

Declaration

public class Graphics

Object

+--javax.microedition.lcdui.Graphics

Description

Provides simple 2D geometric rendering capability.

All implementations MUST support double-buffered graphics. Graphics may be rendered to the display's off-screen buffer 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.

In drawing methods that take an Image as a parameter: if the Image is an instance of ScalableImage, it MUST be rasterized before drawing. If the scalable image does not define an initial viewport size, the default viewport of 100 by 100 pixels MUST be used. If the scalable image contains animation, the rasterized starting frame of the animation MUST be used when drawing.

Coordinate System

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

```java
    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.)

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
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

\[
\begin{align*}
g.\text{drawRect}(x, y, w, h); & \quad // 2 \\
g.\text{fillRect}(x, y, w, h); & \quad // 1 
\end{align*}
\]

has an effect identical to statement (2) above.

The exact pixels painted by \( \text{drawLine()} \) and \( \text{drawArc()} \) 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 \( \text{Graphics} \) object that may be modified by graphics rendering operations.

There is a single clip per \( \text{Graphics} \) 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 \( \text{translate()} \), do not modify the clip. The methods \( \text{getClipX()}, \text{getClipY()}, \text{getClipWidth()} \) and \( \text{getClipHeight()} \) must return a rectangle that, if passed to \( \text{setClip()} \) without an intervening change to the \( \text{Graphics} \) object's coordinate system, must result in the identical set of pixels in the clip. The rectangle returned from the \( \text{getClip} \) family of methods may differ from the clip rectangle that was requested in \( \text{setClip()} \). 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 \( \text{Graphics} \) 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,

\[
\begin{align*}
g.\text{drawLine}(x, y, x + w, y); \\
g.\text{drawLine}(x + w, y, x + w, y + h); \\
g.\text{drawLine}(x + w, y + h, x, y + h); \\
g.\text{drawLine}(x, y + h, x, y); 
\end{align*}
\]
The pixels touched by statement (4) must be identical to the pixels within \((cx, cy, cw, ch)\) touched by statement (3).

**Color Model**

The API supports a 32-bit ARGB color model, with 8 bits for the alpha, red, green, and blue components of a color. Given its pervasive use for internet applications and adoption by the W3C, MIDP devices should conform to the sRGB color space as closely as possible when interpreting RGB color values.

Not all devices support a full 32 bits' worth of color and thus they will map colors requested by the application into colors available on the device. The color depth of off-screen images must not be lower than that of any display that is an integral part of the device. Auxiliary displays that the device connects to may have higher color depths, in which case colors are converted accordingly.

Facilities are provided in the `Display` class for obtaining a specific display's 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.

**Alpha Level and Blending Modes**

In addition to a drawing color, each Graphics object also has an alpha value that controls the overall opacity of the rendered pixels. As with color values, not all devices will support a full 8 bits of alpha channel information. However, all implementations must support at least 4-bit alpha values regardless of their display bit depths.

For the text, line, rectangle, and arc drawing and filling primitives, the source pixel is a pixel having the current color and alpha value of the graphics object.

However, the `drawImage()`, `drawRegion()`, and `drawRGB()` methods use an image or array of pixel values as the source for rendering operations instead of the current color of the graphics object. For these operations, the alpha value of a given source pixel is multiplied by the alpha value of the Graphics object to determine its overall opacity when rendered to the destination. Hence, a fully opaque pixel in the source will adopt the alpha level of the Graphics object. Similarly, the alpha values of the source pixels will be rendered as-is if the alpha value of the graphics object is fully opaque.

Rendered pixels are combined with the destination pixels according to the Graphics object's current blending mode. Two of the Porter-Duff blending modes are supported: `SRC_OVER`, and `SRC`.

`SRC_OVER` is the default blending mode and blends the source pixel on top of the destination pixel. If the source pixel is fully opaque, the destination pixel is effectively replaced with the source pixel. If the source pixel is fully transparent, the destination pixel is unchanged. If the source pixel is partially transparent, its color is blended with the color of the destination pixel. The opacity of the destination pixel cannot be reduced using this blending mode, and thus it may be used on images and surfaces that do not support alpha channels as their pixels are already fully opaque.

The `SRC_OVER` mode computes the destination pixel's red, green, blue, and alpha values according to the follow equations:
where the alpha value A has a value between 0 and 1.0

The SRC blending mode replaces the destination pixel with the source pixel, regardless of the source pixel's opacity. Both the color and the alpha value of the destination pixel are replaced, thus allowing the opacity of the destination pixel to be decreased as well as increased. For this reason, the Source blending mode can only be used for Graphics objects that render to an Image with an alpha channel.

The SRC mode computes the destination pixel's red, green, blue, and alpha values according to the following equations:

\[
\begin{align*}
R(\text{dest}) &= (R(\text{src}) \times A(\text{src})) + (R(\text{dest}) \times (1 - A(\text{src}))) \\
G(\text{dest}) &= (G(\text{src}) \times A(\text{src})) + (G(\text{dest}) \times (1 - A(\text{src}))) \\
B(\text{dest}) &= (B(\text{src}) \times A(\text{src})) + (B(\text{dest}) \times (1 - A(\text{src}))) \\
A(\text{dest}) &= A(\text{src}) + A(\text{dest}) - (A(\text{src}) \times A(\text{dest}))
\end{align*}
\]

Stroke Styles

Lines, arcs, rectangles, and rounded rectangles are 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 and alpha level; spaces between dots are left untouched.

Rendering Text

For a given font, each character is represented by a glyph. The glyph acts as an alpha channel mask containing pixels that form the shape of the character.

Basic font engines provide monochrome glyphs in which a pixel is either fully opaque (that is, part of the character to be drawn) or fully transparent. More sophisticated font engines provide graymap glyphs in which pixels may also be partially opaque to varying degrees, thereby allowing character edges to appear smoother.
When a character is painted, the opacity of a rendered pixel is determined by multiplying the alpha value of the glyph's pixel with that of the Graphics object. The color of the rendered pixel will be the color of the Graphics object. Hence, pixels that are fully opaque in the glyph will be rendered with the Graphics object's opacity and color, and pixels that are fully transparent in the glyph will be rendered as fully transparent pixels.

The text drawing calls `drawChar()`, `drawChars()`, `drawString()`, `drawSubstring()` and `drawText()` all draw text in this manner.

The text drawing calls `drawChars()`, `drawString()`, and `drawSubstring()` all render characters in their exact order. In other words, no bi-directional processing is applied and the order of the characters is preserved. If bi-directional text is required, the String or character data should be pre-processed using a suitable library or the `Text` class should be used instead.

### Anchor Points

The drawing of text is based on "anchor points". Anchor points are used to minimize the amount of computation required when placing text. For example, in order to center a piece of text, an application needs to call `stringWidth()` or `charWidth()` to get the width and then perform a combination of subtraction and division to compute the proper location. The method to draw text is defined as follows:

```java
public void drawString(String text, int x, int y, int anchor);
```

This method draws text in the current color, using the current font with its anchor point at (x,y). The definition of the anchor point must be one of the horizontal constants (LEFT, HCENTER, RIGHT) combined with one of the vertical constants (TOP, BASELINE, BOTTOM) using the bit-wise OR operator. Zero may also be used as the value of an anchor point. Using zero for the anchor point value gives results identical to using TOP | LEFT.

Vertical centering of the text is not specified since it is not considered useful, it is hard to specify, and it is burdensome to implement. Thus, the VCENTER value is not allowed in the anchor point parameter of text drawing calls.

The actual position of the bounding box of the text relative to the (x, y) location is determined by the anchor point. These anchor points occur at named locations along the outer edge of the bounding box. Thus, if f is g's current font (as returned by `g.getFont()`), the following calls will all have identical results:

```java
g.drawString(str, x, y, TOP | LEFT);
g.drawString(str, x + f.stringWidth(str) / 2, y, TOP | HCENTER);
g.drawString(str, x + f.stringWidth(str), y, TOP | RIGHT);

g.drawString(str, x, y + f.getBaselinePosition(), BASELINE | LEFT);
g.drawString(str, x + f.stringWidth(str) / 2, y + f.getBaselinePosition(), BASELINE | HCENTER);
g.drawString(str, x + f.stringWidth(str), y + f.getBaselinePosition(), BASELINE | RIGHT);

drawString(str, x, y + f.getHeight(), BOTTOM | LEFT);
drawString(str, x + f.stringWidth(str) / 2, y + f.getHeight(), BOTTOM | HCENTER);
drawString(str, x + f.stringWidth(str), y + f.getHeight(), BOTTOM | RIGHT);
```

For text drawing, the inter-character and inter-line spacing (leading) specified by the font designer are included as part of the values returned in the `stringWidth()` and `getHeight()` calls of class `Font`. For example, given the following code:
Code fragments (5) and (6) behave similarly if not identically. This occurs because \texttt{f.stringWidth()} includes the inter-character spacing. The exact spacing may differ between these calls if the system supports font kerning and support for bi-directional text rendering.

Similarly, reasonable vertical spacing may be achieved simply by adding the font height to the Y-position of subsequent lines. For example:

```
// (5)  
g.drawString(string1 + string2, x, y, TOP | LEFT);
// (6)  
g.drawString(string1, x, y, TOP | LEFT);  
g.drawString(string2, x + f.stringWidth(string1), y, TOP | LEFT);
```

draws \texttt{string1} and \texttt{string2} on separate lines with an appropriate amount of inter-line spacing.

The \texttt{stringWidth()} of the string and the \texttt{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 \textit{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 positioned at the \((x, y)\) location given in the graphics request. Unlike text, vertical centering of images is well-defined, and thus the \texttt{VCENTER} value may be used within the anchor point parameter of image drawing requests. Because images have no notion of a baseline, the \texttt{BASELINE} value may not be used within the anchor point parameter of image drawing requests.

## Pixel Formats

The Graphics and Image classes include APIs that support the rendering and retrieval of specific pixel values. To maximize the portability of applications, the pixel values are represented using a set of four pixel formats. While the device may use other pixel formats natively, only these formats are supported via the MIDP APIs.

The use of 16-bit formats reduces memory consumption since only two bytes are needed per pixel, but due to the lower number of distinct levels, a smaller set of unique pixel values can be represented. This shortcoming will be most noticeable on displays that support higher color depths. Conversely, 32-bit formats require twice as much memory since four bytes are needed per pixel, but the resulting color depth exceeds that of most mobile device displays.

Unless the device happens to use the same pixel format natively, some conversion will occur when rendering or retrieving pixel values; the computing overhead associated with this conversion will depend on specific formats involved.

### 32-bit ARGB
This format uses the `int` type to encode both color and opacity information for a single pixel. Eight bits are used to encode the alpha value, red component, green component, and blue component. For each color component, a value of 0xFF represents maximum intensity and a value of 0x00 represents minimum intensity. For the alpha value, a value of 0xFF represents full opacity and a value of 0x00 represents full transparency.

### 24-bit RGB

This format is encoded exactly the same as the 32-bit ARGB, with the exception of the alpha value. Full opacity is assumed since this format does not include an alpha value, and the contents of the upper byte are ignored.

### 16-bit ARGB

This format uses the `char` type to encode both color and opacity information for a single pixel. Four bits are used to encode the alpha value, red component, green component, and blue component. For each color component, a value of 0xF represents maximum intensity and a value of 0x00 represents minimum intensity. For the alpha value, a value of 0xF represents full opacity and a value of 0x00 represents full transparency.

### 16-bit RGB

This format uses the `char` type to encode color information for a single pixel. 5 bits are used to encode the red component and blue component, for which a value of 0x1F represents maximum intensity and a value of 0x00 represents minimum intensity. 6 bits are used to encode the green component, for which a value of 0x3F represents maximum intensity and a value of 0x00 represents minimum intensity. Full opacity is assumed since this format does not include an alpha value.

**Reference**

Porter-Duff


**Since:** MIDP 1.0
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clipRect</td>
<td>Intersects the current clip with the specified rectangle.</td>
</tr>
</tbody>
</table>

### Constants

- **BASELINE**
  - Position anchor point at baseline of text.
  - Value: 64

- **BOTTOM**
  - Position anchor point of text and images below text or image.
  - Value: 32

- **DOTTED**
  - Constant for the DOTTED stroke style.
  - Value: 1

- **HCENTER**
  - Center text and images horizontally around anchor point.
  - Value: 1

- **LEFT**
  - Position anchor point of text and images to the left of text or image.
  - Value: 4

- **RIGHT**
  - Position anchor point of text and images to the right of text or image.
  - Value: 8

- **SOLID**
  - Constant for the SOLID stroke style.
  - Value: 0

- **SRC**
  - Constant for the SRC blending mode.
  - Value: 1

- **SRC_OVER**
  - Constant for the SRC_OVER blending mode.
  - Value: 0

- **TOP**
  - Position anchor point of text and images above text or image.
  - Value: 16

- **VCENTER**
  - Center images vertically around anchor point.
  - Value: 2
<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>copyArea(int x_src, int y_src, int width, int height, int x_dest, int y_dest, int anchor)</td>
<td>Copies the contents of a rectangular area ((x_{\text{src}}, y_{\text{src}}, width, height)) to a destination area, whose anchor point identified by anchor is located at ((x_{\text{dest}}, y_{\text{dest}})).</td>
</tr>
<tr>
<td>void</td>
<td>drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)</td>
<td>Draws the outline of a circular or elliptical arc covering the specified rectangle, using the current color, alpha, and stroke style.</td>
</tr>
<tr>
<td>void</td>
<td>drawARGB16(short[] argbData, int offset, int scanlength, int x, int y, int width, int height)</td>
<td>Renders a series of device-independent ARGB values in a specified region.</td>
</tr>
<tr>
<td>void</td>
<td>drawChar(char character, int x, int y, int anchor)</td>
<td>Draws the specified character using the current font, color, and alpha.</td>
</tr>
<tr>
<td>void</td>
<td>drawChars(char[] data, int offset, int length, int x, int y, int anchor)</td>
<td>Draws the specified characters using the current font, color, and alpha.</td>
</tr>
<tr>
<td>void</td>
<td>drawImage(Image img, int x, int y, int anchor)</td>
<td>Draws the specified image by using the anchor point.</td>
</tr>
<tr>
<td>void</td>
<td>drawLine(int x1, int y1, int x2, int y2)</td>
<td>Draws a line between the coordinates ((x_1,y_1)) and ((x_2,y_2)) using the current color, alpha, and stroke style.</td>
</tr>
<tr>
<td>void</td>
<td>drawRect(int x, int y, int width, int height)</td>
<td>Draws the outline of the specified rectangle using the current color, alpha, and stroke style.</td>
</tr>
<tr>
<td>void</td>
<td>drawRegion(Image src, int x_src, int y_src, int width, int height, int transform, int x_dest, int y_dest, int anchor)</td>
<td>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.</td>
</tr>
<tr>
<td>void</td>
<td>drawRegion(Image src, int x_src, int y_src, int width, int height, int transform, int x_dest, int y_dest, int anchor, int width_dest, int height_dest)</td>
<td>Scales and transforms a region of the specified source image to a region within the destination, possibly transforming (rotating and reflecting) the image data using the chosen transform function and scaling the pixels to fit the destination region.</td>
</tr>
<tr>
<td>void</td>
<td>drawRGB(int[] rgbData, int offset, int scanlength, int x, int y, int width, int height, boolean processAlpha)</td>
<td>Renders a series of device-independent ARGB values in a specified region.</td>
</tr>
<tr>
<td>void</td>
<td>drawRGB16(short[] rgbData, int offset, int scanlength, int x, int y, int width, int height)</td>
<td>Renders a series of device-independent RGB values in a specified region.</td>
</tr>
<tr>
<td>void</td>
<td>drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)</td>
<td>Draws the outline of the specified rounded corner rectangle using the current color, alpha, and stroke style.</td>
</tr>
<tr>
<td>void</td>
<td>drawString(String str, int x, int y, int anchor)</td>
<td>Draws the specified String using the current font, color, and alpha.</td>
</tr>
<tr>
<td>Method</td>
<td>Parameters</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>void drawSubstring(String str, int offset, int len, int x, int y, int anchor)</code></td>
<td>Draws the specified String using the current font, color, and alpha.</td>
<td></td>
</tr>
<tr>
<td><code>void drawText(Text text, int x, int y)</code></td>
<td>Draws a Text object to the Graphics context at the requested location.</td>
<td></td>
</tr>
<tr>
<td><code>void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)</code></td>
<td>Fills a circular or elliptical arc covering the specified rectangle using the current color and alpha.</td>
<td></td>
</tr>
<tr>
<td><code>void fillRect(int x, int y, int width, int height)</code></td>
<td>Fills the specified rectangle with the current color and alpha.</td>
<td></td>
</tr>
<tr>
<td><code>void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)</code></td>
<td>Fills the specified rounded corner rectangle with the current color and alpha.</td>
<td></td>
</tr>
<tr>
<td><code>void fillTriangle(int x1, int y1, int x2, int y2, int x3, int y3)</code></td>
<td>Fills the specified triangle with the current color and alpha level.</td>
<td></td>
</tr>
<tr>
<td><code>int getAlpha()</code></td>
<td>Gets the current alpha value.</td>
<td></td>
</tr>
<tr>
<td><code>int getAlphaColor()</code></td>
<td>Gets the current drawing color and alpha value.</td>
<td></td>
</tr>
<tr>
<td><code>int getBlendingMode()</code></td>
<td>Gets the current blending mode for this Graphics object.</td>
<td></td>
</tr>
<tr>
<td><code>int getBlueComponent()</code></td>
<td>Gets the blue component of the current color.</td>
<td></td>
</tr>
<tr>
<td><code>int getClipHeight()</code></td>
<td>Gets the height of the current clipping area.</td>
<td></td>
</tr>
<tr>
<td><code>int getClipWidth()</code></td>
<td>Gets the width of the current clipping area.</td>
<td></td>
</tr>
<tr>
<td><code>int getClipX()</code></td>
<td>Gets the X offset of the current clipping area, relative to the coordinate system origin of this graphics context.</td>
<td></td>
</tr>
<tr>
<td><code>int getClipY()</code></td>
<td>Gets the Y offset of the current clipping area, relative to the coordinate system origin of this graphics context.</td>
<td></td>
</tr>
<tr>
<td><code>int getColor()</code></td>
<td>Gets the current drawing color.</td>
<td></td>
</tr>
<tr>
<td><code>int getDisplayColor(int color)</code></td>
<td>Gets the color that will be displayed if the specified color is requested.</td>
<td></td>
</tr>
<tr>
<td><code>javax.microedition.lcdui.Font getFont()</code></td>
<td>Gets the current font.</td>
<td></td>
</tr>
<tr>
<td><code>int getGrayScale()</code></td>
<td>Gets the current grayscale value of the color being used for rendering operations.</td>
<td></td>
</tr>
</tbody>
</table>
### javax.microedition.lcdui.Graphics

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int getGreenComponent()</td>
<td>Gets the green component of the current color.</td>
</tr>
<tr>
<td>int getRedComponent()</td>
<td>Gets the red component of the current color.</td>
</tr>
<tr>
<td>int getStrokeStyle()</td>
<td>Gets the stroke style used for drawing operations.</td>
</tr>
<tr>
<td>int getTranslateX()</td>
<td>Gets the X coordinate of the translated origin of this graphics context.</td>
</tr>
<tr>
<td>int getTranslateY()</td>
<td>Gets the Y coordinate of the translated origin of this graphics context.</td>
</tr>
<tr>
<td>void setAlpha(int alpha)</td>
<td>Sets the alpha value for this Graphics object.</td>
</tr>
<tr>
<td>void setAlphaColor(int ARGB)</td>
<td>Sets the current color and alpha to the specified 32-bit ARGB value.</td>
</tr>
<tr>
<td>void setAlphaColor(int alpha, int red, int green, int blue)</td>
<td>Sets the current color and alpha to the specified values.</td>
</tr>
<tr>
<td>void setBlendingMode(int mode)</td>
<td>Sets the current blending mode for this Graphics object.</td>
</tr>
<tr>
<td>void setClip(int x, int y, int width, int height)</td>
<td>Sets the current clip to the rectangle specified by the given coordinates.</td>
</tr>
<tr>
<td>void setColor(int RGB)</td>
<td>Sets the current color to the specified 24-bit RGB value.</td>
</tr>
<tr>
<td>void setColor(int red, int green, int blue)</td>
<td>Sets the current color to the specified RGB values.</td>
</tr>
<tr>
<td>void setFont(Font font)</td>
<td>Sets the font for all subsequent text rendering operations.</td>
</tr>
<tr>
<td>void setGrayScale(int value)</td>
<td>Sets the current grayscale to be used for all subsequent rendering operations.</td>
</tr>
<tr>
<td>void setStrokeStyle(int style)</td>
<td>Sets the stroke style used for drawing lines, arcs, rectangles, and rounded rectangles.</td>
</tr>
<tr>
<td>void translate(int x, int y)</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**

```java
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**

```java
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**

```java
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**

```java
public static final int SOLID
```

Constant for the SOLID stroke style.

- Value 0 is assigned to SOLID.
Since: MIDP 3.0

### SRC

```java
public static final int SRC
```

Constant for the `SRC` blending mode. The destination's color and alpha value are replaced with those of the source.

Value 1 is assigned to `SRC`.
Constant value: 1

Since: MIDP 3.0

### SRC_OVER

```java
public static final int SRC_OVER
```

Constant for the `SRC_OVER` blending mode. The source is composited over the destination.

Value 0 is assigned to `SRC_OVER`.
Constant value: 0

Since: MIDP 3.0

### TOP

```java
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

```java
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

```java
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:** `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 \((x_{\text{src}}, y_{\text{src}}, \text{width}, \text{height})\) to a destination area, whose anchor point identified by anchor is located at \((x_{\text{dest}}, y_{\text{dest}})\). 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 \((x_{\text{src}}, y_{\text{src}})\) and \((x_{\text{dest}}, y_{\text{dest}})\) 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 \(\text{getTranslateX()}\) and \(\text{getTranslateY()}\), 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 an actual display device. This restriction is necessary because allowing a copyArea method on the display would adversely impact certain techniques for implementing double-buffering.

Copying is performed using direct pixel replacement. That is, both the color and the alpha value of the destination pixel are replaced with those of the source pixel. The color and alpha value of the Graphics object have no impact on the copy operation.

Parameters:
- \(x_{\text{src}}\) - the x coordinate of upper left corner of source area
- \(y_{\text{src}}\) - the y coordinate of upper left corner of source area
- \(\text{width}\) - the width of the source area
- \(\text{height}\) - the height of the source area
- \(x_{\text{dest}}\) - the x coordinate of the destination anchor point
- \(y_{\text{dest}}\) - the y coordinate of the destination anchor point
- \(\text{anchor}\) - the anchor point for positioning the region within the destination image

Throws:
- \(\text{IllegalStateException}\) - if the destination of this Graphics object is a display device
- \(\text{IllegalArgumentException}\) - if the region to be copied exceeds the bounds of the source image

Since: MIDP 2.0
**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, alpha, and stroke style.

The resulting arc begins at **startAngle** and extends for **arcAngle** degrees, using the current color and alpha. Angles are interpreted such that 0 degrees is at the 3 o'clock position. A positive value indicates a counter-clockwise rotation while a negative value indicates a clockwise rotation.

The center of the arc is the center of the rectangle whose origin is (x, y) and whose size is specified by the **width** and **height** arguments.

The resulting arc covers an area **width + 1** pixels wide by **height + 1** pixels tall. If either **width** or **height** is less than zero, nothing is drawn.

The angles are specified relative to the non-square extents of the bounding rectangle such that 45 degrees always falls on the line from the center of the ellipse to the upper right corner of the bounding rectangle. As a result, if the bounding rectangle is noticeably longer in one axis than the other, the angles to the start and end of the arc segment will be skewed farther along the longer axis of the bounds.

**Parameters:**
- x - the x coordinate of the upper-left corner of the arc to be drawn
- y - the y coordinate of the upper-left corner of the arc to be drawn
- width - the width of the arc to be drawn
- height - the height of the arc to be drawn
- startAngle - the beginning angle
- arcAngle - the angular extent of the arc, relative to the start angle

**See Also:** **fillArc(int, int, int, int, int, int)**

---

**drawARGB16**

public void **drawARGB16**(short[] argbData,
        int offset,
        int scanlength,
        int x,
        int y,
        int width,
        int height)


Renders a series of device-independent ARGB values in a specified region. The values are stored in the `argbData` char array in a 16-bit ARGB format, 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 `argbData` 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:

\[
P(a, b) = \text{argbData}[\text{offset} + (a - x) + (b - y) \times \text{scanlength}]
\]

for

\[
x \leq a < x + \text{width} \\
y \leq b < y + \text{height}
\]

If either `width` or `height` is zero or less, no exception is thrown, and nothing is drawn.

The alpha value of each pixel in the array is multiplied by the alpha value of the `Graphics` object to determine its effective opacity when rendered.

**Parameters:**
- `argbData` - an array of 16-bit ARGB values
- `offset` - the array index of the first ARGB value
- `scanlength` - the relative array offset between the corresponding pixels in consecutive rows in the `argbData` 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

**Throws:**
- `ArrayIndexOutOfBoundsException` - if the requested operation will attempt to access an element of `argbData` whose index is either negative or beyond its length
- `NullPointerException` - if `argbData` is `null`

**Since:** MIDP 3.0

---

**drawChar**

```java
public void drawChar(char character,
                     int x,  
                     int y, 
                     int anchor)
```

Draws the specified character using the current font, color, and alpha.

**Parameters:**
- `character` - the character to be drawn
- `x` - the x coordinate of the anchor point
- `y` - the y coordinate of the anchor point
- `anchor` - the anchor point for positioning the text; see anchor points

**Throws:**

---
IllegalArgumentException - if anchor is not a legal value

See Also: drawString(String, int, int, int), drawChars(char[], int, int, int, int)

---

drawChars

public void drawChars(char[] data, int offset, int length, int x, int y, int anchor)

Draws the specified characters using the current font, color, and alpha.

The offset and length parameters must specify a valid range of characters within the character array data. The offset parameter must be within the range [0..(data.length)], inclusive. The length parameter must be a non-negative integer such that (offset + length) <= data.length.

Parameters:
- data - the array of characters to be drawn
- offset - the start offset in the data
- length - the number of characters to be drawn
- x - the x coordinate of the anchor point
- y - the y coordinate of the anchor point
- anchor - the anchor point for positioning the text; see anchor points

Throws:
- ArrayIndexOutOfBoundsException - if offset and length do not specify a valid range within the data array
- IllegalArgumentException - if anchor is not a legal value
- NullPointerException - if data is null

See Also: drawString(String, int, int, int)

---

drawImage

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 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.

The alpha value of each pixel in the image is multiplied by the alpha value of the Graphics object to determine its effective opacity when rendered.

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
class javax.microedition.lcdui.Graphics

public 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, alpha, and stroke style.

**Parameters:**
- \(x_1\) - the x coordinate of the start of the line
- \(y_1\) - the y coordinate of the start of the line
- \(x_2\) - the x coordinate of the end of the line
- \(y_2\) - the y coordinate of the end of the line

---

**drawRect**

```java
class javax.microedition.lcdui.Graphics

public void drawRect(int x, int y, int width, int height)
```

Draws the outline of the specified rectangle using the current color, alpha, 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:**
- \(x\) - the x coordinate of the rectangle to be drawn
- \(y\) - the y coordinate of the rectangle to be drawn
- \(\text{width}\) - the width of the rectangle to be drawn
- \(\text{height}\) - the height of the rectangle to be drawn

**See Also:** `fillRect(int, int, int, int)`

---

**drawRegion**

```java
class javax.microedition.lcdui.Graphics

public 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.

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.

The \((x, y)\) coordinates are relative to the upper left corner of the source image. The \(x, y, width,\) 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:

\[
\begin{align*}
x &\geq 0 \\
y &\geq 0 \\
x + width &\leq source width \\
y + height &\leq source height
\end{align*}
\]

The \((x, y)\) 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 \((x, y)\) in the destination.

The alpha value of each pixel in the image is multiplied by the alpha value of the Graphics object to determine its effective opacity when rendered.

Parameters:
- \(\text{src}\) - the source image to copy from
- \(x, y\) - the \(x, y\) coordinate of the upper left corner of the region within the source image to copy
- \(width, height\) - the width and height of the region to copy
- \(\text{transform}\) - the desired transformation for the selected region being copied
- \(x, y\) - the \(x, y\) coordinate of the anchor point in the destination drawing area
- \(\text{anchor}\) - the anchor point for positioning the region within the destination image

Throws:
- IllegalArgumentException - if \(\text{src}\) is the same image as the destination of this Graphics object
- NullPointerException - if \(\text{src}\) is \(null\)
- IllegalArgumentException - if \(\text{transform}\) is invalid
- IllegalArgumentException - if \(\text{anchor}\) is invalid
- IllegalArgumentException - if the region to be copied exceeds the bounds of the source image
drawRegion

public void drawRegion(Image src,  
   int x_src,  
   int y_src,  
   int width,  
   int height,  
   int transform,  
   int x_dest,  
   int y_dest,  
   int anchor,  
   int width_dest,  
   int height_dest)

Scales and transforms a region of the specified source image to a region within the destination, possibly transforming (rotating and reflecting) the image data using the chosen transform function and scaling the pixels to fit the destination region.

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. If either width_dest or height_dest is zero or less, nothing is drawn.

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 (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:

\[
\begin{align*}
   x_{\text{src}} &\geq 0 \\
   y_{\text{src}} &\geq 0 \\
   x_{\text{src}} + \text{width} &\leq \text{source width} \\
   y_{\text{src}} + \text{height} &\leq \text{source height}
\end{align*}
\]

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 scaling and 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.

The alpha value of each pixel in the image is multiplied by the alpha value of the Graphics object to determine its effective opacity when rendered.
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 source region
   height - the height of the source region
   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
   width_dest - the width of the region in the destination drawing area
   height_dest - the height of the region in the destination drawing area

Throws:
   IllegalArgumentException - if src is the same image as the destination of this Graphics object
   NullPointerException - if src is null
   IllegalArgumentException - if transform is invalid
   IllegalArgumentException - if anchor is invalid
   IllegalArgumentException - if the region to be copied exceeds the bounds of the source image

Since: MIDP 3.0

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 ARGB values in a specified region. The values are stored in the `rgbData` array using the 32-bit ARGB or 24-bit RGB format. 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:

\[
P(a, b) = \text{rgbData}[\text{offset} + (a - x) + (b - y) \times \text{scanlength}]
\]

for

\[
x \leq a < x + \text{width}\\
y \leq b < y + \text{height}
\]

If either width or height is zero or less, no exception is thrown, and nothing is drawn.

If `processAlpha` is `true`, the pixel values are assumed to be in the 32-bit RGB format and high-order byte specifies opacity. The alpha value of each pixel in the array is multiplied by the alpha value of the `Graphics` object to determine its effective opacity when rendered.

If `processAlpha` is `false`, the pixel values are assumed to be in the 24-bit RGB format. The value in the high-order byte is ignored and all pixels are rendered with the alpha value of the `Graphics` object.

**Parameters:**
- `rgbData` - an array of 32-bit ARGB or 24-bit RGB values
- `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 the pixel values should be processed assuming a 32-bit ARGB format, `false` if the pixel values should be processed assuming a 24-bit RGB format

**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

`drawRGB16`

```java
public void drawRGB16(short[] rgbData,
                      int offset,
                      int scanlength,
                      int x,
                      int y,
                      int width,
                      int height)
```
Renders a series of device-independent RGB values in a specified region. The values are stored in the rgbData char array in a 16-bit RGB format, 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 RGB data is rasterized horizontally from left to right within each row. The RGB 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:

$$P(a, b) = \text{rgbData}[\text{offset} + (a - x) + (b - y) \times \text{scanlength}]$$

for

$$x \leq a < x + \text{width}$$
$$y \leq b < y + \text{height}$$

If either width or height is zero or less, no exception is thrown, and nothing is drawn.

The alpha value of the Graphics object determines the effective opacity of the pixel values when rendered.

**Parameters:**
- rgbData - an array of 16-bit RGB values
- offset - the array index of the first RGB 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

**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 3.0

**drawRoundRect**

```java
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, alpha, and stroke style. The resulting rectangle will cover an area $(width + 1)$ pixels wide by $(height + 1)$ pixels tall. If either width or height is less than zero, nothing is drawn.

**Parameters:**
- x - the x coordinate of the rectangle to be drawn
- y - the y coordinate of the rectangle to be drawn
width - the width of the rectangle to be drawn
height - the height of the rectangle to be drawn
arcWidth - the horizontal diameter of the arc at the four corners
arcHeight - the vertical diameter of the arc at the four corners

See Also: fillRoundRect(int, int, int, int, int, int)

---

drawString

public void drawString(String str, int x, int y, int anchor)

Draws the specified String using the current font, color, and alpha. The x,y position is the position of the anchor point. See anchor points.

Parameters:
str - the String to be drawn
x - the x coordinate of the anchor point
y - the y coordinate of the anchor point
anchor - the anchor point for positioning the text

Throws:
NullPointerException - if str is null
IllegalArgumentException - if anchor is not a legal value

See Also: drawChars(char[], int, int, int, int, int)

---

drawSubstring

public void drawSubstring(String str, int offset, int len, int x, int y, int anchor)

Draws the specified String using the current font, color, and alpha. The x,y position is the position of the anchor point. See anchor points.

The offset and len parameters must specify a valid range of characters within the string str. The offset parameter must be within the range [0..(str.length())], inclusive. The len parameter must be a non-negative integer such that (offset + len) <= str.length().

Parameters:
str - the String to be drawn
offset - zero-based index of first character in the substring
len - length of the substring
x - the x coordinate of the anchor point
y - the y coordinate of the anchor point
anchor - the anchor point for positioning the text

Throws:
StringIndexOutOfBoundsException - if offset and length do not specify a valid range within the String str
IllegalArgumentException - if anchor is not a legal value
NullPointerException - if str is null

See Also: drawString(String, int, int, int)

---

drawText

public void drawText(Text text, int x, int y)

Draw a Text object to the Graphics context at the requested location. The Text object contains the characters to be drawn and the location and size of the bounding box. The fonts and color of each character is set in the Text class. The alpha value of the Graphics object is uniformly applied to the entire text and it is rendered according to the current blending mode.

**Parameters:**
- text - the Text object to draw.
  - x - the x offset of the upper left corner of the text bounding box
  - y - the y offset of the upper left corner of the text bounding box

**Since:** MIDP 3.0

---

### fillArc

```java
class javax.microedition.lcdui.Graphics

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 using the current color and alpha.

The resulting arc begins at `startAngle` and extends for `arcAngle` degrees. Angles are interpreted such that 0 degrees is at the 3 o'clock position. A positive value indicates a counter-clockwise rotation while a negative value indicates a clockwise rotation.

The center of the arc is the center of the rectangle whose origin is `(x, y)` and whose size is specified by the `width` and `height` arguments.

If either `width` or `height` is zero or less, nothing is drawn.

The filled region consists of the “pie wedge” region bounded by the arc segment as if drawn by `drawArc()`, the radius extending from the center to this arc at `startAngle` degrees, and radius extending from the center to this arc at `startAngle + arcAngle` degrees.

The angles are specified relative to the non-square extents of the bounding rectangle such that 45 degrees always falls on the line from the center of the ellipse to the upper right corner of the bounding rectangle. As a result, if the bounding rectangle is noticeably longer in one axis than the other, the angles to the start and end of the arc segment will be skewed farther along the longer axis of the bounds.

**Parameters:**
- x - the x coordinate of the upper-left corner of the arc to be filled.
- y - the y coordinate of the upper-left corner of the arc to be filled.
- width - the width of the arc to be filled
- height - the height of the arc to be filled
- startAngle - the beginning angle.
- arcAngle - the angular extent of the arc, relative to the start angle.

**See Also:** `drawArc(int, int, int, int, int, int)`

---

### fillRect

```java
class javax.microedition.lcdui.Graphics

public void fillRect(int x, int y, int width, int height)
```

Fills the specified rectangle with the current color and alpha. Nothing is drawn if either width or height is zero or less.

**Parameters:**
- x - the x coordinate of the rectangle to be filled.
- y - the y coordinate of the rectangle to be filled.
- width - the width of the rectangle to be filled
- height - the height of the rectangle to be filled

**See Also:** `drawRect(int, int, int, int)`
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 and alpha. If either width or height is zero or less, nothing is drawn.

Parameters:
   x - the x coordinate of the rectangle to be filled
   y - the y coordinate of the rectangle to be filled
   width - the width of the rectangle to be filled
   height - the height of the rectangle to be filled
   arcWidth - the horizontal diameter of the arc at the four corners
   arcHeight - the vertical diameter of the arc at the four corners

See Also: drawRoundRect(int, int, int, int, int, int)

fillTriangle

public void fillTriangle(int x1,
   int y1,
   int x2,
   int y2,
   int x3,
   int y3)

Fills the specified triangle will the current color and alpha level. 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

getAlpha

public int getAlpha()

Gets the current alpha value.

An alpha value of 255 is fully opaque, and a value of 0 is fully transparent. The alpha value is 255 (fully opaque) by default.

Returns:
   the alpha value (0-255)

Since: MIDP 3.0

getAlphaColor

public int getAlphaColor()

Gets the current drawing color and alpha value.

Returns:
   the current color and alpha value encoded using the 32-bit ARGB format
See Also: `setColor(int, int, int)`

### `getBlendingMode`

`public int getBlendingMode()`

Gets the current blending mode for this Graphics object.

**Returns:**
- the current blending mode (`SRC` or `SRC_OVER`)

See Also: `setBlendingMode(int)`

Since: MIDP 3.0

See Also: `setColor(int, int, int)`

### `getBlueComponent`

`public int getBlueComponent()`

Gets the blue component of the current color.

**Returns:**
- integer value in range 0-255

See Also: `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: `clipRect(int, int, int, int), 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: `clipRect(int, int, int, int), 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: `clipRect(int, int, int, int), 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: `clipRect(int, int, int, int), setClip(int, int, int, int)`

---

**getColor**

```java
public int getColor()
```

Gets the current drawing color. This method does not return the alpha value of the Graphics object, and the upper byte of the return value will always contain a value of 0x00.

Returns:
the current color encoded using the 24-bit RGB format

See Also: `setColor(int, int, int)`

---

**getDisplayColor**

```java
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 0xFFFFFFFF (white) or 0x000000 (black) depending on the brightness of the specified color.

Parameters:
color - the desired color encoded in the 24-bit RGB format

Returns:
the corresponding color that will be displayed on the device's screen encoded in the 24-bit RGB format

Since: MIDP 2.0

---

**getFont**

```java
public javax.microedition.lcdui.Font getFont()
```

Gets the current font.

Returns:
current font

See Also: `Font, 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: `setGrayScale(int)`
java.awt.Graphics

getGreenComponent

public int getGreenComponent()

Gets the green component of the current color.

Returns:
integer value in range 0-255

See Also: setColor(int, int, int)

getRedComponent

public int getRedComponent()

Gets the red component of the current color.

Returns:
integer value in range 0-255

See Also: setColor(int, int, int)

getStrokeStyle

public int getStrokeStyle()

Gets the stroke style used for drawing operations.

Returns:
stroke style, SOLID or DOTTED

See Also: setStrokeStyle(int)

getTranslateX

public int getTranslateX()

Gets the X coordinate of the translated origin of this graphics context.

Returns:
X of current origin

getTranslateY

public int getTranslateY()

Gets the Y coordinate of the translated origin of this graphics context.

Returns:
Y of current origin

setAlpha

public void setAlpha(int alpha)

Sets the alpha value for this Graphics object. All operations performed using this Graphics object will be rendered with the current alpha level using the selected blending mode. The red, green, and blue components of the current color are unaffected.

An alpha value of 255 is fully opaque, and a value of 0 is fully transparent. The alpha value is 255 (fully opaque) by default.

Parameters:
alpha - the new alpha value for this Graphics object

Throws:
IllegalArgumentException - if the alpha value is outside of range 0-255

Since: MIDP 3.0

**setAlphaColor**

```java
public void setAlphaColor(int ARGB)
```

Sets the current color and alpha to the specified 32-bit ARGB value. All subsequent rendering operations will use this specified color and alpha value.

**Parameters:**

- ARGB - the new drawing color and alpha value encoded using the 32-bit ARGB format

See Also: `getColor()`

Since: MIDP 3.0

**setAlphaColor**

```java
public void setAlphaColor(int alpha,
                          int red,
                          int green,
                          int blue)
```

Sets the current color and alpha to the specified values. All subsequent rendering operations will use this specified color and alpha value.

**Parameters:**

- alpha - the alpha component of the color being set in range 0-255
- 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 parameter is outside of the range 0-255

See Also: `getColor()`

Since: MIDP 3.0

**setBlendingMode**

```java
public void setBlendingMode(int mode)
```

Sets the current blending mode for this Graphics object. The blending mode dictates how rendered pixels are combined with the destination pixels.

If **SRC_OVER** is used, the source pixel is blended on top of the destination pixel.

If **SRC** is used, the destination pixel is fully replaced with the source pixel, including the source pixel's alpha value. This mode can only be used on a Graphics object that renders to an Image with an alpha channel, as determined by calling `Image.hasAlpha`

The **SRC_OVER** blending mode is used by default.

**Parameters:**

- mode - the desired blending mode (**SRC** or **SRC_OVER**)  

**Throws:**

- IllegalArgumentException - if mode is not a valid value
- IllegalArgumentException - if the SRC mode is requested on a Graphics object that renders to a surface without an alpha channel

See Also: `getBlendingMode()`

Since: MIDP 3.0
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: clipRect(int, int, int, int)

setColor

public void setColor(int RGB)

Sets the current color to the specified 24-bit RGB value. All subsequent rendering operations will use this specified color.

Note that this method only changes the drawing color. The upper byte of the RGB value is ignored and the alpha value of the Graphics object is unaffected by this method.

Parameters:
  RGB - the new drawing color encoded using the 24-bit RGB format

See Also: 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: 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, getFont(), drawString(String, int, int, int), drawChars(char[], 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:** 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:** 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 coordinates passed to this method are interpreted relative to the current translated origin, and thus the effect of calls to translate() are cumulative. For example, calling translate(1, 2) and then translate(3, 4) has the same effect as calling translate(4, 6).

The application can set the origin in terms of absolute coordinates (ax, ay) using the following technique:

```java
g.translate(ax - g.getTranslateX(), ay - g.getTranslateY())
```

**Parameters:**
- x - the x coordinate of the new translation origin
- y - the y coordinate of the new translation origin

**See Also:** getTranslateX(), getTranslateY()
IdleItem

Declaration

public abstract class IdleItem extends CustomItem

Object
--- javax.microedition.lcdui.Item
    ++ javax.microedition.lcdui.CustomItem
    ++ javax.microedition.lcdui.IdleItem

Description

This class represents a dedicated UI component that can be used to render content to the idle screen.

Relationship to CustomItem

IdleItem is defined as a subclass of the CustomItem. The behavior of the IdleItem is the same as in CustomItem. The CustomItem is used because it provides rendering power and flexibility of the Canvas component and supports the concept of being embedded in a broader UI context. This provides support for negotiating size and location of content, and includes well-defined notions of focus traversal, both within the component and within the broader UI. These ideas are fundamental to content within an idle screen.

The two new methods added in this class provide the additional functionality needed in idle screen items. All other functionality is the same as in CustomItem.

As a subclass of CustomItem, IdleItem has no constrains to what kinds of content can be displayed in it. The MIDlet using IdleItem has just as much control over rendering as it has within a Canvas. Two additional methods are defined to allow greater ability to control the appearance of the IdleItem. Also, the semantics of the paint method are refined to allow MIDlet content to appear over the top of any user-selected wallpaper or background image in the idle screen.

Position and Size

The platform on the device decides how the IdleItem is positioned on the idle screen. However, it is important that IdleItem objects are able to request the amount of screen space they require, and to learn what space they have been provided. CustomItem defines a comprehensive set of methods for indicating the minimum and preferred sizes of items, as well as a means to determine what size they have available to them (via paint). All of these methods are directly applicable to IdleItem, and operate in just the same way as for CustomItem.

Focus, Traversal and Interaction Modes

CustomItem defines a sophisticated model for focus, internal traversal and scrolling in form items. This model is directly relevant to the IdleItem, and applies without modification.

A platform indicates what traversal is or is not possible on the idle screen through getInteractionModes method. This allows a MIDlet to determine whether internal traversal, key events and pointer events are supported on IdleItem.
Commands

As a subclass of Item it is possible for a MIDlet to add commands to an IdleItem. This allows MIDlets to have full softkey support and options menus associated with content on the idle screen. When the idle screen MIDlet receives focus, the commands attached to it MUST be visible in the softkeys.

Labels

As a subclass of Item it is possible for a MIDlet to associate a label with each IdleItem. The platform is free to use this label as appropriate, or to ignore the label if labels do not normally appear on the idle screen.

Painting and Appearance

Because IdleItem is a subclass of CustomItem, the painting model automatically provides a great deal of freedom to a MIDlet. Within the area of the IdleItem the MIDlet has absolute control over what is painted and how. It is perfectly feasible for a MIDlet to use services such as JSR184 for 3D content or JSR226 for SVG content to paint the idle screen content.

In addition the IdleItem provides a number of support methods that can be used to match the basic appearance and focus conventions of the platform. These revolve around three key issues:

- Drawing focus and highlight
- Background images/wallpaper
- Fonts and Colors used on the idle screen

Drawing focus and highlight

If the IdleItem contains more than one focusable element then it typically needs a way to indicate which element is focused.

Painting the idle item

On many platforms the content on the idle screen appears over the top of a user-selected background image or wallpaper. The Java content should be able to achieve the same affect. To achieve this an application should set the paintMode to false (transparent) with method CustomItem.setPaintMode method. If paintMode is false (transparent), the application can only paint the pixels it needs and the implementation handles the other pixels. If paintMode is true, application should render every pixel on the painted area.

Fonts and Colors

Accessory methods are provided to determine appropriate fonts (Font.getFont) and colors (Display.getColor) for use on the idle screen.

Relationship to Display

The Display.setIdleItem method is provided to allow a MIDlet to set, or clear, the IdleItem used on the display. Each MIDlet may have a single IdleItem associated with each Display, which may be used when/if the Display switches to an idle state.

The following example shows how a MIDlet should request to add content to the idle screen.
Visibility and Adding/Removing from Display events

The IdleItem receives showNotify and hideNotify events to indicate when it is actually visible on the idle screen. It also receives events to say when it is actually addedToDisplay or removedFromDisplay.

Event Sequence

The relationship between these events is as follows:

1. MIDlet requests that a new IdleItem be added to a Display with setIdleItem.
2. The IdleItem is informed when/if it actually becomes a part of the idle screen of a Display through a call to the addedToDisplay method. This always occurs before calls to showNotify and traverse.
3. The IdleItem is informed that it has become visible through showNotify. This event will only ever occur after a call to addedToDisplay, and before a call to traverse.
4. The IdleItem is informed that it has become focused through traverse.
5. The IdleItem is informed that it has lost the focus through traverseOut.
6. The IdleItem is informed that it is no longer visible through hideNotify. This event will always occur after a call to removedFromDisplay.
7. The IdleItem is informed when it is removed from the idle screen through a call to the removedFromDisplay method. This always occurs after calls to traverseOut and hideNotify.

Since: MIDP 3.0

Fields inherited from class javax.microedition.lcdui.CustomItem

KEY_PRESS, KEY_RELEASE, KEY_REPEAT, NONE, POINTER_DRAG, POINTER_PRESS, POINTER_RELEASE, TRAVERSE_HORIZONTAL, TRAVERSE_VERTICAL

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 IdleItem(String label)
Constructor, so that the IdleItem can specify its label.

Method Summary

void addedToDisplay(Display display)
Indicates that this IdleItem is part of the idle screen.
<table>
<thead>
<tr>
<th>void removedFromDisplay(Display display)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indicates that this IdleItem is no longer part of the idle screen.</td>
</tr>
</tbody>
</table>

### Methods inherited from class `javax.microedition.lcdui.CustomItem`:
- `getGameAction`, `getInteractionModes`, `getKeyCode`, `getMinContentHeight`, `getMinContentWidth`, `getPrefContentHeight`, `getPrefContentWidth`, `hideNotify`, `invalidate`, `keyPressed`, `keyReleased`, `keyRepeated`, `paint`, `pointerDragged`, `pointerPressed`, `pointerReleased`, `repaint`, `repaint`, `setKeyListener`, `setPaintMode`, `showNotify`, `sizeChanged`, `traverse`, `traverseOut`

### Methods inherited from class `javax.microedition.lcdui.Item`:
- `addCommand`, `getCommands`, `getLabel`, `getLayout`, `getLayoutHint`, `getMinimumHeight`, `getMinimumWidth`, `getPreferredSizeHeight`, `getPreferredSizeWidth`, `notifyStateChanged`, `removeCommand`, `setCommand`, `setDefaultCommand`, `setItemCommandListener`, `setLabel`, `setLayout`, `setLayoutHint`, `setPreferredSize`

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

### Constructors

**IdleItem**

```java
protected IdleItem(String label)
```

Constructor, so that the IdleItem can specify its label.

This constructor copies the signature of the CustomItem constructor which allows for a label to be defined. It is up to the platform how or even if this label is used.

**Parameters:**
- `label` - the label for the IdleItem.

### Methods

**addedToDisplay**

```java
protected void addedToDisplay(Display display)
```

Indicates that this IdleItem is part of the idle screen.

`Display.setIdleItem` does not guarantee that an IdleItem will become part of the idle screen for a display, it merely requests that this happens. This method is called to inform the IdleItem that it is actually part of the idle screen.

This method does not indicate that the IdleItem is visible. Visibility is handled separately with the standard `showNotify` method.

**Parameters:**
- `display` - the Display this item was added to.
removedFromDisplay

protected void removedFromDisplay(Display display)

Indicates that this IdleItem is no longer part of the idle screen.

Called to indicate that the item is no longer part of the idle screen, either because the MIDlet or the user removed it.

Parameters:
   display - the Display this was removed from.
Image

Declaration

class Image

Object

Direct Known Subclasses:
- javax.microedition.lcdui.AnimatedImage
- javax.microedition.lcdui.ScalableImage

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.

Mutable vs. Immutable Images

Images are either mutable or immutable depending upon how they are created. Immutable images are generally created by loading image data from resource bundles, from files, or from the network. They may not be modified once created. Mutable images are created as blank images containing only white pixels. The application may render on a mutable image by calling getGraphics() on the Image to obtain a Graphics object expressly for this purpose.

Images, including AnimatedImages and ScalableImages, may be placed within Alert, Choice, Form, ImageItem, Command, List, Menu, Notification, or TabbedPane 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, ImageItem Command, List, Menu, Notification, or TabbedPane object, the effect is as if an immutable copy is taken of its current contents. This immutable copy 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);
```

If the mutable image includes alpha channel information, the following code is used instead:

```java
```

Mobile Information Device Profile v3.0 - JSR 271
© Copyright Motorola, Inc. 2002-2009. All Rights Reserved.
Alpha Channels

An Image may include an alpha channel that specifies the opacity of each pixel. Pixels may be fully opaque, fully transparent, or semi-transparent. All implementations must store alpha information with at least 4 bits of accuracy (i.e. 16 distinct levels).

An immutable Image will include an alpha channel if the resource used to create it contains an alpha channel or transparency information. A mutable Image will include an alpha channel if one was requested during instantiation.

PNG Image Format

Implementations are required to support images stored in the PNG format, as specified by the PNG (Portable Network Graphics) Specification, Version 1.2. 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, although the alpha values may be converted based on the number of alpha levels supported by the device.
- For color types 4 & 6 (grayscale with alpha and RGB with alpha, respectively) the alpha channel must be decoded and stored in the resulting image.
- 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 JARs; 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 (alpha = 0). 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, any pixels with intermediate alpha values must be carried through to the resulting image.

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

All conforming MIDP implementations MUST support ISO/IEC JPEG together with JFIF. 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].

GIF Image Format

All conforming MIDP implementations MUST support the GIF89a image format including animated variants, in accordance to the GIF89a Specification.

SVG Image Format

All conforming MIDP implementations MUST support the SVG Tiny 1.1 image format as defined in the Mobile SVG Profiles specification. Support for animation is OPTIONAL, and interactive functionality MAY be disabled.
Scalable images can be created using those static factory methods of the Image class which return immutable objects. Methods that return mutable images MUST rasterize the scalable image and return the result as a bitmap image, as described in the method documentation.

Methods that return pixel-level information (getRGB, getRGB16, getARGB16) MUST first rasterize the scalable image to its current viewport size, and the returned image MUST be based on the result of the rasterization.

References

Since: MIDP 1.0

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static</td>
<td>createImage(byte[] imageData, int imageOffset, int imageLength)</td>
</tr>
<tr>
<td>Creates an immutable image which is decoded from the data stored in the specified byte array at the specified offset and length.</td>
<td></td>
</tr>
<tr>
<td>static</td>
<td>createImage(Image source)</td>
</tr>
<tr>
<td>Creates an immutable image from a source image.</td>
<td></td>
</tr>
<tr>
<td>static</td>
<td>createImage(Image image, int x, int y, int width, int height, int transform)</td>
</tr>
<tr>
<td>Creates an immutable image using pixel data from the specified region of a source image, transformed as specified.</td>
<td></td>
</tr>
<tr>
<td>static</td>
<td>createImage(Image image, int x, int y, int width, int height, int transform, int img_width, int img_height)</td>
</tr>
<tr>
<td>Creates an immutable image using pixel data from the specified region of a source image, transformed as specified and scaled to a specific size.</td>
<td></td>
</tr>
<tr>
<td>static</td>
<td>createImage(InputStream stream)</td>
</tr>
<tr>
<td>Creates an immutable image from decoded image data obtained from an InputStream.</td>
<td></td>
</tr>
<tr>
<td>static</td>
<td>createImage(int width, int height)</td>
</tr>
<tr>
<td>Creates a new, mutable image without an alpha channel for off-screen drawing.</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>static javax.microedition.lcdui.Image</td>
<td>createImage(int width, int height, boolean withAlpha, int fillColor)</td>
</tr>
<tr>
<td></td>
<td>Creates a new mutable image with or without an alpha channel for off-screen drawing.</td>
</tr>
<tr>
<td>static javax.microedition.lcdui.Image</td>
<td>createImage(String name)</td>
</tr>
<tr>
<td></td>
<td>Creates an immutable image from decoded image data obtained from the named resource.</td>
</tr>
<tr>
<td>static javax.microedition.lcdui.Image</td>
<td>createRGBImage(int[] rgb, int width, int height, boolean processAlpha)</td>
</tr>
<tr>
<td></td>
<td>Creates an immutable image from a sequence of ARGB values, specified using the 32-bit ARGB format.</td>
</tr>
<tr>
<td>void getARGB16(short[] argbData, int offset, int scanlength, int x, int y, int width, int height)</td>
<td>Obtains ARGB pixel data from the specified region of this image and stores it in the provided array of chars.</td>
</tr>
<tr>
<td>javax.microedition.lcdui.Graphics getGraphics()</td>
<td>Creates a new Graphics object that renders to this image.</td>
</tr>
<tr>
<td>int getHeight()</td>
<td>Gets the height of the image in pixels.</td>
</tr>
<tr>
<td>void getRGB(int[] rgbData, int offset, int scanlength, int x, int y, int width, int height)</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>void getRGB16(short[] rgbData, int offset, int scanlength, int x, int y, int width, int height)</td>
<td>Obtains RGB pixel data from the specified region of this image and stores it in the provided array of shorts.</td>
</tr>
<tr>
<td>int getWidth()</td>
<td>Gets the width of the image in pixels.</td>
</tr>
<tr>
<td>boolean hasAlpha()</td>
<td>Checks if this image has alpha channel information.</td>
</tr>
<tr>
<td>boolean isAnimated()</td>
<td>Checks if this Image is an AnimatedImage comprised of several frames.</td>
</tr>
<tr>
<td>boolean isMutable()</td>
<td>Check if this image is mutable.</td>
</tr>
<tr>
<td>boolean isScalable()</td>
<td>Checks if this Image is a ScalableImage that can be rasterized at a variety of different pixel sizes.</td>
</tr>
</tbody>
</table>

Methods inherited from class Object

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

Methods
javax.microedition.lcdui.Image
createImage

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 the mandatory image formats.

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.

This method returns a object of type Image for static, bitmapped image data. An AnimatedImage is returned if the image data contains an animated bitmap image. A ScalableImage is returned if the image data contains valid vector graphics content such as SVG Tiny 1.1.

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

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 the source image contains alpha channel 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.

This method can be used to create a pixel-based Image object using a ScalableImage as the source. In this case, a new Image is returned that is a copy of the ScalableImage in its rasterized form. The dimensions of the new Image will equal the current pixel dimensions of the ScalableImage and cannot be changed.

Parameters:
- source - the source image to be copied

Returns:
- the new, immutable image

Throws:
- NullPointerException - if source is null
javax.microedition.lcdui.Image
createImage

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. Alpha channel 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.

If the source image is a ScalableImage, the returned `Image` will contain a rasterized version of the vector graphics content. The specified region is interpreted in terms of the ScalableImage's current pixel dimensions. The specified transform is applied to the resulting bitmap data.

**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 source image is not a ScalableImage or AnimatedImage;
- 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

public static javax.microedition.lcdui.Image createImage(Image image,
    int x,
    int y,
    int width,
    int height,
    int transform,
    int img_width,
    int img_height)

Creates an immutable image using pixel data from the specified region of a source image, transformed as specified and scaled to a specific size.

The source image may be mutable or immutable. If it is mutable, the new Image will represent a snapshot of the source image's contents; subsequent changes to the source image are not reflected in the new image. Alpha channel 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.

If the source image is a ScalableImage, the returned Image will contain a rasterized version of the vector graphics content. The specified region is interpreted in terms of the ScalableImage's current pixel dimensions, and the vector graphics content is rasterized directly into an image of the requested dimensions to avoid scaling artifacts. The specified transform is applied to the resulting bitmap data.

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 source image is not a ScalableImage or AnimatedImage;
- the region represents the entire source image;
- the requested image dimensions match that of the 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
- img_width - the width of the new Image
javax.microedition.lcdui.Image

createImage

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.

This method returns a object of type Image for static, bitmapped image data. An AnimatedImage is returned if the image data contains an animated bitmap image. A ScalableImage is returned if the image data contains valid vector graphics content such as SVG Tiny 1.1.

Parameters:
stream - the name of the resource containing the image data in one of the supported image formats

Returns: the created image

Throws:
NullPointerException - if stream is null
java.io.IOException - if an I/O error occurs, if the image data cannot be loaded, or if the image data cannot be decoded

Since: MIDP 2.0

createImage

public static javax.microedition.lcdui.Image createImage(int width, int height)

Creates a new, mutable image without an alpha channel for off-screen drawing. Each pixel within the newly created image is initially white and fully opaque. The width and height of the image must both be greater than zero.

This method is equivalent to calling createImage(width, height, false, 0x00FFFFFF)

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 3.0
createImage

public static javax.microedition.lcdui.Image createImage(int width, int height, boolean withAlpha, int fillColor)

Creates a new mutable image with or without an alpha channel for off-screen drawing. The width and height of the image must both be greater than zero.

If withAlpha is true, the returned Image includes an alpha channel and the opacity of each pixel can be modified using a Graphics object with the SRC blending mode.

The initial content of the pixels is determined by the fillColor parameter. If withAlpha is false, the fillColor value is interpreted as a 24-bit RGB color value (0x00RRGGBB); the value of the upper byte is ignored and the pixels are all fully opaque. If withAlpha is true, the fillColor value is interpreted as a 32-bit ARGB color value (0xAARRGGBB) and the pixels' alpha values are set according to the value of the upper byte.

Parameters:
- width - the width of the new image, in pixels
- height - the height of the new image, in pixels
- withAlpha - true to create an image with an alpha channel, false to create a opaque image without an alpha channel
- fillColor - the color (and alpha, if applicable) that the pixels are initially filled with

Returns:
the new image

Throws:
- IllegalArgumentException - if either width or height is zero or less

Since: MIDP 3.0

createImage

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 java.lang package documentation.

This method returns a object of type Image for static, bitmapped image data. An AnimatedImage is returned if the image data contains an animated bitmap image. A ScalableImage is returned if the image data contains valid vector graphics content such as SVG Tiny 1.1.

Parameters:
- name - the name of the resource containing the image data in one of the supported image formats

Returns:
the created image

Throws:
- NullPointerException - if name is null
- java.io.IOException - if the resource does not exist, the data cannot be loaded, or the image data cannot be decoded
createRGBImage

```java
public static javax.microedition.lcdui.Image createRGBImage(int[] rgb,
        int width,
        int height,
        boolean processAlpha)
```

Creates an immutable image from a sequence of ARGB values, specified using the 32-bit ARGB format. 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 and the resulting image will have an alpha channel. That is, 0x00RRGGBB specifies a fully transparent pixel, 0xFFRRGGBB specifies a fully opaque pixel, and intermediate alpha values specify semi-transparency.

If `processAlpha` is false, the high-order byte is ignored and the resulting image will not have an alpha channel. All pixels will be 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

\[
0 <= a < width \\
0 <= b < height
\]

**Parameters:**
- `rgb` - an array of ARGB values that composes the image
- `width` - the width of the image
- `height` - the height of the image
- `processAlpha` - true if the image should include alpha channel information contained in `rgb`, false if the image and all of its pixels should be 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

getARGB16

```java
public void getARGB16(short[] argbData,
        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 chars. Each pixel value is stored in a **16-bit ARGB** format that includes both the color and alpha values of the pixels.

The returned values are not guaranteed to be identical to values from the original source, such as from `createRGBImage` or from an image of the mandatory image format. Color values may be resampled to reflect the display capabilities of the device, number of alpha levels, and the limitations of the 16-bit format.

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 `argbData` 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:

\[
\text{argbData}[\text{offset} + (a - x) + (b - y) \times \text{scanlength}] = P(a, b);
\]

for

\[
\begin{align*}
x & \leq a < x + \text{width} \\
y & \leq b < y + \text{height}
\end{align*}
\]

The source rectangle is required to not exceed the bounds of the image. This means:

\[
\begin{align*}
x & \geq 0 \\
y & \geq 0 \\
x + \text{width} & \leq \text{image width} \\
y + \text{height} & \leq \text{image height}
\end{align*}
\]

If any of these conditions is not met an `IllegalArgumentException` is thrown. Otherwise, in cases where \( \text{width} \leq 0 \) or \( \text{height} \leq 0 \), no exception is thrown, and no pixel data is copied to `argbData`.

**Parameters:**
- `argbData` - an array of shorts in which the ARGB pixel data is stored
- `offset` - the index into the array where the first ARGB value is to be 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 `argbData` array whose index is either negative or beyond its length (the contents of the array are unchanged)
getGraphics

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 current alpha level is fully opaque;
- the current blending mode is SRC_OVER;
- 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 blending mode may be changed to SRC only if the Image includes an alpha channel.

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:
- 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 argbData is null

getHeight

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

public void getRGB(int[] argbData, int offset, int scanlength, int x, int y, int width, int height)
jjavax.microedition.lcdui.Image

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 using the 32-bit ARGB format.

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 alpha values are returned regardless of the image type, but they will all be 0xFF if the image does not include an alpha channel.

The returned values are not guaranteed to be identical to values from the original source, such as from createRGBImage or from an image of the mandatory image format. 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). Alpha channel values may be resampled to reflect the number of levels alpha supported by the device.

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:

\[
\text{rgbData}[\text{offset} + (a - x) + (b - y) \times \text{scanlength}] = P(a, b);
\]

for

\[
\begin{align*}
x & \leq a < x + \text{width} \\
y & \leq b < y + \text{height}
\end{align*}
\]

The source rectangle is required to not exceed the bounds of the image. This means:

\[
\begin{align*}
x & \geq 0 \\
y & \geq 0 \\
x + \text{width} & \leq \text{image width} \\
y + \text{height} & \leq \text{image height}
\end{align*}
\]

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
getRGB16

public void getRGB16(short[] rgbData,
        int offset,
        int scanlength,
        int x,
        int y,
        int width,
        int height)

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
Obtains RGB pixel data from the specified region of this image and stores it in the provided array of shorts. Each pixel value is stored in a 16-bit RGB format. The alpha values of the pixels are not reflected in the data returned.

The returned values are not guaranteed to be identical to values from the original source, such as from `createRGBImage` or from an image of the mandatory image format. Color values may be resampled to reflect the display capabilities of the device and the limitations of the 16-bit format.

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:

\[
\text{rgbData}[\text{offset} + (a - x) + (b - y) \times \text{scanlength}] = P(a, b);
\]

for

\[
x \leq a < x + \text{width} \\
y \leq b < y + \text{height}
\]

The source rectangle is required to not exceed the bounds of the image. This means:

\[
x \geq 0 \\
y \geq 0 \\
x + \text{width} \leq \text{image width} \\
y + \text{height} \leq \text{image height}
\]

If any of these conditions is not met an `IllegalArgumentException` is thrown. Otherwise, in cases where \( \text{width} \leq 0 \) or \( \text{height} \leq 0 \), no exception is thrown, and no pixel data is copied to `rgbData`.

**Parameters:**
- `rgbData` - an array of shorts in which the RGB pixel data is stored
- `offset` - the index into the array where the first RGB 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 3.0

**getWidth**

```java
public int getWidth()
```

Gets the width of the image in pixels. The value returned must reflect the actual width of the image when rendered.

**Returns:** width of the image

**hasAlpha**

```java
public boolean hasAlpha()
```

Checks if this image has alpha channel information. All of the pixels are fully opaque in an Image without an alpha channel. If an alpha channel is present, pixels within the Image may be opaque, or partially or fully transparent.

An immutable Image will have an alpha channel if it was created with an image resource having an transparent pixels. That is, the image resource contain at least one non-opaque pixel present in the image data. A mutable Image will have an alpha channel if it was explicitly created to have one; an alpha channel is required in order to use the SRC blending mode when rendering to a mutable image.

**Returns:** true if the image has an alpha channel false otherwise

Since: MIDP 3.0

**isAnimated**

```java
public boolean isAnimated()
```

Checks if this Image is an AnimatedImage comprised of several frames. If this method returns true, the Image is an instance of AnimatedImage and it may be cast to an AnimatedImage as needed.

**Returns:** true if this Image is an instance of an AnimatedImage

See Also: AnimatedImage

Since: MIDP 3.0

**isMutable**

```java
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.

**Returns:** true if the image is mutable, false otherwise

**isScalable**

```java
public boolean isScalable()
```

Checks if this Image is a ScalableImage that can be rasterized at a variety of different pixel sizes. If this method returns true, the Image is an instance of ScalableImage and it may be cast to a ScalableImage as needed.

**Returns:**
true if this Image is an instance of a ScalableImage

**See Also:** ScalableImage

**Since:** MIDP 3.0
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

Field Summary

<table>
<thead>
<tr>
<th>Field Access</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final</td>
<td>LAYOUT_CENTER</td>
</tr>
<tr>
<td>See Item.LAYOUT_CENTER.</td>
<td></td>
</tr>
<tr>
<td>Value: 3</td>
<td></td>
</tr>
<tr>
<td>public static final</td>
<td>LAYOUT_DEFAULT</td>
</tr>
<tr>
<td>See Item.LAYOUT_DEFAULT.</td>
<td></td>
</tr>
<tr>
<td>Value: 0</td>
<td></td>
</tr>
<tr>
<td>public static final</td>
<td>LAYOUT_LEFT</td>
</tr>
<tr>
<td>See Item.LAYOUT_LEFT.</td>
<td></td>
</tr>
<tr>
<td>Value: 1</td>
<td></td>
</tr>
<tr>
<td>public static final</td>
<td>LAYOUT_NEWLINE_AFTER</td>
</tr>
<tr>
<td>See Item.LAYOUT_NEWLINE_AFTER.</td>
<td></td>
</tr>
<tr>
<td>Value: 512</td>
<td></td>
</tr>
</tbody>
</table>
public static final LAYOUT_NEWLINE_BEFORE
See Item.LAYOUT_NEWLINE_BEFORE.
Value: 256

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

g是怎样工作的？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, getCommands, getLabel, getLayoutHint, getMinimumHeight, getMinimumWidth, getPreferredHeight, getPreferredWidth, notifyStateChanged, removeCommand, setCommand, setDefaultCommand, setItemCommandListener, setLabel, setLayoutHint, setPreferredSize

Methods inherited from class Object
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: 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. 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
javax.microedition.lcdui.ImageItem

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

cgetAltText

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: setAltText(String)

cgetAppearanceMode

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

cgetImage

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: setImage(Image)

cgetLayout

public int getLayout()

Gets the layout directives used for placing the image.

Returns:
a combination of layout directive values

See Also: setLayout(int)

csetAltText

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
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

```java
imageItem.setImage(imageItem.getImage());
```

to refresh the ImageItem's snapshot of its Image.

If the ImageItem is visible on the display when the snapshot is updated through a call to setImage, the display is updated with the new snapshot as soon as it is feasible for the implementation to so do.

Parameters:
- img - the Image for this ImageItem, or null if none

See Also: getImage()
javax.microedition.lcdui.Item

Declaration

class Item

Object

javax.microedition.lcdui.Item

Direct Known Subclasses:


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 depends on the Forms layout policy. Each subclass of FormLayoutPolicy defines which layout directives and ItemLayoutHints influence its layout. The default flow layout policy uses the predefined layout directives:

- LAYOUT_DEFAULT
- LAYOUT_LEFT
- LAYOUT_RIGHT
- LAYOUT_CENTER
- LAYOUT_TOP
- LAYOUT_BOTTOM
- LAYOUT_VCENTER
- LAYOUT_NEWLINE_BEFORE
- LAYOUT_NEWLINE_AFTER
- LAYOUT_SHRINK
- LAYOUT_VSHRINK
- LAYOUT_EXPAND
- LAYOUT_VEXPAND
- LAYOUT_2

The LAYOUT_DEFAULT directive indicates that the container's default layout policy is to be used for this item. LAYOUT_DEFAULT has the value zero and has no effect when combined with other layout directives.
It is useful within programs in order to document the programmer's intent.

The `LAYOUT_LEFT`, `LAYOUT_RIGHT`, and `LAYOUT_CENTER` directives indicate horizontal alignment and are mutually exclusive. Similarly, the `LAYOUT_TOP`, `LAYOUT_BOTTOM`, and `LAYOUT_VCENTER` directives indicate vertical alignment and are mutually exclusive.

A horizontal alignment directive, a vertical alignment directive, and any combination of other layout directives may be combined using the bit-wise OR operator (`|`) to compose a layout directive value. Such a value is used as the parameter to the `setLayout(int)` method and is the return value from the `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.

Some layout policy subclasses use additional layout hints that are set on Items using the `setLayoutHint` method. The current layout hint is available from the `getLayoutHint` method. Refer to the specific layout subclass for the types and values of hints.

Applications may select a layout policy and may create new subclasses of `FormLayoutPolicy`. A complete specification of the default flow layout of Items within a Form is given in `Form`.

**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 getPreferredWidth 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 addCommand(Command) or setDefaultCommand(Command) and if the Command has not been removed with a subsequent call to 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 item's commands as if they were of type ITEM.

Items may have a default Command. This state is controlled by the 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 of Displayable on which it occurs, while those Items of 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 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

---

**Field Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final</td>
<td>BUTTON</td>
<td>An appearance mode value indicating that the Item MUST appear as a button.</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Value: 2</td>
</tr>
<tr>
<td>public static final</td>
<td>HYPERLINK</td>
<td>An appearance mode value indicating that the Item MUST appear as a hyperlink.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Value: 1</td>
</tr>
<tr>
<td>public static final</td>
<td>LAYOUT_2</td>
<td>A layout directive indicating that new MIDP 2.0 layout rules are in effect</td>
</tr>
<tr>
<td></td>
<td>16384</td>
<td>for this Item.</td>
</tr>
<tr>
<td>public static final</td>
<td>LAYOUT_BOTTOM</td>
<td>A layout directive indicating that this Item MUST have a bottom-aligned</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>layout.</td>
</tr>
<tr>
<td>Method</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LAYOUT_CENTER</td>
<td>3</td>
<td>A layout directive indicating that this Item MUST have a horizontally centered layout.</td>
</tr>
<tr>
<td>LAYOUT_DEFAULT</td>
<td>0</td>
<td>A layout directive indicating that this Item MUST follow the default layout policy of its container.</td>
</tr>
<tr>
<td>LAYOUT_EXPAND</td>
<td>2048</td>
<td>A layout directive indicating that this Item's width may be increased to fill available space.</td>
</tr>
<tr>
<td>LAYOUT_LEFT</td>
<td>1</td>
<td>A layout directive indicating that this Item MUST have a left-aligned layout.</td>
</tr>
<tr>
<td>LAYOUT_NEWLINE_AFTER</td>
<td>512</td>
<td>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.</td>
</tr>
<tr>
<td>LAYOUT_NEWLINE_BEFORE</td>
<td>256</td>
<td>A layout directive indicating that this Item MUST be placed at the beginning of a new line or row.</td>
</tr>
<tr>
<td>LAYOUT_RIGHT</td>
<td>2</td>
<td>A layout directive indicating that this Item MUST have a right-aligned layout.</td>
</tr>
<tr>
<td>LAYOUT_SHRINK</td>
<td>1024</td>
<td>A layout directive indicating that this Item's width may be reduced to its minimum width.</td>
</tr>
<tr>
<td>LAYOUT_TOP</td>
<td>16</td>
<td>A layout directive indicating that this Item MUST have a top-aligned layout.</td>
</tr>
<tr>
<td>LAYOUT_VCENTER</td>
<td>48</td>
<td>A layout directive indicating that this Item MUST have a vertically centered layout.</td>
</tr>
<tr>
<td>LAYOUT_VEXPAND</td>
<td>8192</td>
<td>A layout directive indicating that this Item's height MUST be increased to fill available space.</td>
</tr>
</tbody>
</table>
public static final LAYOUT_VSHRINK
A layout directive indicating that this Item's height may be reduced to its minimum height.
Value: 4096

public static final PLAIN
An appearance mode value indicating that the Item MUST have a normal appearance.
Value: 0

Method Summary

void addCommand(Command cmd)
Adds a context sensitive Command to the item.

javax.microedition.lcdui.Command[] getCommands()
Gets the set of commands that has been added to the Item.

java.lang.String getLabel()
Gets the label of this Item object.

int getLayout()
Gets the layout directives used for placing the item.

javax.microedition.lcdui.ItemLayoutHint getLayoutHint()
Gets the layout hint used by the layout policy for this item.

int getMinimumHeight()
Gets the minimum height for this Item.

int getMinimumWidth()
Gets the minimum width for this Item.

int getPreferredHeight()
Gets the preferred height of this Item.

int getPreferredWidth()
Gets the preferred width of this Item.

void notifyStateChanged()
Causes this Item's containing Form to notify the Item's ItemStateListener.

void removeCommand(Command cmd)
Removes the context sensitive command from item.

void setCommand(Command cmd, int placement)
Adds or sets a Command to the Item at the given placement.

void setDefaultCommand(Command cmd)
Sets default Command for this Item.

void setItemCommandListener(ItemCommandListener l)
Sets a listener for Commands to this Item, replacing any previous ItemCommandListener.

void setLabel(String label)
Sets the label of the Item.
**void** setLayout(int layout)
Sets the layout directives for this item.

**void** setLayoutHint(ItemLayoutHint hint)
Sets the layout hint used for this item by the layout policy.

**void** setPreferredSize(int width, int height)
Sets the preferred width and height for this Item.

### 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

**Since:** MIDP 2.0

**LAYOUT_2**

`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**

`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
javax.microedition.lcdui.Item

LAYOUT_CENTER

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

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

public static final int LAYOUT_EXPAND

A layout directive indicating that this Item’s width may be increased to fill available space.

Value 0x800 is assigned to LAYOUT_EXPAND.
Constant value: 2048
Since: MIDP 2.0

LAYOUT_LEFT

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**

```java
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**

```java
public static final int LAYOUT_SHRINK
```

A layout directive indicating that this `Item`'s width may be reduced to its minimum width.

Value 0x400 is assigned to `LAYOUT_SHRINK`.
Constant value: 1024

Since: MIDP 2.0

**LAYOUT_TOP**

```java
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
```

A layout directive indicating that this `Item`'s height MUST 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
```

A layout directive indicating that this `Item`'s height may be reduced to its minimum height.

Value 0x1000 is assigned to `LAYOUT_VSHRINK`.
Constant value: 4096
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

addClass

public void addClass(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
- DisplayCapabilityException - if the Item has Commands and this Form is currently being displayed on a Display that does not support Commands

Since: MIDP 2.0

getCommands

public javax.microedition.lcdui.Command[] getCommands()

Gets the set of commands that has been added to the Item.

Returns:
- An array containing all commands added to the item via addClass()

Since: MIDP 3.0

getLabel

public java.lang.String getLabel()

Gets the label of this Item object.

Returns:
- the label string

See Also: setLabel(String)

Since: MIDP 1.0

getLayout

public int getLayout()
**getLayoutHint**

```java
public javax.microedition.lcdui.ItemLayoutHint getLayoutHint()
```

Gets the layout hint used by the layout policy for this item.

**Returns:**
A `ItemLayoutHint` instance or null.

**See Also:** `setLayout(int)`, `setLayoutHint(ItemLayoutHint)`

**Since:** MIDP 3.0

---

**getMinimumHeight**

```java
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**

```java
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

---

**getPreferredSizeHeight**

```java
public int getPreferredSizeHeight()```

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:** `getPreferredSizeWidth()`, `setPreferredSize(int, int)`

**Since:** MIDP 2.0

---

**getPreferredSizeWidth**

```java
public int getPreferredSizeWidth()
```
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:** `getPreferredSize()`, `setPreferredSize(int, int)`

**Since:** MIDP 2.0

### notifyStateChanged

```java
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 `ItemStateListener.itemStateChanged` may be delayed in order to be serialized with the event stream. The `notifyStateChanged` method does not block awaiting the completion of the `itemStateChanged` method.

**Throws:**
`IllegalStateException` - if the `Item` is not owned by a `Form`

**Since:** MIDP 2.0

### removeCommand

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


Removed 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

---

**setCommand**

```java
public void setCommand(Command cmd, int placement)
```

Adds or sets a Command to the Item at the given placement. If the Command already exists on another placement of this Item, it will be moved to the new placement. Just before the Command is added, any Menu or Command already occupying the given placement will be removed from the associated Item or Displayable. This method should only be called from inside an implementation of the CommandLayoutPolicy interface. 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.

**Parameters:**
- cmd - the Command to be added
- placement - the placement of the Command to be added

**Throws:**
- NullPointerException - if cmd is null
- DisplayCapabilityException - if the Item has Commands and this Form is currently being displayed on a Display that does not support Commands
- IllegalArgumentException - if the provided placement is not valid or is associated with a placement that does not support commands
- IllegalStateException - if the method is not called from a CommandLayoutPolicy.onCommandLayout() implementation.

**Since:** MIDP 3.0

---

**setDefaultCommand**

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

**Since:** MIDP 3.0
Sets default Command for this Item. If the Item previously had a default Command, that Command is no longer the default, but it remains present on the Item.

If not null, the Command object passed becomes the default Command for this Item. If the Command object passed is not currently present on this Item, it is added as if addCommand(Command) 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
   DisplayCapabilityException - if the Item has Commands and this Form is currently being displayed on a Display that does not support Commands

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

Since: MIDP 2.0

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: getLabel()
Since: MIDP 1.0

setLayout

public void setLayout(int layout)

Sets the layout directives for this item.

It is illegal to call this method if this Item is contained within an Alert.

Parameters:
   layout - a combination of layout directive values for this item

Throws: None
IllegalArgumentException - if the value of layout is not a bit-wise OR combination of layout directives
IllegalStateException - if this Item is contained within an Alert

See Also: getLayout()  
Since: MIDP 2.0

setLayoutHint

public void setLayoutHint(ItemLayoutHint hint)

Sets the layout hint used for this item by the layout policy. The hint is only used by the layout if it is a type required by the form layout, otherwise it is ignored. If the layout policy of the form is changed (Form.setLayoutPolicy) the application supplied layout hints may need to be updated with hints appropriate for the new layout policy.

If the Item is contained within an Alert then the layout hint is ignored.

Parameters:
   hint - an instance of ItemLayoutHint appropriate for the current Layout policy; may be null.

See Also: getLayout(), getLayoutHint()  
Since: MIDP 3.0

setPreferredSize

public void setPreferredSize(int width, int height)

Sets the preferred width and height for this Item. Values for width and height less than -1 are illegal. If the width is between zero and the minimum width, inclusive, the minimum width is used instead. If the height is between zero and the minimum height, inclusive, the minimum height is used instead.

Supplying a width or height value greater than the minimum width or height locks that dimension to the supplied value. The implementation may silently enforce a maximum dimension for an Item based on factors such as the screen size. Supplying a value of -1 for the width or height unlocks that dimension. See Item Sizes for a complete discussion.

It is illegal to call this method if this Item is contained within an Alert.

Parameters:
   width - the value to which the width should be locked, or -1 to unlock
   height - the value to which the height should be locked, or -1 to unlock

Throws:
   IllegalArgumentException - if width or height is less than -1
   IllegalStateException - if this Item is contained within an Alert

See Also: getPreferredHeight(), getPreferredSize()  
Since: MIDP 2.0
javax.microedition.lcdui

ItemCommandListener

Declaration

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>void commandAction(Command c, Item item)</td>
<td>Called by the system to indicate that a command has been invoked on a particular item.</td>
</tr>
</tbody>
</table>

Methods

commandAction

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
ItemLayoutHint

Declaration

public interface ItemLayoutHint

Description

ItemLayoutHint is an interface to identify classes containing hints that control the layout of Items by subclasses of FormLayoutPolicy. Each layout subclass defines the classes that specify the hints it uses and how they affect layout. Refer to the subclasses of FormLayoutPolicy for a description of the layout hints accepted.

For example, a layout policy might choose to display items in a priority order. Each item would need to have a priority set for it.

```java
class ItemPriority implements ItemLayoutHint {
    int getPriority();
    void setPriority(int priority);
}
```

To set the priority of each item an ItemPriority class would be created and set for each item.

```java
Item[] items = ...;
for (int i = 0; i < items.length; i++) {
    // Set the priority to reverse the list.
    ItemPriority prio = new ItemPriority(items.length - i);
    items[i].setLayoutHint(prio);
}
```

Since: MIDP 3.0

See Also: Item.setLayoutHint(ItemLayoutHint)
javax.microedition.lcdui

ItemStateListener

Declaration

public interface ItemStateListener

Description

This interface is used by applications that need to receive events indicating changes in the internal state of the interactive items within a Form screen.

Since: MIDP 1.0

See Also: Form.setItemStateListener(ItemStateListener), ItemTraversalListener

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>void itemStateChanged(Item item)</td>
</tr>
</tbody>
</table>

Called when internal state of an Item has been changed by the user.

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.ItemTraversalListener

Declaration

public interface ItemTraversalListener

Description

This interface is used by applications that need to receive events indicating changes in focus for Items in a Form screen.

An Item has focus when it is selected as the active recipient of regular key events. Thus, an item will not have focus if another Item has focus, if its Form is hidden, or if its Form is shown on a Display that does not have foreground status. None of the Items of a Form will have focus if the Form is not currently being shown on a Display having foreground status.

Traversal events within a given Item are not reported to this listener. For standard Item classes such as StringItem and ChoiceGroup, internal traversal is platform-dependent. For a CustomItem, internal traversal can be controlled by overriding its traverse and traverseOut methods.

Since: MIDP 3.0

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

Method Summary

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>itemTraversedIn</td>
<td>void itemTraversedIn(Item item)</td>
<td>Called when the item gains focus.</td>
</tr>
<tr>
<td>itemTraversedOut</td>
<td>void itemTraversedOut(Item item)</td>
<td>Called when the item loses focus.</td>
</tr>
</tbody>
</table>

Methods

itemTraversedIn

public void itemTraversedIn(Item item)

Called when the item gains focus. The item may gain focus for a variety of reasons including:

- The user navigated to it
- The parent Form was shown and the item has default focus
- The Display associated with the parent Form gained foreground status and the item had focus before the Display was assigned background status.

Parameters:

item - The item that gained focus

Since: MIDP 3.0

itemTraversedOut

public void itemTraversedOut(Item item)
javax.microedition.lcdui.ItemTraversalListener

Called when the item loses focus. The item may lose focus for a variety of reasons including:

- The user navigated to a different Item
- The parent Form was hidden or another screen was shown instead
- The parent Form's Display lost foreground status.

**Parameters:**

- `item` - The item that lost focus

**Since:** MIDP 3.0
javax.microedition.lcdui.KeyListener

Declaration

public interface KeyListener

Description

Classes implementing this interface provide methods that are called when user of the device will generate key events, for example, pressing the keys available in a system keypad or keyboard. When a key is pressed, held down or released, the appropriate method will be invoked. The methods have following parameters:

- **keyCode**: keyCodes often represent the character of the key that was typed. This is the final character that results after all modifiers have been applied. The keyCode values are the same that are used and defined for key event delivery methods of Canvas.
- **keyModifier**: carries the key modifier (Alt, Shift, Ctrl, Chr etc.) information. Modifier keys are only available in some devices, especially those with full keyboards.
- The KeyListener may be set with Canvas.setKeyListener() method and CustomItem.setKeyListener().

Since: MIDP 3.0

Field Summary

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODIFIER_ALT</td>
<td>Key modifier mask indicating the Alt key.</td>
<td>16384</td>
</tr>
<tr>
<td>MODIFIER_CHR</td>
<td>Key modifier mask indicating the Chr key.</td>
<td>8388608</td>
</tr>
<tr>
<td>MODIFIER_COMMAND</td>
<td>Key modifier mask indicating the Command key.</td>
<td>4194304</td>
</tr>
<tr>
<td>MODIFIER_CTRL</td>
<td>Key modifier mask indicating the Control / Ctrl key.</td>
<td>262144</td>
</tr>
<tr>
<td>MODIFIER_MASK</td>
<td>A bitmask for all key modifiers.</td>
<td>13041664</td>
</tr>
<tr>
<td>MODIFIER_SHIFT</td>
<td>Key modifier mask indicating the SHIFT key.</td>
<td>131072</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void keyPressed(int keyCode, int keyModifier)</td>
<td>Called when a key is pressed.</td>
</tr>
</tbody>
</table>
javax.microedition.lcduiKeyListener

| void keyReleased(int keyCode, int keyModifier) |
| Called when a key is released. |
| void keyRepeated(int keyCode, int keyModifier) |
| Called when a key is repeated (held down). |

**Fields**

**MODIFIER_ALT**

```java
class Modifier {
    public static final int MODIFIER_ALT = 65536;
    // Key modifier mask indicating the Alt key.
}
```

**MODIFIER_CHR**

```java
class Modifier {
    public static final int MODIFIER_CHR = 8388608;
    // Key modifier mask indicating the Chr key.
}
```

**MODIFIER_COMMAND**

```java
class Modifier {
    public static final int MODIFIER_COMMAND = 4194304;
    // Key modifier mask indicating the Command key.
}
```

**MODIFIER_CTRL**

```java
class Modifier {
    public static final int MODIFIER_CTRL = 262144;
    // Key modifier mask indicating the Control / Ctrl key.
}
```

**MODIFIER_MASK**

```java
class Modifier {
    public static final int MODIFIER_MASK = 13041664;
    // A bitmask for all key modifiers. Not initialized in the declaration to stop the compiler from inlining the value.
}
```

**MODIFIER_SHIFT**

```java
class Modifier {
    public static final int MODIFIER_SHIFT = 131072;
    // Key modifier mask indicating the SHIFT key.
}
```

**Methods**

**keyPressed**

```java
public void keyPressed(int keyCode, int keyModifier) {
}
```
Called when a key is pressed.

**Parameters:**
- keyCode - the key code of the key that was pressed
- keyModifier - the key modifier mask of the modifier key that was pressed

---

**keyReleased**

```java
public void keyReleased(int keyCode, int keyModifier)
```

Called when a key is released.

**Parameters:**
- keyCode - the key code of the key that was released
- keyModifier - the key modifier mask of the modifier key that was released

---

**keyRepeated**

```java
public void keyRepeated(int keyCode, int keyModifier)
```

Called when a key is repeated (held down).

**Parameters:**
- keyCode - the key code of the key that was held down
- keyModifier - the key modifier mask of the modifier key that was held down
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>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td><strong>List(String title, int listType)</strong></td>
</tr>
<tr>
<td></td>
<td>Creates a new, empty List, specifying its title and the type of the list.</td>
</tr>
<tr>
<td>public</td>
<td><strong>List(String title, int listType, String[] stringElements, Image[] imageElements)</strong></td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td><strong>append(String stringPart, Image imagePart)</strong></td>
</tr>
<tr>
<td></td>
<td>Appends an element to the List.</td>
</tr>
<tr>
<td>void</td>
<td><strong>delete(int elementNum)</strong></td>
</tr>
<tr>
<td></td>
<td>Deletes the element referenced by elementNum.</td>
</tr>
<tr>
<td>void</td>
<td><strong>deleteAll()</strong></td>
</tr>
<tr>
<td></td>
<td>Deletes all elements from this List.</td>
</tr>
<tr>
<td>int</td>
<td><strong>getFitPolicy()</strong></td>
</tr>
<tr>
<td></td>
<td>Gets the application's preferred policy for fitting Choice element contents to the available screen space.</td>
</tr>
<tr>
<td><strong>javax.microedition.lcdui.Font</strong></td>
<td><strong>getFont(int elementNum)</strong></td>
</tr>
<tr>
<td></td>
<td>Gets the application's preferred font for rendering the specified element of this Choice.</td>
</tr>
<tr>
<td>int</td>
<td><strong>getHeight()</strong></td>
</tr>
<tr>
<td></td>
<td>Returns the height in pixels of the displayable area available for elements.</td>
</tr>
<tr>
<td><strong>javax.microedition.lcdui.Image</strong></td>
<td><strong>getImage(int elementNum)</strong></td>
</tr>
<tr>
<td></td>
<td>Gets the Image part of the element referenced by elementNum.</td>
</tr>
<tr>
<td>int</td>
<td><strong>getSelectedFlags(boolean[] selectedArray_return)</strong></td>
</tr>
<tr>
<td></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</td>
<td><strong>getSelectedIndex()</strong></td>
</tr>
<tr>
<td></td>
<td>Returns the index number of an element in the List that is selected.</td>
</tr>
<tr>
<td>java.lang.String</td>
<td><strong>getString(int elementNum)</strong></td>
</tr>
<tr>
<td></td>
<td>Gets the String part of the element referenced by elementNum.</td>
</tr>
<tr>
<td>int</td>
<td><strong>getWidth()</strong></td>
</tr>
<tr>
<td></td>
<td>Returns the width in pixels of the displayable area available for elements.</td>
</tr>
<tr>
<td>void</td>
<td><strong>insert(int elementNum, String stringPart, Image imagePart)</strong></td>
</tr>
<tr>
<td></td>
<td>Inserts an element into the List just prior to the element specified.</td>
</tr>
<tr>
<td>boolean</td>
<td><strong>isEnabled(int elementNum)</strong></td>
</tr>
<tr>
<td></td>
<td>Gets a boolean value indicating whether the specified Choice element is enabled or disabled.</td>
</tr>
<tr>
<td>boolean</td>
<td><strong>isSelected(int elementNum)</strong></td>
</tr>
<tr>
<td></td>
<td>Gets a boolean value indicating whether this element is selected.</td>
</tr>
</tbody>
</table>
### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 setEnabled(int elementNum, boolean isEnabled)</td>
<td>Sets the Choice element into enabled or disabled mode.</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>
<tr>
<td>void setSelectedFlags(boolean[] selectedArray)</td>
<td>Sets the selected state of all elements of the List.</td>
</tr>
<tr>
<td>void setselectedIndex(int elementNum, boolean selected)</td>
<td>Sets the selected state of an element.</td>
</tr>
<tr>
<td>int size()</td>
<td>Gets the number of elements in the List.</td>
</tr>
</tbody>
</table>

Methods inherited from class [javax.microedition.lcdui.Displayable](https://docs.oracle.com/javase/8/docs/api/javax/microedition/lcdui/Displayable.html)

- addCommand, getCommand, getCommandLayoutPolicy, getCommands, getCurrentDisplay, getMenu, getTicker, getTitle, invalidateCommandLayout, isShown, removeCommandOrMenu, setCommand, setCommandLayoutPolicy, setCommandListener, setMenu, setTicker, setTitle, sizeChanged

Methods inherited from class [Object](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html)

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

Methods inherited from interface [javax.microedition.lcdui.Choice](https://docs.oracle.com/javase/8/docs/api/javax/microedition/lcdui/Choice.html)

### Fields

**SELECT_COMMAND**

public static final [javax.microedition.lcdui.Command](https://docs.oracle.com/javase/8/docs/api/javax/microedition/lcdui/Command.html) 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**

```java
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:**
- IllegalArgumentException - if `listType` is not one of IMPLICIT, EXCLUSIVE, or MULTIPLE

**See Also:** Choice

---

**List**

```java
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.

The elementNum parameter must be within the range [0..size()-1], inclusive.

Parameters:
elementNum - the index of the element to be deleted, starting from zero

Throws:
java.lang.IndexOutOfBoundsException - if elementNum is invalid

deleteAll

public void deleteAll()

Deletes all elements from this List.

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.

See Also: 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.
javax.microedition.lcdui.List

**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:** `setFont(int, Font)`

**Since:** MIDP 2.0

---

**getHeight**

```java
public int getHeight()
```

Returns the height in pixels of the displayable area available for elements. This value is the height available to items in the `List` that can be shown without scrolling. If the `List` is bound to a `Display` through a call to `setCurrent`, the height returned MUST NOT be larger that `getHeight`. If a `List` in a `TabbedPane` is brought into focus, then the height returned MUST NOT be larger than `getHeight`.

**Returns:**
- the height in pixels available to the elements of the `List`

**Since:** MIDP 2.0

---

**getImage**

```java
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, starting from zero

**Returns:**
- the image part of the element, or `null` if there is no image

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

**See Also:** `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`.

**Parameters:**
- `selectedArray_return` - array to contain the results

**Returns:**
- the number of selected elements in the `Choice`

**Throws:**
- `IllegalArgumentException` - if `selectedArray_return` is shorter than the size of the `List`
- `NullPointerException` - if `selectedArray_return` is `null`

**See Also:** `getSelectedFlags(boolean[])`

---

**getSelectedIndex**

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

Returns the index number of an element in the List that is selected.

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

See Also: 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, starting from zero

Returns:
   the string part of the element

Throws:
   IndexOutOfBoundsException - if elementNum is invalid

See Also: getImage(int)

getWidth

public int getWidth()

Returns the width in pixels of the displayable area available for elements. This value is the width available to items in the List that can be shown without scrolling. If the List is bound to a Display through a call to setCurrent, the width returned MUST NOT be larger that getWidth. If a List in a TabbedPane is brought into focus, then the width returned MUST NOT be larger than getWidth.

Returns:
   the width in pixels available to the elements of the List

Since: MIDP 2.0

insert

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

Inserts an element into the List just prior to the element specified.

The elementNum parameter must be within the range [0..size()-1], inclusive.

Parameters:
   elementNum - the index of the element where insertion is to occur, starting from zero
   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

isEnabled

public boolean isEnabled(int elementNum)

Gets a boolean value indicating whether the specified Choice element is enabled or disabled.

The elementNum parameter must be within the range [0..size()-1], inclusive.
Since: MIDP 3.0

isSelected

```java
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, starting from zero

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

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

removeCommand

```java
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.

The elementNum parameter must be within the range [0..size()-1], inclusive.

Parameters:
- elementNum - the index of the element to be set, starting from zero
- 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

setEnabled

public void setEnabled(int elementNum,
           boolean isEnabled)

Sets the Choice element into enabled or disabled mode. The set mode applies for the element of the Choice object specified by the elementNum parameter. The elementNum parameter must be within the range [0..size()-1], inclusive. A disabled Choice element SHOULD be visually different from an enabled Choice element.

Parameters:
- elementNum - the index of the element to which the mode change is applied, the index starts from zero
- isEnabled - true if enabled, false if disabled

Throws:
- IndexOutOfBoundsException - if elementNum is invalid

See Also: isEnabled(int)
Since: MIDP 3.0

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: getFitPolicy()
Since: MIDP 2.0

setFont

public void setFont(int elementNum,
           Font font)
setSelectCommand

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

public void setSelectedFlags(boolean[] selectedArray)

Sets the selected state of all elements of the List.

Parameters:
selectedArray - an array in which the method collect the selection status

Throws:
IllegalArgumentException - if selectedArray is shorter than the size of the List
NullPointerException - if selectedArray is null

See Also: setSelectedFlags(boolean[])
**setSelectedIndex**

```java
class javax.microedition.lcdui.List
```

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

Sets the selected state of an element.

The `elementNum` parameter must be within the range `[0..size()-1]`, inclusive.

**Parameters:**
- `elementNum` - the index of the element, starting from zero
- `selected` - the state of the element, where `true` means selected and `false` means not selected

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

**See Also:** `getSelectedIndex()`

---

**size**

```java
class javax.microedition.lcdui.List
```

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

Gets the number of elements in the `List`.

**Returns:**
- the number of elements in the `List`
javax.microedition.lcdui.Menu

Declaration

public class Menu

Object

-java.microedition.lcdui.Menu

Description

A visual container for Commands and other Menus. The Menu class is intended to be used in association with exact placement of commands to group together commands that do not fit on the given display or should be grouped together for usability reasons.

Menu Creation and Population

A Menu Object is always created empty, and later populated with commands using either the append() or insert() methods. A given Command may only be added once to a given Menu, but can be added to more than one Menu.

Versions of these methods are also provided for adding sub-menus to a menu, thereby enabling the creation of a menu tree. A menu may have multiple menus added to it as sub-menus, but a menu may be added to no more than one parent menu.

It is up to the implementation how to handle manipulation (addition/removal) of menu content when a menu is visible. The application may use the isVisible() method to find out if the Menu is presently visible on any display.

The platform may limit the depth of sub-menus on a display. The present depth of a menu, may be retrieved by the getMenuDepth() method. The device-wide maximum menu depth is returned by the getMaxMenuDepth() method; the implementation MUST support a menu depth of up to five(5) menus, but MAY support a higher menu depth.

Adding Menus to Displayables

Menus are added to a Displayable with the Displayable.setMenu method. The same Menu can be added to multiple Displayables at the same time. However, the platform MUST ensure that only the menu belonging to the Displayable currently in focus is shown. This means that the same menu button may appear on multiple Displays, but the menu itself can only be shown on one Display at a time. A Menu may generally be placed on the same exact placements as Commands. However, setMenu will throw java.lang.IllegalArgumentException when the attempted placement is not allowed. The reason for failure can be that the particular placement does not support menus. Doing additional setMenu to a Displayable with different placement will move the Menu to the new placement.

Note that there is a difference between Displayable.setMenu(Menu) and Menu.append(Menu); A single Menu object may be added to multiple Displayables, but only be shown on one at the same time. The same Menu object may only be appended to one menu at any time.

As an example of using the Menu class, consider the following example. The developer wants to set up a menu tree consisting of a top menu with two levels of sub menus.
Visibility and Interaction

A menu is initially not visible, but is brought into view only as it is selected by the user, typically by pressing a associated soft key. The implementation must provide the ability to navigate a menu, to select a command in a menu, and to allow the user to leave the menu without selecting anything (this is required for all displays that support menus). When a Command is selected, it is passed to the CommandListener of the Displayable of the Display where the Menu is presently visible. It is up to the implementation to associate the correct Displayable with a particular Menu when it is to be used. When a command is selected, or the user navigates out of the menu, the menu should become invisible as soon as possible.

A Menu MUST NOT be visible on more than one Display at a time. If a Menu is visible on a Display that loses focus, then the Menu should be made invisible as soon as possible, and if the that Display should regain focus, the Menu should again start out as invisible. There MUST NOT be any interaction capability with a Menu that remains visible after its Display has lost focus.

Images

The application may add an Image to the Menu to be shown together with the labels. The device shows the image on a best effort basis, depending on what can be done on the particular display. For example, a Menu with an image is added to two different displays, one with high resolution and one with low, the implementation may decide either to show the image only on the high resolution display or ignore the image completely. Implementations may truncate or scale the icon image if it is larger than the size supported by device.

The Image may be mutable or immutable. If the Image is mutable, then a snapshot of its contents is taken before the Menu() constructor or setImage() method returns. The snapshot is used whenever the contents of the Menu are to be displayed. Even if the application subsequently draws into the Image, the snapshot is not modified until the next call to setImage(). If the Menu is visible on the display then the display SHOULD be updated with the new snapshot as soon as it is feasible for the implementation to do so.
Applications can query the implementation's Menu icon size by calling Display.getBestImageWidth(int) and Display.getBestImageHeight(int) methods using the Display.MENU image type. The style and appearance of menus are platform-dependent. If the image is changed after the Menu is first displayed, then it is up to the implementation to act on the change as soon as possible.

**Since:** MIDP 3.0

### Constructor Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>Menu(String shortLabel, String longLabel, Image image)</td>
<td>Creates a new Menu object with the given labels and image.</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>append(Command cmd)</td>
<td>Appends a Command to the Menu.</td>
</tr>
<tr>
<td>int</td>
<td>append(Menu menu)</td>
<td>Appends a sub-menu to this Menu.</td>
</tr>
<tr>
<td>javax.microedition.lcdui.Command</td>
<td>getCommand(int index)</td>
<td>Gets the command at index.</td>
</tr>
<tr>
<td>javax.microedition.lcdui.Font</td>
<td>getFont()</td>
<td>Gets the application's preferred font for rendering the labels for this Menu.</td>
</tr>
<tr>
<td>javax.microedition.lcdui.Image</td>
<td>getImage()</td>
<td>Gets the Image for this Menu.</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getLabel()</td>
<td>Gets the label of the Menu.</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getLongLabel()</td>
<td>Gets the long label of the Menu.</td>
</tr>
<tr>
<td>static int</td>
<td>getMaxMenuDepth()</td>
<td>Returns the device-wide maximum menu depth.</td>
</tr>
<tr>
<td>javax.microedition.lcdui.Menu</td>
<td>getMenu(int index)</td>
<td>Gets the sub-menu at index.</td>
</tr>
<tr>
<td>int</td>
<td>getMenuDepth()</td>
<td>Returns this menu's depth within its menu tree.</td>
</tr>
<tr>
<td>void</td>
<td>insert(int index, Command cmd)</td>
<td>Inserts a Command into the Menu just prior to the element specified.</td>
</tr>
<tr>
<td>void</td>
<td>insert(int index, Menu menu)</td>
<td>Inserts a Menu into the Menu just prior to the element specified.</td>
</tr>
<tr>
<td>boolean</td>
<td>isCommand(int index)</td>
<td>Checks if the element at the indicated index is a Command.</td>
</tr>
<tr>
<td>boolean</td>
<td>isEnabled()</td>
<td>Gets a boolean value indicating whether the Menu is enabled or disabled.</td>
</tr>
<tr>
<td>boolean</td>
<td>isVisible()</td>
<td>Gets a boolean value indicating whether the Menu is presently visible on any Display.</td>
</tr>
</tbody>
</table>
void onParentEnabled(boolean enabled)

Informs the menu of a change in its parent's enabled value.

void remove(Command cmd)

Removes a command from the menu.

void remove(Menu menu)

Removes a sub-menu from the menu.

void setEnabled(boolean isEnabled)

Sets the Menu into enabled or disabled mode.

void setFont(Font font)

Sets the application's preferred font for rendering the labels for this Menu.

void setImage(Image image)

Sets the Image for this Menu.

void setLabel(String shortLabel)

Sets the short label of the Menu.

void setLongLabel(String longLabel)

Sets the long label of the Menu.

int size()

Gets the number of Commands and sub-Menus in the Menu.

Methods inherited from class Object

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

Constructors

Menu

public Menu(String shortLabel,
        String longLabel,
        Image image)

Creates a new Menu object with the given labels and image.

The short label is required and must not be null. The long label is optional and may be null if the Menu is to have no long label. The labels are used like the labels on commands, they may also be used by the implementation as descriptive headers on the menu when it is being shown.

Parameters:

    shortLabel - the Menu's short label
    longLabel - the Menu's long label, or null if none
    image - the Menu's image, or null if none

Throws:

    NullPointerException - if shortLabel is null

Methods

append

public int append(Command cmd)
append

```java
public int append(Menu menu)
```

Appends a sub-menu to this Menu. The sub-menu will be the last element of the menu and the size of the menu will grow by one. A sub-menu can be in only one menu at a time; the application is responsible for removing the sub-menu from a previous menu, if any, before it can be appended to a new menu.

**Parameters:**
- `menu` - the menu to be added

**Returns:**
- the zero-based index of the Menu in this menu

**Throws:**
- `NullPointerException` - if `menu` is `null`
- `IllegalArgumentException` - if the `menu` already has a parent menu or if it is the top menu in this menu's tree.
- `IllegalStateException` - if the resulting menu depth exceeds the maximum depth supported by the implementation. The menu depth includes all of the menus above this menu, the menu being inserted, and all of its sub-menus.

getCommand

```java
public javax.microedition.lcdui.Command getCommand(int index)
```

Gets the command at index. The `index` parameter must be within the range `[0..size()-1]`, inclusive.

**Returns:**
- the command at the given index

**Throws:**
- `IndexOutOfBoundsException` - if `index` is less than zero or greater than or equal to `size()`.
- `IllegalArgumentException` - if element at index is not a command

getFont

```java
public javax.microedition.lcdui.Font getFont()
```

Gets the application's preferred font for rendering the labels for this Menu. 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 the element

getImage

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

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

Gets the label of the Menu.

**Returns:**
the short label of the Menu

**getLongLabel**

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

Gets the long label of the Menu.

**Returns:**
the long label of the Menu, or null if there is no long label

**getMaxMenuDepth**

```java
public static int getMaxMenuDepth()
```

Returns the device-wide maximum menu depth. The maximum is at least five (5), but may be greater.

**Returns:**
the device-wide menu depth limit

**getMenu**

```java
public javax.microedition.lcdui.Menu getMenu(int index)
```

Gets the sub-menu at index. The index parameter must be within the range [0..size()-1], inclusive.

**Returns:**
the menu at the given index

**Throws:**
IndexOutOfBoundsException - if index is less than zero or greater than or equal to size().
IllegalArgumentException - if element at index is not a menu

**getMenuDepth**

```java
public int getMenuDepth()
```

Returns this menu's depth within its menu tree. The top menu is of depth 0, the menus attached to it are of depth 1, 2, etc. The top menu is the menu in the menu tree that has no parent menu. If a menu is attached to both a Displayable and another menu, its depth is calculated based on the menu to which it is attached. For instance, a menu A is a sub-menu of menu B and is also attached to a Displayable. If menu B is of depth 3, then menu A will return a depth of 4.

**Returns:**
the depth of this menu within its menu tree
insert

public void insert(int index,
        Command cmd)

Inserts a Command into the Menu just prior to the element specified. The size of the Menu grows by one. The index parameter must be within the range [0..size()], inclusive. If the value of index equals size(), the new element is inserted immediately after the last element; in this case, the effect is identical to calling append().

Parameters:
index - the index of the element where insertion is to occur, starting from zero
        cmd - the command to be inserted

Throws:
        IndexOutOfBoundsException - if index is less than zero or greater than size().
        NullPointerException - if cmd is null
        IllegalArgumentException - if the cmd is already added to this menu.

insert

public void insert(int index,
        Menu menu)

Inserts a Menu into the Menu just prior to the element specified. The size of the Menu grows by one. The sub-menu can only be in one menu at a time. The application is responsible for removing the sub-menu from a previous menu, if any, before it can be inserted into a new menu. The index parameter must be within the range [0..size()], inclusive. If the value of index equals size(), the new element is inserted immediately after the last element; in this case, the effect is identical to calling append().

Parameters:
index - the index of the element where insertion is to occur, starting from zero
        menu - the menu to be inserted

Throws:
        IndexOutOfBoundsException - if index is less than zero or greater than size().
        NullPointerException - if menu is null
        IllegalArgumentException - if the menu already has a parent menu or if it is the top menu in this menu's tree.
        IllegalStateException - if the resulting menu depth exceeds the maximum depth supported by the implementation. The menu depth includes all of the menus above this menu, the menu being inserted, and all of its sub-menus.

isCommand

public boolean isCommand(int index)

Checks if the element at the indicated index is a Command. The index parameter must be within the range [0..size()-1], inclusive. The index of the last element is size()-1

Returns:
true if element is a Command, or false if it is a Menu

Throws:
        IndexOutOfBoundsException - if index is less than zero or greater than or equal to size().

isEnabled

public boolean isEnabled() 

Gets a boolean value indicating whether the Menu is enabled or disabled.

Returns:
state of the Menu: true if enabled, false if disabled
isVisible

public boolean isVisible()

Gets a boolean value indicating whether the Menu is presently visible on any Display.

Note that this method does NOT indicate which Displayable that is currently shown on the Display in focus. Being visible means that the Menu or one of its sub-menus is being browsed.

Returns:
visibility: true if visible, false if invisible

onParentEnabled

public void onParentEnabled(boolean enabled)

Informs the menu of a change in its parent's enabled value. The application must itself decide how the Menu should handle this information. The default is to do nothing.

Since: MIDP 3.0

remove

public void remove(Command cmd)

Removes a command from the menu. The menu's size shrinks by one.

Throws:
IllegalArgumentException - if cmd is not in menu

remove

public void remove(Menu menu)

Removes a sub-menu from the menu. The menu's size shrinks by one. The contents of the removed sub-menu and its sub-menus (if any) are not modified.

Parameters:

menu - the Menu to remove

Throws:
IllegalArgumentException - if menu is not in menu

setEnabled

public void setEnabled(boolean isEnabled)

Sets the Menu into enabled or disabled mode. A Menu which is disabled cannot be navigated into. If a Menu is set to disabled while it is visible, it should immediately be made invisible and focus should be taken back to the level containing this Menu.

Parameters:

isEnabled - true if enabled, false if disabled

setFont

public void setFont(Font font)
javax.microedition.lcdui.Menu

Sets the application's preferred font for rendering the labels for this Menu. Note that the set font is for the Menu object itself, and not for the Commands and Menus it may contain. An menu's font is a hint, and the implementation may disregard the application's preferred font. Note that this method does not influence the fonts used by the contents (commands and sub-menus) of the menu, and these must be set individually using the respective `setFont(Font)` methods.

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:**
- `font` - the preferred font to use to render the Menu's labels

---

**setImage**

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

Sets the Image for this Menu.

**Parameters:**
- `image` - the Image for this menu or `null` if the menu is not to have an Image

---

**setLabel**

```java
public void setLabel(String shortLabel)
```

Sets the short label of the Menu. The label must be not be `null`.

**Parameters:**
- `shortLabel` - the short label

**Throws:**
- `NullPointerException` - if `label` is `null`

---

**setLongLabel**

```java
public void setLongLabel(String longLabel)
```

Sets the long label of the Menu.

**Parameters:**
- `longLabel` - the long label, or `null` if there is no long label

---

**size**

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

Gets the number of Commands and sub-Menus in the Menu.

**Returns:**
- the number of Commands and sub-Menus in the Menu
javax.microedition.lcdui

Notification

Declaration

public class Notification

Object

+--javax.microedition.lcdui.Notification

Description

Represents a small unobtrusive informational note to be shown to the user. All Notifications have a NotificationType that provides the common information for all notifications of the same type, such as a default image and label.

Notifications are used to enable a MIDlet to post notifications to the user, without necessarily taking control of the screen. The Notifications are managed by the implementation, and the only way MIDlets interact with them are by posting and removing Notifications. A typical use of Notification is an email application running in the background, and presenting information about newly arrived messages in a standardized way. Notifications are intended to be used for unobtrusive informational purposes that do not require immediate user attention.

A Notification can exist in one of two logical states: available and removed.

Upon instantiation, a Notification exists in the removed state. A Notification may be created without setting the label and image. If a label and/or image are not provided, the default label and image of the associated NotificationType will be used.

A Notification can have a String label and an Image. The image is used for the visual representation of the notification, with the label optionally being shown by the implementation as a result of the user inspecting the notification. The implementation MAY truncate the label to fit the available size of the displayed text. The image attached to the notification is the primary way that user defined notifications are displayed to the user. The implementation MUST display the image on a best effort basis, depending on what can be done on the particular display. Implementations MAY truncate or scale the icon image based on what's supported by the device. Applications can query the implementation's notification icon size using the Display.getBestImageHeight() and Display.getBestImageWidth() methods using image type Display.NOTIFICATION. A device may show Notifications on multiple displays. In this case, these methods return the values for the primary notification display, and the implementation is expected to handle the other displays in a satisfying way.

A Notification becomes available after a successful call to post(). Depending on the implementation and the NotificationType, the Notification may or may not be directly visible; but the implementation MUST ensure that the user is made aware by visible means that a Notification has been posted. If a NotificationListener is set on a Notification, then after the Notification has been either selected or dismissed by the user, notificationDismissed or notificationSelected is called on the NotificationListener that was set on the Notification.

Implementations SHOULD show the Notification as soon as possible after a post() call, but in the case where many MIDlets post Notifications at the same time, the implementation MAY queue the visual notification for a short time to create a better user experience. A Notification can be updated by calling post() again. This update is then carried out as soon as possible, with the possibility for queuing as described above.

The post(boolean, int) method provides applications with a way to provide a hint to the implementation that the Notification should only be available for a specific length of time (in milliseconds). If the Notification has not been inspected during the requested time, the Notification becomes removed. The specified duration is the time requested by the application for the Notification to be available, and is measured from the time the Notification first became available. Implementations SHOULD use the requested duration but MAY replace it with a system wide preference to better conform to device user preferences.
experience. A Notification that is posted without specifying a duration, with `post(boolean)`, MUST remain *available* to the user until it is removed.

The state diagram below defines how Notifications are posted, removed and re-posted (updated), how the events on NotificationListener are triggered, and how Notifications transition between the *available* and *removed* states.

A Notification can become *removed* in the following ways:

- By the implementation, when a Notification is dismissed or selected by the user, before the callbacks (`notificationSelected()` or `notificationDismissed()`) are triggered.
- By a MIDlet using the `remove()` method.
- By the implementation, due to the duration being passed (see below), which will trigger the `notificationTimeout()` method.

A Notification MUST remain *available* until removed by application or implementation code in the ways defined above.

Since MIDP may be implemented on devices with a wide range of capabilities, the way that Notifications are shown is implementation specific. Notifications may be shown on the primary display, on secondary displays, in a transparent overlay on the main display, or wherever the implementation decides. The MIDlet has no control of how and where the Notifications are shown.

The following example code creates, posts and updates a Notification.
The following example code creates and posts two notifications of the same user defined type.

```java
NotificationType type = new NotificationType("chats", defaultImage);
Notification n = new Notification(type);

n.setImage(image);
n.setLabel("New message from Alice");
n.post();    // notification is made available (label and image must be set)
//
// ... after some time
//
Notification n = new Notification(type, "New message from Bob", image);
n.post(true);
```

**Since:** MIDP 3.0

### Constructor Summary

<table>
<thead>
<tr>
<th>Public Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td><code>Notification(NotificationType type, String label, Image image)</code></td>
<td>Creates a new Notification with the given type, label and image.</td>
</tr>
<tr>
<td>public</td>
<td><code>Notification(NotificationType type, String label)</code></td>
<td>Creates a new Notification with the given type and label.</td>
</tr>
<tr>
<td>public</td>
<td><code>Notification(NotificationType type)</code></td>
<td>Creates a new Notification with the given type.</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Public Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td><code>getTimestamp()</code></td>
<td>Gets the time when the Notification was last posted.</td>
</tr>
<tr>
<td><code>javax.microedition.lcdui.NotificationType</code></td>
<td><code>getType()</code></td>
<td>Get the NotificationType associated with this Notification.</td>
</tr>
<tr>
<td>void</td>
<td><code>post(boolean selectable)</code></td>
<td>Posts the Notification.</td>
</tr>
</tbody>
</table>
void post(boolean selectable, int duration)
Posts the Notification with a set limit on how long it should be available.

void remove()
Removes the Notification.

void setImage(Image image)
Sets the Image of the Notification.

void setLabel(String label)
Sets the label of the Notification.

void setListener(NotificationListener listener)
Sets the NotificationListener of the Notification, replacing any previous NotificationListener.

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

Constructors

Notification
public Notification(NotificationType type)
Creates a new Notification with the given type.

Parameters:
  type - the notification type

Throws:
  NullPointerException - if the type is null

See Also: NotificationType

Notification
public Notification(NotificationType type, String label)
Creates a new Notification with the given type and label.

Parameters:
  type - the Notification type
  label - the Notification's label

Throws:
  NullPointerException - if the type is null

See Also: NotificationType

Notification
public Notification(NotificationType type, String label, Image image)
Creates a new Notification with the given type, label and image.

Parameters:
  type - the notification type
javax.microedition.lcdui.Notification

label - the notification's label
image - the notification's image

Throws:
    NullPointerException - if the type is null

See Also: NotificationType

Methods

getTimestamp

public long getTimestamp()

Gets the time when the Notification was last posted. The time is in the same format as java.lang.System.currentTimeMillis.

Returns:
    the time when the Notification was last posted. If the Notification has not been posted, zero is returned.

getype

public javax.microedition.lcdui.NotificationType getType()

Get the NotificationType associated with this Notification.

Returns:
    the notification type

See Also: NotificationType

post

public void post(boolean selectable)

Posts the Notification. Notifications are not required to have a label and image set. If they are not set, the label and image of the notification type will be used. The time from java.lang.System.currentTimeMillis initializes the value of the Notification.getTimestamp method.

Parameters:
    selectable - true if this Notification must be presented in a way that is selectable by the user. If set to false, the NotificationListener.notificationSelected(Notification) method will not be called for any NotificationListener set on the Notification by (setListener).

Throws:
    NotificationException - If the post() call fails, due to too many active notifications.

See Also: NotificationType

post

public void post(boolean selectable, int duration)

Posts the Notification with a set limit on how long it should be available. Notifications are not required to have a label and image set. If they are not set, the label and image of the notification type will be used. The time from java.lang.System.currentTimeMillis initializes the value of the Notification.getTimestamp method.

Parameters:
    selectable - true if this Notification must be presented in a way that is selectable by the user. If set to false, the NotificationListener.notificationSelected(Notification) method will not be called for any NotificationListener set on the Notification by (setListener).
    duration - the duration of the notification (in milliseconds)
Throws:
   NotificationException - If the post() call fails, due to too many active notifications.
   IllegalArgumentException - If duration is not a positive integer.

See Also: NotificationType

remove
public void remove()

Removes the Notification.

Throws:
   NotificationException - If the Notification could not be removed.
   IllegalStateException - If the Notification is already in the removed state.

setImage
public void setImage(Image image)

Sets the Image of the Notification. The Image may be mutable or immutable. If image is mutable, then a snapshot is taken of the image's contents before the call to post() returns. The snapshot is used whenever the contents of the Notification are to be displayed. If image is already the image of this Notification, a new snapshot of image's contents is taken. For example, after painting into a mutable image contained by a Notification, an application may refresh the snapshot by calling post() again. Example:

```java
notification.setImage(mutableImage);
//...
//mutable image is updated.
//...
notification.post();
```

The updated image will be displayed only after post is called. If the Notification is visible on the display then the display SHOULD be updated with the new snapshot as soon as it is feasible for the implementation to do so.

Parameters:
   image - the Image to set for the Notification; may be null.

setLabel
public void setLabel(String label)

Sets the label of the Notification.

Parameters:
   label - the label for the Notification.

setListener
public void setListener(NotificationListener listener)

Sets the NotificationListener of the Notification, replacing any previous NotificationListener. A null reference is allowed and has the effects of:

- removing any existing listener

Parameters:
javax.microedition.lcdui.Notification

listener - the new NotificationListener for the Notification.
javax.microedition.lcdui

NotificationException

Declaration

public class NotificationException extends RuntimeException

Object
   +-- Throwable
      +-- Exception
         +-- RuntimeException
             +-- javax.microedition.lcdui.NotificationException

Description

Indicates that an operation on a Notification has failed.

Since: MIDP 3.0
See Also: Notification

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public NotificationException()</td>
<td>Constructs an exception with no specified detail message.</td>
</tr>
<tr>
<td>public NotificationException(String s)</td>
<td>Constructs an exception with the specified detail message.</td>
</tr>
</tbody>
</table>

Methods inherited from class Throwable

getMessage, printStackTrace, printStackTrace, toString

Methods inherited from class Object

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

Constructors

NotificationException

public NotificationException()

Constructs an exception with no specified detail message.

NotificationException

public NotificationException(String s)

Constructs an exception with the specified detail message.

Parameters:
s - the detail message
javax.microedition.lcdui

NotificationListener

Declaration

public interface NotificationListener

Description

This interface is used by applications that need to receive events indicating changes in the state of a Notification. An application may provide one or more implementations of NotificationListener (typically by using a nested class or an inner class). Different parts of the application typically have their own NotificationListener to handle their own Notification callbacks.

Before notificationSelected, notificationTimeout or notificationDismissed are called, the Notification MUST be removed by the implementation.

The specification does not require the implementation to create several threads for Notification delivery. Thus, if a NotificationListener method does not return or the return is delayed, the system may be blocked. So, there is the following note to application developers:

- the NotificationListener methods should return immediately.

Since: MIDP 3.0

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void notificationDismissed(Notification notification)</td>
<td>Called when a Notification has been dismissed by the user.</td>
</tr>
<tr>
<td>void notificationSelected(Notification notification)</td>
<td>Called when a Notification has been selected by the user.</td>
</tr>
<tr>
<td>void notificationTimeout(Notification notification)</td>
<td>Called when a Notification has been removed by the system.</td>
</tr>
</tbody>
</table>

Methods

notificationDismissed

public void notificationDismissed(Notification notification)

Called when a Notification has been dismissed by the user. After this method has been called, the Notification will no longer be active in the implementation.

Parameters:
- notification - the Notification being dismissed by the user. The MIDlet must implement the functionality to be performed when the Notification is dismissed.

notificationSelected

public void notificationSelected(Notification notification)

Called when a Notification has been selected by the user. After this method has been called, the Notification will no longer be active in the implementation.
javax.microedition.lcdui.NotificationListener

**Parameters:**
notification - the Notification being responded to by the user. The MIDlet must implement the functionality to be performed when the Notification is selected.

**See Also:** Notification.post(boolean), Notification.post(boolean, int)

---

**notificationTimeout**

```java
public void notificationTimeout(Notification notification)
```

Called when a Notification has been removed by the system. After this method has been called, the Notification will no longer be active in the system. This method is called after the Notification has been removed (due to displayTime being exceeded).

**Parameters:**
notification - the Notification being removed by the device. The MIDlet must implement the functionality to be performed when the Notification is removed.

**See Also:** Notification.post(boolean, int)
javax.microedition.lcdui

**NotificationType**

**Declaration**

```java
public final class NotificationType
```

**Description**

Represents the Notification type (or category) used for grouping, sorting and filtering Notification objects of the same type. A NotificationType can either be a user defined type, or a standard type (defined as constant fields in this class). Notifications associated with any of the standard types are not required to have a label and image.

The primary purpose of notification types is to make it possible for implementations to group, sort and filter notifications. Additionally, the standard notification types enable localization of Notification labels and/or images.

Note that some standard notification types might not be available for all security domains, depending on the security policy enforced by the implementation. An implementation MAY:

- display all notifications independent of the NotificationType
- display Notifications grouped and/or sorted by NotificationType
- display the time and date for Notifications of some types
- choose where and how (in the device user interface) each Notification should be displayed, based on NotificationType
- filter whether Notifications should be displayed or not, based on their NotificationType

Notification types have a default image and label associated with the posted Notification. An application can choose to set its own label and/or image for the posted Notification, which will override the default label and/or image. The default label of a standard Notification type SHOULD support localization.

**Since:** MIDP 3.0

**Field Summary**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static</td>
<td>CALL</td>
<td>The standard notification type for missed calls.</td>
</tr>
<tr>
<td>final</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EMAIL</td>
<td>The standard notification type for email messages.</td>
</tr>
<tr>
<td></td>
<td>IM</td>
<td>The standard notification type for instant messaging messages.</td>
</tr>
<tr>
<td></td>
<td>MMS</td>
<td>The standard notification type for mms messages.</td>
</tr>
<tr>
<td></td>
<td>REMINDER</td>
<td>The standard notification type for reminders (typically calendar reminders,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or task reminders).</td>
</tr>
<tr>
<td></td>
<td>SMS</td>
<td>The standard notification type for sms messages.</td>
</tr>
</tbody>
</table>
**Constructor Summary**

| public NotificationType(String defaultLabel, Image defaultImage) |
| Creates a new user defined NotificationType |

**Methods inherited from class Object**

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

**Fields**

**CALL**

public static final javax.microedition.lcdui.NotificationType CALL

The standard notification type for missed calls.

**EMAIL**

public static final javax.microedition.lcdui.NotificationType EMAIL

The standard notification type for email messages.

**IM**

public static final javax.microedition.lcdui.NotificationType IM

The standard notification type for instant messaging messages.

**MMS**

public static final javax.microedition.lcdui.NotificationType MMS

The standard notification type for mms messages.

**REMINDER**

public static final javax.microedition.lcdui.NotificationType REMINDER

The standard notification type for reminders (typically calendar reminders, or task reminders).

**SMS**

public static final javax.microedition.lcdui.NotificationType SMS

The standard notification type for sms messages.

**Constructors**

**NotificationType**

public NotificationType(String defaultLabel, Image defaultImage)

Creates a new user defined NotificationType
javax.microedition.lcdui.NotificationType

Parameters:
  defaultLabel - The default label of this notification type. This label will be used for associated notifications, if no label is set on the notification.
  defaultImage - The default image of this notification type. This image will be used for associated notifications, if no image is set on the notification. If the image is mutable, a snapshot of it will be taken.

Throws:
  NullPointerException - if defaultLabel or defaultImage is null
javax.microedition.lcdui

ScalableImage

Declaration

```java
public class ScalableImage extends Image

Object
   +--javax.microedition.lcdui.Image
   +--javax.microedition.lcdui.ScalableImage
```

Description

A ScalableImage object encapsulates vector graphics content. The enclosed vector content can be used directly in high-level LCDUI components, and can also be used (in a rasterized, bitmap form) as an Image of variable dimensions in low-level graphics APIs.

Vector graphics content has a viewport to define the area within its own coordinate system that should be rendered. Typically, the viewport region is defined by the content author (e.g. using the viewBox attribute in SVG) and is sized to contain all of its drawing operations. If a viewport is not explicitly defined in the content, the implementation may examine the drawing operations to determine a suitable viewport; if such an examination is unfeasible or not supported by the implementation, a default viewport of (0, 0, 100, 100) may be used instead. The viewport dimensions may be obtained using the `getViewportWidth()` and `getViewportHeight()` methods.

As a subclass of Image, a ScalableImage also has a pixel-based width and height, as reported by the `getWidth()` and `getHeight()` methods. For a ScalableImage, these values indicate its rasterized size when used as a pixel-based Image in low-level graphics API's, and they are initially set to match the content's viewport dimensions. The pixel dimensions of the image may be subsequently modified using the `setWidth()` and `setHeight()` methods, thus changing the size of the rasterized Image that it represents; however, it is important to note that such changes do not alter the viewport of the content itself. The initial frame of the content is rasterized if the content includes animations.

When used with high-level UI components, the implementation SHOULD use the vector-based content directly and render it at a size that is appropriate for the given component. The size may be determined based on the content's viewport size and the preferred image dimensions for the Display. Support for animation is OPTIONAL and interactive functionality MAY be disabled. The pixel-based dimensions of a ScalableImage may be ignored when it is used in this manner.

Creating ScalableImages

A ScalableImage instance can be created by calling one of the following factory methods in the Image class with valid vector graphics content (e.g. SVG Tiny 1.1):

- `createImage(byte[] data, int imageOffset, int imageLength)`
- `createImage(InputStream in)`
- `createImage(String resource)`

All ScalableImage objects are immutable and cannot be drawn on with a Graphics object; calls to `isMutable()` will return false and an attempt to obtain a Graphics Object will result in an IllegalStateException being thrown. The LCDUI does not provide any capabilities to modify the content of the ScalableImage. Any modifications to the vector-based content are done through an auxiliary vector graphics API like M2G, if the ScalableImage has been created using the `bind(Object)` method.
Creating Pixel-Based Images From ScalableImages

As stated above, a ScalableImage itself can be used directly as a regular pixel-based Image for low-level graphics APIs. However, it is also possible create a pixel-based Image instance from a ScalableImage by using it as the source for the following factory methods in Image:

- `createImage(Image source)`
- `createImage(Image source, int x, int y, int width, int height, int transform)`
- `createImage(Image source, int x, int y, int width, int height, int transform, int img_width, int img_height)`

Use of ScalableImage with Sprite and TiledLayer

A ScalableImage may be used with the Sprite and TiledLayer class in the Game API. However, since these APIs assume fixed image dimensions, the implementation must behave as if a fixed-size, bitmapped snapshot of the ScalableImage is taken when the relevant method is invoked. That is, the following code:

```
new Sprite(myScalableImage);
```

must behave as:

```
new Sprite(Image.createImage(myScalableImage));
```

Thus, subsequent changes to the dimensions or content of the ScalableImage do not effect the size or appearance of the Sprite or TiledLayer.

**Since:** MIDP 3.0

---

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static javax.microedition.lcdui.ScalableImage bind(Object extScalableImage)</code></td>
<td>Binds a ScalableImage instance to a more advanced scalable image object from an external graphics API.</td>
</tr>
<tr>
<td>int <code>getViewportHeight()</code></td>
<td>Gets the height of the content's viewport.</td>
</tr>
<tr>
<td>int <code>getViewportWidth()</code></td>
<td>Gets the width of the content's viewport.</td>
</tr>
<tr>
<td>void <code>setHeight(int height, boolean matchAspectRatio)</code></td>
<td>Sets the pixel-based height of the ScalableImage, as returned by getHeight.</td>
</tr>
</tbody>
</table>
#### javax.microedition.lcdui.ScalableImage

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td><code>setWidth(int width, boolean matchAspectRatio)</code></td>
<td>Sets the pixel-based width of the ScalableImage, as returned by <code>getWidth</code>.</td>
</tr>
<tr>
<td>void</td>
<td><code>unbind()</code></td>
<td>Unbinds this ScalableImage instance from an external scalable image object.</td>
</tr>
</tbody>
</table>

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

- `createImage`, `createImage`, `createImage`, `createImage`, `createImage`, `createImage`, `createImage`, `createImage`, `createRGBImage`, `getARGB16`, `getGraphics`, `getHeight`, `getRGB`, `getRGB16`, `getWidth`, `hasAlpha`, `isAnimated`, `isMutable`, `isScalable`

**Methods inherited from class** `Object`

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

### Methods

#### bind

```java
public static javax.microedition.lcdui.ScalableImage bind(Object extScalableImage)
```

Binds a ScalableImage instance to a more advanced scalable image object from an external graphics API. This method creates and returns a new scalable image instance that can be used for drawing within LCDUI. If this method is called again for the same target, it MUST return the same ScalableImage instance that was returned from the initial call to this method.

For example, M2G has the `javax.microedition.m2g.ScalableImage` class. This method allows to get a `javax.microedition.lcdui.ScalableImage` representation out of `javax.microedition.m2g.ScalableImage`.

As long as the binding is in effect (i.e. the `unbind` method has not been called), modifications to `extScalableImage` MUST be visible directly in the `ScalableImage` object returned from the method.

**Parameters:**
- `extScalableImage` - the external scalable image object to bind to

**Returns:**
- `ScalableImage` instance bound to external API scalable image object

**Throws:**
- `java.lang.NullPointerException` - if `target` is `null`
- `java.lang.IllegalArgumentException` - if `target` is `invalid`

#### getViewportHeight

```java
public int getViewportHeight()
```

Gets the height of the content's viewport.

**Returns:**
- the viewport height

#### getViewportWidth

```java
public int getViewportWidth()
```
Gets the width of the content's viewport.

**Returns:**
the viewport width

---

**setHeight**

```java
public void setHeight(int height, boolean matchAspectRatio)
```

Sets the pixel-based height of the ScalableImage, as returned by `getHeight`. This dimension only applies when the image is used as a pixel-based `Image` in low-level graphics APIs such as `drawImage`.

If `matchAspectRatio` is true, the pixel-based width is automatically changed to match the aspect ratio of the content's viewport given the new height. Mathematically:

\[
\text{width} = \frac{\text{viewport}_\text{width} \times \text{height}}{\text{viewport}_\text{height}}
\]

**Parameters:**
- `height` - The new pixel-based height of the image, in pixels
- `matchAspectRatio` - true if the width should be changed to match the content's aspect ratio

**Throws:**
- `IllegalArgumentException` - if height is not a positive value

---

**setWidth**

```java
public void setWidth(int width, boolean matchAspectRatio)
```

Sets the pixel-based width of the ScalableImage, as returned by `getWidth`. This dimension only applies when the image is used as a pixel-based `Image` in low-level graphics APIs such as `drawImage`.

If `matchAspectRatio` is true, the pixel-based height is automatically changed to match the aspect ratio of the content's viewport given the new width. Mathematically:

\[
\text{height} = \frac{\text{viewport}_\text{height} \times \text{width}}{\text{getViewportWidth()}}
\]

**Parameters:**
- `width` - The new pixel-based width of the image, in pixels
- `matchAspectRatio` - true if the height should be changed to match the content's aspect ratio

**Throws:**
- `IllegalArgumentException` - if width is not a positive value

---

**unbind**

```java
public void unbind()
```


Unbinds this `ScalableImage` instance from an external scalable image object. From the point `unbind()` is called, modifications to the previously bound external scalable image will no longer be reflected in this `ScalableImage` instance. The data and attributes of the `ScalableImage` reflect the state of the external scalable image at the point `unbind()` was called. If the instance is not bound to an external object, this method does nothing.

See Also: `bind(Object)`
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.FileSelector, javax.microedition.lcdui.Form,
javax.microedition.lcdui.List, javax.microedition.lcdui.TabbedPane,
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, getCommand, getCommandLayoutPolicy, getCommands, getCurrentDisplay,
getHeight, getMenu, getTicker, getTitle, getWidth, invalidateCommandLayout,
isShown, removeCommand, removeCommandOrMenu, setCommand, setCommandLayoutPolicy,
setCommandListener, setMenu, 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 IllegalArgumentException whenever it is called.

void setDefaultCloseOperation(Command cmd)

Spacers are restricted from having Commands, so this method will always throw IllegalArgumentException whenever it is called.

void setLabel(String label)

Spacers are restricted to having null labels, so this method will always throw IllegalArgumentException whenever it is called.

void setMinimumSize(int minWidth, int minHeight)

Sets the minimum size for this spacer.
Methods inherited from class `javax.microedition.lcdui.Item`

getCommands, getLabel, getLayout, getLayoutHint, getMinimumHeight, getMinimumWidth, getPreferredHeight, getPreferredWidth, notifyStateChanged, removeCommand, setCommand, setItemCommandListener, setLayout, setLayoutHint, setPreferredSize

Methods inherited from class `Object`

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

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

javax.microedition.lcdui.StringItem

Description

An item that can contain a string. A StringItem is display-only; the user cannot edit the contents. Both the label and the textual content of a StringItem may be modified by the application. The visual representation of the label may differ from that of the textual contents.

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

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.

double getFontHeight()
Gets the height of the font used by this StringItem.

javax.microedition.lcdui.Font getFont acquitted()
Returns the font used by this StringItem.

javax.microedition.lcdui.Font getFont acquitted acquitted()
Returns the font used by this StringItem. The acquitted acquitted is ignored.

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
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: 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:

StringItem strItem =
    new StringItem("Default: ", "Set",
        Item.BUTTON);
strItem.setDefaultCommand(
    new Command("Set", Command.ITEM, 1);
// icl is ItemCommandListener
strItem.setItemCommandListener(icl);
javax.microedition.lcdui.StringItem

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 Item.HYPERLINK.

Throws:
- IllegalArgumentException - if appearanceMode invalid

Since: MIDP 2.0

Methods

getAppearanceMode

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

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: setFont(Font)

Since: MIDP 2.0

getText

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: setText(String)

setFont

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: 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:** `getText()`
javax.microedition.lcdui
TabbedPane

Declaration

public class TabbedPane extends Screen

Object

javax.microedition.lcdui.Displayable

javax.microedition.lcdui.Screen

javax.microedition.lcdui.TabbedPane

Description

TabbedPane is a Screen subclass that presents a series of Screens to the users and allows them to navigate between screens by selecting the corresponding tab.

Each tab has a Screen object that is its contents. Only Form and List objects can be added as the contents for a tab. Another TabbedPane cannot be used as the contents for a tab. A Screen MUST only be placed on one tab at a time.

Each tab must include an icon Image object or text to indicate its content. Therefore, the TabbedPane class has two modes to represent the contents of the tabs.

1) String mode In String mode, Strings are used to represent each tab's contents. The tab's name is the title of the tab content, which can be null. In this mode, TabbedPane's title area always displays TabbedPane's title to avoid redundancy.

2) Image mode In Image mode, icons represent each tab's contents. The Image may be mutable or immutable; if the Image is mutable, the effect is as if a snapshot of the Image is taken at the time the TabbedPane is constructed or the tab is added to the TabbedPane. Subsequent drawing operations performed on the Image will not automatically appear in the tab. Explicitly calling #setTabIcon with the same Image will cause the icon to be refreshed with the Image's current contents.

Implementations may truncate or scale the icon image if it is larger than the size supported by device. Applications can query the implementation's tab icon size by calling Display.getBestImageWidth(int) and Display.getBestImageHeight(int) methods using the Display.TAB image type. The style and appearance of tabs are platform-dependent.

The number of tabs is not limited and may exceed the number that can be shown on the screen at one time. In that case, implementations must indicate to users that more tabs are available and provide a mechanism for accessing them. An application can be notified of tab navigation events on a given TabbedPane using the TabListener interface.

In Image mode, the title shown for the TabbedPane depends on the suppressTitle parameter. In String mode, TabbedPane's title is always shown to avoid redundancy.
Similarly, the TabbedPane’s Ticker (if present) takes precedence over the Ticker belonging to the selected tab’s contents.

Commands and Menus may be added to the TabbedPane and to the contents of its tabs. The Commands and Menus shown to the user include those belonging to the TabbedPane and those belonging to the selected tab’s contents. A Command or Menu is shown only once even if it belongs to both objects. If such a Command is invoked, both objects' CommandListeners are notified; the listener for the tab’s content is notified first, followed by that of the TabbedPane.

A tab is removed from the tabbed pane using `removeTab(index)`. If a tab at index `n` is removed, then the implementation must select the tab at index `n-1`. If tab at index `n-1` does not exist, then tab at index `n+1` must be selected. Therefore, `getSelectedIndex()` after a `removeTab()` will return the next selected tab. If the `removeTab(index)` removes the last tab in the tabbedpane, then no tabs are selected and the `getSelectedIndex()` will return `-1`.

Calling `setCurrent` on the TabbedPane content MUST set the focus on the content. Calling `setCurrent` on the TabbedPane MUST set the focus on the first tab element, not the content. Calling `setFocus` MUST set the focus on the tab element indicated by the given index parameter. By default the focus MUST be on the first tab element.

The mechanism used by the user to navigate between tabs is implementation dependent. However, it must allow for the user to navigate within the contents of each tab and between tabs. TabbedPane implementations on keypad-based devices SHOULD support vertical and horizontal traversals in the follow manner.

<table>
<thead>
<tr>
<th>Initial State</th>
<th>Down Key press</th>
<th>Up key press</th>
<th>Left/Right key press</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus on tab (tab icon)</td>
<td>Focus is set into the tab content</td>
<td>None or platform specific implementation.</td>
<td>Change tab focus to next/prior tab. The set of commands shown to the user changes to those belonging to the TabbedPane and those belonging to the newly selected tab’s contents. A TabListener.tabChangeEvent is generated.</td>
</tr>
<tr>
<td>Focus into the tab content (Screen object)</td>
<td>Default traversal already implemented on both Form and List classes.</td>
<td>Focus is set on tab (tab icon). This behavior supposes that the topmost focusable item is selected on tabbed pane content. If TabbedPane content has items that cannot get the focus, the traversal with hardware keys SHOULD be implemented so that if up key is pressed then the focus SHOULD move to the tab even if there are unfocusable items in-between.</td>
<td>Default traversal already implemented on either Form and List classes or specific device implementation. If there is no horizontal traversal in the tab content, the left and right keys SHOULD be reserved for traversing between tabs, and up and down keys for traversing inside the tab content.</td>
</tr>
</tbody>
</table>
## Constructor Summary

```java
public TabbedPane(String title, boolean stringTab, boolean suppressTitle)
    Creates a new, empty TabbedPane, specifying its title.
```

## Method Summary

```java
void addTab(Screen tab, Image icon)
    Adds a tab element to the TabbedPane as the last element.

void addTabListener(TabListener tabListener)
    Sets a listener for focus change to this TabbedPane, replacing any previous TabListener.

int getCount()
    Gets the number of tab elements in the TabbedPane.

int getHeight()
    Returns the height in pixels of the area available for Displayables added to the TabbedPane.

djavax.microedition.lcdui.Screen getScreen(int index)
    Gets the content for the tab references by the given index.

int getSelectedIndex()
    Returns the index number of a tab element (current tab) in the TabbedPane.

djavax.microedition.lcdui.Image getTabIcon(int index)
    Gets the Image for the tab referenced by the given index.

int getWidth()
    Returns the width in pixels of the area available for Displayables added to the TabbedPane.

void insertTab(int index, Screen tab, Image icon)
    Inserts a tab element into the TabbedPane just prior to the element specified.

void removeTab(int index)
    Removes a tab element from TabbedPane.

void setFocus(int index)
    Sets the focus on a tab element.

void setTabIcon(int index, Image icon)
    Sets the Image part of the tab element referenced by index, replacing the previous image of the tab.
```

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

- `addCommand`, `getCommand`, `getCommandLayoutPolicy`, `getCommands`, `getCurrentDisplay`, `getMenu`, `getTicker`, `getTitle`, `invalidateCommandLayout`, `isVisible`, `removeCommand`, `removeCommandOrMenu`, `setCommand`, `setCommandLayoutPolicy`, `setCommandListener`, `setMenu`, `setTicker`, `setTitle`, `sizeChanged`

### Methods inherited from class `Object`
javax.microedition.lcdui.TabbedPane

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

Constructors

TabbedPane

public TabbedPane(String title,
                   boolean stringTab,
                   boolean suppressTitle)

Creates a new, empty TabbedPane, specifying its title. If the TabbedPane is in "String mode", the
suppressTitle argument is ignored.

Parameters:
  title - the screen's title (see Displayable).
  stringTab - a boolean value representing the mode of the TabbedPane
              true - indicates that the TabbedPane is in "String mode"
              false - indicates that the TabbedPane is in "Image mode"
  suppressTitle - a boolean value representing the state of the screen's title.
                true - indicates that title will be set according to tab element's
                title (current tab), ignoring any title set on the TabbedPane object.
                false - indicates that title will be fixed, set on the TabbedPane object.

Since: MIDP 3.0

Methods

addTab

public void addTab(Screen tab,
                    Image icon)

Adds a tab element to the TabbedPane as the last element. If the TabbedPane is in "String mode",
the icon argument is ignored.

Parameters:
  tab - screen object (tab element) to be added.
  icon - image part of the tab element.

Throws:
  NullPointerException - if the icon is null and the TabbedPane is in "Image mode", or if the
                         tab is null in any mode
  IllegalArgumentException - if the screen is not an instance of Form, or List
  IllegalStateException - if the tab has already been placed on this or another TabbedPane
                          IllegalStateException - if the tab is current on another Display

Since: MIDP 3.0

addTabListener

public void addTabListener(TabListener tabListener)

Sets a listener for focus change to this TabbedPane, replacing any previous TabListener. A null
reference is allowed and has the effect of removing any existing listener.

Parameters:
  tabListener - The new listener or null.

Since: MIDP 3.0

getCount

public int getCount()
getHeight

public int getHeight()

Returns the height in pixels of the area available for Displayables added to the TabbedPane. This area excludes the tabs themselves and any borders that are drawn around the added Displayables. The height returned is the maximum height that getHeight and getHeight can return if the Form or List is in focus within the TabbedPane.

Returns: the height in pixels available to display a Form or a List

Since: MIDP 3.0

getScreen

public javax.microedition.lcdui.Screen getScreen(int index)

Gets the content for the tab references by the given index. The index parameter must be within the range [0..getCount()-1], inclusive.

Parameters:
index - the index number of the tab to be returned, starting from zero.

Returns: a Screen object (tab element) in the TabbedPane referenced by index, or null if index is invalid.

Throws: IndexOutOfBoundsException - if index is not within the range [0..getCount()-1], inclusive.

Since: MIDP 3.0

getSelectedIndex

public int getSelectedIndex()

Returns the index number of a tab element (current tab) in the TabbedPane.

Returns: index of selected tab, or -1 if none.

Since: MIDP 3.0

getTabIcon

public javax.microedition.lcdui.Image getTabIcon(int index)

Gets the Image for the tab referenced by the given index. If the TabbedPane is in "String mode", this always returns null. The index parameter must be within the range [0..getCount()-1], inclusive.

Parameters:
index - the index number of the tab to be queried.

Returns: the image part of the tab referenced by index, or null if index is invalid or if the TabbedPane is in the "String mode".

Throws: IndexOutOfBoundsException - if index is not within the range [0..getCount()-1], inclusive.
**Since:** MIDP 3.0

**getWidth**

```java
public int getWidth()
```

Returns the width in pixels of the area available for Displayables added to the TabbedPane. This area excludes the tabs themselves and any borders that are drawn around the added Displayables. The width returned is the maximum width that getWidth can return if the Form or List is in focus within the TabbedPane.

Returns:
the width in pixels available to display a Form or a List

**Since:** MIDP 3.0

**insertTab**

```java
public void insertTab(int index,
                      Screen tab,
                      Image icon)
```

Inserts a tab element into the TabbedPane just prior to the element specified. The size of the TabbedPane grows by one. If the TabbedPane is in "String mode", the icon argument is ignored. The index parameter must be within the range [0..ItemCount], inclusive. If the value of ItemCount() is used for the index, the new element is inserted immediately after the last element. In this case, the effect is identical to addTab().

Parameters:
- index - the index of the tab where insertion is to occur, starting from zero.
- tab - the tab object (Screen) to be inserted.
- icon - the image part of the tab element.

Throws:
- IndexOutOfBoundsException - if index is not within the range [0..ItemCount()], inclusive.
- NullPointerException - if either tab or icon is null while the TabbedPane is in "Image mode"
- IllegalArgumentException - if the screen is not an instance of Form, or List
- IllegalStateException - if the tab has already been placed on this or another TabbedPane
- IllegalStateException - if the tab is current on another Display

**Since:** MIDP 3.0

**removeTab**

```java
public void removeTab(int index)
```

Removes a tab element from TabbedPane. If the tab element is being shown on the display, the implementation should update the display as soon as it is feasible to do so. The index parameter must be within the range [0..ItemCount()-1], inclusive.

Parameters:
- index - the index number of the tab to be removed.

Throws:
- IndexOutOfBoundsException - if index is not within the range [0..ItemCount()-1], inclusive.

**Since:** MIDP 3.0

**setFocus**

```java
public void setFocus(int index)
```

Sets the focus on a tab element. The index parameter must be within the range [0..ItemCount()-1], inclusive.

Parameters:
- index - the index of the tab element to receive the focus.
Throws:
    IndexOutOfBoundsException - if index is not within the range [0..getCount()-1], inclusive.

Since: MIDP 3.0

**setTabIcon**

```java
public void setTabIcon(int index,
    Image icon)
```

Sets the Image part of the tab element referenced by index, replacing the previous image of the tab. This method may also be used to update a tab that was created using a mutable Image; this method should be called with the same Image after its contents have been changed to update the appearance of the tab. If the TabbedPane is in “String mode”, then the icon argument is ignored. The index parameter must be within the range [0..getCount()-1], inclusive.

Parameters:
   index - the index of the tab element to be set.
   icon - the new image of the tab element.

Throws:
    IndexOutOfBoundsException - if index is not within the range [0..getCount()-1], inclusive.
    NullPointerException - if icon is null while the TabbedPane is in "Image mode"

Since: MIDP 3.0
DECLARATION

public final class TableLayoutPolicy extends FormLayoutPolicy

DESCRIPTION

TableLayoutPolicy displays the Items in a Form aligned in columns. The Items in the Form are placed in columns filling rows either from left-to-right or right-to-left depending on the platform's layout direction. When each column in a row is full, the next Item is placed in the next row. If there are not enough Items to fill the last row the cells are left empty. All but the last row will have an Item in every column.

The width of each column is determined based on the minimum and preferred width of items given the layout directives for shrink and expand. When calculating table column width, the smallest column width is used that accommodates every item in the column. The minimum width of each item in a column is the item's minimum width, if it has the layout directive LAYOUT_SHRINK, otherwise it is the item's preferred width. The column's width is the largest of those widths. When the sum of the column widths is less than the width of the display, the additional space is evenly distributed to columns that contain any item with LAYOUT_EXPAND. Individual Items are expanded to the column width if they have the layout directive LAYOUT_EXPAND.

The height of each row is determined based on the minimum and preferred height of items given the layout directives for vertical shrink and expand. When calculating table row height, the smallest row height is used that accommodates every item in the row. The minimum height of each item in a row is the item's minimum height, if it has the layout directive LAYOUT_VSHRINK, otherwise it is the item's preferred height. The row's height is the largest of those heights. Individual Items are expanded to the row height if they have the layout directive LAYOUT_VEXPAND.

The layout directives from each Item.getLayout are used as follows:

Item.LAYOUT_SHRINK
The item's initial width is its minimum width and is expanded to as much as its preferred width if additional horizontal space is available. Otherwise, the preferred width is used. See LAYOUT_SHRINK for the details of the computation.

Item.LAYOUT_EXPAND
The item may be expanded with its share of the available horizontal space. See LAYOUT_EXPAND for the details of the computation.

Item.LAYOUT_VSHRINK
The item's initial height is its minimum height and is expanded to as much as its preferred height if additional vertical space is available. See LAYOUT_VSHRINK for the details of the computation.

Item.LAYOUT_VEXPAND
The item may be expanded with its share of the available vertical space. See LAYOUT_VEXPAND for the details of the computation.

Item.LAYOUT_LEFT
Is ignored.

Item.LAYOUT_CENTER
Is ignored.

Item.LAYOUT_RIGHT
Is ignored.

Item.LAYOUT_TOP
Is ignored.

Item.LAYOUT_BOTTOM
Is ignored.
Item.LAYOUT_VCENTER
Is ignored.
Item.LAYOUT_NEWLINE_BEFORE
Is ignored.
Item.LAYOUT_NEWLINE_AFTER
Is ignored.

TableLayoutPolicy only uses the layout directives above and does not introduce any ItemLayoutHints.

**Since:** MIDP 3.0

**See Also:** FormLayoutPolicy

### Fields inherited from class [javax.microedition.lcdui.FormLayoutPolicy](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ javax-microedition-lcdui/)

- DIRECTION_LTR, DIRECTION_RTL

### Constructor Summary

**public** TableLayoutPolicy(Form form, int columns)

Creates a new instance of TableLayoutPolicy.

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>doLayout(int viewportX, int viewportY, int viewportWidth, int viewportHeight, int[] totalSize)</td>
<td>Compute or update the position and size of each Item in the Form.</td>
</tr>
<tr>
<td>int</td>
<td>getColumns()</td>
<td>Gets the number of columns in this TableLayoutPolicy.</td>
</tr>
<tr>
<td>[javax.microedition.lcdui.Item](<a href="https://docs.oracle.com/en/middleware/javase/11/javase-platform/">https://docs.oracle.com/en/middleware/javase/11/javase-platform/</a> javax-microedition-lcdui/)*</td>
<td>getTraverse(Item item, int dir)</td>
<td>Gets the Item logically adjacent to an existing Item in the traversal direction.</td>
</tr>
</tbody>
</table>

### Methods inherited from class [javax.microedition.lcdui.FormLayoutPolicy](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ javax-microedition-lcdui/)

- [getForm](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ javax-microedition-lcdui/)
- [getHeight](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ javax-microedition-lcdui/)
- [getLayoutDirection](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ javax-microedition-lcdui/)
- [getWidth](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ javax-microedition-lcdui/)
- [getX](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ javax-microedition-lcdui/)
- [getY](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ javax-microedition-lcdui/)
- [isValid](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ javax-microedition-lcdui/)
- [setPosition](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ javax-microedition-lcdui/)
- [setSize](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ javax-microedition-lcdui/)
- [setValid](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ javax-microedition-lcdui/)

### Methods inherited from class [Object](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ java-lang/)

- [equals](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ java-lang/)
- [getClass](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ java-lang/)
- [hashCode](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ java-lang/)
- [notify](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ java-lang/)
- [notifyAll](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ java-lang/)
- [toString](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ java-lang/)
- [wait](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ java-lang/)
- [wait](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ java-lang/)
- [wait](https://docs.oracle.com/en/middleware/javase/11/javase-platform/ java-lang/)

### Constructors

**public** TableLayoutPolicy(Form form, int columns)

Creates a new instance of TableLayoutPolicy. Items are arranged in a fixed number of columns.

**Parameters:**
- form - the Form bound to this table layout policy.
- columns - the number of columns for the Table layout.

**Throws:**
- IllegalArgumentException - if columns is less than 1.
- NullPointerException - if form is null.
Methods

doLayout

```java
protected void doLayout(int viewportX,
            int viewportY,
            int viewportWidth,
            int viewportHeight,
            int[] totalSize)
```

Compute or update the position and size of each Item in the Form. Items are placed in rows based on the number of columns. The horizontal order of the items in the Form is determined by the layout direction of the platform. The vertical direction is always top to bottom. The width of each column is determined based on the minimum and preferred width of items given the layout directives for shrink and expand.

The algorithm makes a first pass over the items to determine the width of each column. Then the height of each row is determined. Another pass is used to set the width, height, and validity of each item in the column and to compute and set the height.

**Parameters:**
- `viewportX` - The x offset of the viewport on the form.
- `viewportY` - The y offset of the viewport on the form.
- `viewportWidth` - The width of the viewport.
- `viewportHeight` - The height of the viewport.
- `totalSize` - An output parameter to be set to the full width and height required for all items in the form. The width is stored in `totalSize[0]` and the height is stored in `totalSize[1]`.

**Throws:**
- `ArrayIndexOutOfBoundsException` - is thrown if the length of the `totalSize` array is less than 2.

**Since:** MIDP 3.0

getColumns

```java
public int getColumn()
```

Gets the number of columns in this TableLayoutPolicy. The number of columns is fixed when the TableLayoutPolicy is created.

**Returns:**
- the number of columns.

**Since:** MIDP 3.0

getTraverse

```java
protected javax.microedition.lcdui.Item getTraverse(Item item,
                  int dir)
```

Mobile Information Device Profile v3.0 - JSR 271
Java Community Process - Final Release
© Copyright Motorola, Inc. 2002-2009. All Rights Reserved.
javax.microedition.lcdui.TableLayoutPolicy

Since: MIDP 3.0

Gets the Item logically adjacent to an existing Item in the traversal direction.

The next item depends on the traversal direction requested:

- **Canvas.UP**: The item in the same column in the previous row wrapping around as necessary from the first row to the highest row that contains an item in that column.
- **Canvas.DOWN**: The item in the same column in the next row wrapping around as necessary from the highest row that contains an item in that column to the first row.
- **Canvas.LEFT**: For left-to-right layout direction, this is the previous item in the sequence of items, wrapping around from zero to `form.size()-1` as necessary. For right-to-left layout direction, this is the next item in the sequence of items, wrapping around from `form.size()-1` to zero as necessary.
- **Canvas.RIGHT**: For left-to-right layout direction, this is the next item in the sequence of items, wrapping around from `form.size()-1` to zero as necessary. For right-to-left layout direction, this is the previous item in the sequence of items, wrapping around from 0 to `form.size()-1` as necessary.

**Parameters:**
- `item` - a current Item
- `dir` - the traversal direction from the item to the adjacent item

**Returns:**
the item in the traversal direction requested; may be null.

** Throws:**
IllegalArgumentException - if `dir` is not one of `Canvas.UP, Canvas.DOWN, Canvas.LEFT, or Canvas.RIGHT`.

© Copyright Motorola, Inc. 2002-2009. All Rights Reserved.
javax.microedition.lcdui

TabListener

Declaration

public interface TabListener

Description
This interface is used to receive events related to changes on a TabbedPane.

Since: MIDP 3.0

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabAddedEvent(int index, Screen tab)</td>
<td>Indicates that a new tab is added.</td>
</tr>
<tr>
<td>tabChangeEvent(Screen tab)</td>
<td>Indicates that a new tab is receiving the focus (being selected).</td>
</tr>
<tr>
<td>tabRemovedEvent(int index)</td>
<td>Indicates that a tab is removed.</td>
</tr>
</tbody>
</table>

Methods

**tabAddedEvent**

public void **tabAddedEvent**(int index, Screen tab)

Indicates that a new tab is added.

Parameters:

- tab - the tab that is added.
- index - the index of the tab where the insertion is to occur.

Since: MIDP 3.0

**tabChangeEvent**

public void **tabChangeEvent**(Screen tab)

Indicates that a new tab is receiving the focus (being selected).

Parameters:

- tab - the tab that is receiving the focus.

Since: MIDP 3.0

**tabRemovedEvent**

public void **tabRemovedEvent**(int index)

Indicates that a tab is removed.

Parameters:
javax.microedition.lcdui.TabListener

index - the index number of the removed tab.

Since: MIDP 3.0
javax.microedition.lcdui

Text

Description

The Text class is used to layout and render text within a specific area. Text supports:

- Alignment of each line to left, center, right or locale default
- Justification of each line
- Bidirectional layout based on Unicode and locale
- Default font for all characters
- Per character font to supersede default font for regions of characters
- Query of position and extent of each character
- Query of index of character at position
- Query of characters displayed
- Insert and delete of characters
- Spacing above and below lines
- Indentation
- Scrolling within bounding box
- Support for a caret
- Support for highlight region
- Support for setting the color of regions

The Text class encapsulates the text, its formatting attributes, and a rectangular area within which it is to be rendered. The text contents are laid out within the bounding box area accordingly, and the locations of specific characters can be queried.

The text can display a caret indicating a character. The caret, if present, MUST be drawn when the Text is drawn using the foreground Color. The application may set and reset the caret position before it draws the Text to achieve visual effects such as blinking the caret. The caret is only moved using the setCaret and moveCaret methods since the Text instance has no access to any input such as keys or pointer. The text can highlight a range of characters. The Text can set the font and color of regions of characters. The Text can set the background color for the region, by default the background is transparent.

The Text object MUST support directional text appropriate for all locales supported by the device. The Unicode 3.0 standard defines a standard nomenclature and algorithm for layout of bi-directional text. Strings passed to this Text are in logical order. The implementation of Text is responsible for displaying the characters in the appropriate direction for the locale and Unicode 3.0 encodings. The initial display direction can be set as Left-To-Right, Right-To-Left, or neutral. If neutral, the device or sequence of characters determines the initial direction. The alignment of text on each line can be set to left, right, centered, justified or default. The default alignment MUST match the default initial display direction. The alignment changes to the opposite (right or left) if the first strong directional character in the paragraph changes the initial text direction.

A Text object contains a sequence of text to be drawn within a bounding box. When drawn on a graphics context all of the characters fully visible or partially visible within the bounding box are drawn. Scrolling is supported with a scroll offset. The scroll offset shifts the text to be displayed up within the bounding box. The scroll offset affects all of the mappings from locations to characters within the text and vis-a-versa. The Text does not draw any indication of the scrolling offset or scrollbars; if needed by the application, it must draw them itself.
Implementation note: Modifying the attributes of a Text object may necessitate an update to the layout of the text. While implementations are not required to recompute the layout immediately, they MUST ensure that the layout is up to date prior to performing any operation that is dependent on the layout. For example, if the text and font are changed, the layout does not need to be updated immediately. However, if an attempt is made to render this Text or query the location of a character, the implementation must ensure that the layout is recomputed prior to processing these requests.

**Since:** MIDP 3.0  
**See Also:** `Graphics.drawText(Text, int, int)`

## Field Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALIGN_CENTER</td>
<td>1</td>
<td>Center each line of text horizontally within this Text.</td>
</tr>
<tr>
<td>ALIGN_DEFAULT</td>
<td>4</td>
<td>Align each line using the default.</td>
</tr>
<tr>
<td>ALIGN_JUSTIFY</td>
<td>3</td>
<td>Justify the each line both left and right.</td>
</tr>
<tr>
<td>ALIGN_LEFT</td>
<td>0</td>
<td>Align each line of text to the left edge of this Text.</td>
</tr>
<tr>
<td>ALIGN_RIGHT</td>
<td>2</td>
<td>Align each line of text to the right edge of this Text.</td>
</tr>
<tr>
<td>DIRECTION_LTR</td>
<td>10</td>
<td>Initial Left-to-Right display order.</td>
</tr>
<tr>
<td>DIRECTION_NEUTRAL</td>
<td>12</td>
<td>Initial neutral display order.</td>
</tr>
<tr>
<td>DIRECTION_RTL</td>
<td>11</td>
<td>Initial Right-to-Left display order.</td>
</tr>
</tbody>
</table>

## Constructor Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text()</td>
<td>Creates a new Text object.</td>
</tr>
<tr>
<td>Text(String contents, int width, int height)</td>
<td>Creates a new Text object with the contents of a String and the specified width and height.</td>
</tr>
<tr>
<td>Text(String contents, int offset, int length, int width, int height)</td>
<td>Creates a new Text object with the specified substring, width and height.</td>
</tr>
</tbody>
</table>
### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void delete(int index, int length)</code></td>
<td>Deletes the range of characters in the text.</td>
</tr>
<tr>
<td><code>int getAlignment()</code></td>
<td>Gets the alignment for this Text object.</td>
</tr>
<tr>
<td><code>int getBackgroundColor()</code></td>
<td>Gets the color used to fill the background of the Text.</td>
</tr>
<tr>
<td><code>int getCaret()</code></td>
<td>Gets the index of the caret.</td>
</tr>
<tr>
<td><code>void getCharExtent(int index, int[] extent)</code></td>
<td>Gets the extent within this Text where the specified character is located.</td>
</tr>
<tr>
<td><code>int getCharIndex(int x, int y)</code></td>
<td>Gets the index of the character nearest the location within the bounding box.</td>
</tr>
<tr>
<td><code>javax.microedition.lcdui.Font getFont()</code></td>
<td>Gets the default Font for this Text object.</td>
</tr>
<tr>
<td><code>javax.microedition.lcdui.Font getFont(int index)</code></td>
<td>Gets the Font in use for the character at index.</td>
</tr>
<tr>
<td><code>int getForegroundColor()</code></td>
<td>Gets the color used to draw any character that does not have a per character color.</td>
</tr>
<tr>
<td><code>int getForegroundColor(int index)</code></td>
<td>Gets the color used to draw the character at index.</td>
</tr>
<tr>
<td><code>int getHeight()</code></td>
<td>Gets the height of this Text.</td>
</tr>
<tr>
<td><code>int getHighlightIndex()</code></td>
<td>Gets the highlight index.</td>
</tr>
<tr>
<td><code>int getHighlightLength()</code></td>
<td>Gets the highlight length.</td>
</tr>
<tr>
<td><code>int getIndent()</code></td>
<td>Gets the horizontal first line indent used at the beginning of each paragraph.</td>
</tr>
<tr>
<td><code>int getInitialDirection()</code></td>
<td>Gets the initial display direction.</td>
</tr>
<tr>
<td><code>int getRequiredHeight()</code></td>
<td>Gets the height, in pixels, needed to fully render the text given its current width, font, and contents.</td>
</tr>
<tr>
<td><code>int getRequiredLineCount()</code></td>
<td>Gets the number of lines needed to fully render this Text.</td>
</tr>
<tr>
<td><code>int getScrollOffset()</code></td>
<td>Gets the scroll offset for the text.</td>
</tr>
</tbody>
</table>
javax.microedition.lcdui.Text

int getSpaceAbove()

Gets the space above each line of text.

int getSpaceBelow()

Gets the space below each line of text.

java.lang.String getText(int index, int length)

Gets a substring from this Text.

int getTextLength()

Gets the length of the Text in characters.

int getWidth()

Gets the width of this Text.

void insert(int index, String contents)

Inserts the contents into this Text object.

int lastRenderedIndex()

Gets the index of the last character that fits within the bounding box.

void moveCaret(int nlines)

Move caret forward or backward by one line in the display. The getCharExtent method is use to find the bounding box of the character identified getCaret.

void setAlignment(int alignment)

Sets the alignment for this Text object.

void setBackgroundColor(int color)

Sets the color used to fill the background of the Text.

void setCaret(int index)

Sets the index of the caret.

void setFont(Font font)

Sets the default Font for this Text object.

void setFont(Font font, int index, int length)

Sets the per character Font for a subrange of this Text object.

void setForegroundColor(int color)

Sets the color used to draw characters that do not have a per character foreground color.

void setForegroundColor(int color, int index, int length)

Sets the color used to draw length characters starting at index.

void setHeight(int height)

Sets the height of this Text.

void setHighlight(int index, int length)

Sets the highlight on a range in the text.

void setIndent(int indent)

Sets the horizontal first line indent used at the beginning of each paragraph.

void setInitialDirection(int direction)

Sets the initial display direction for Text.
### Methods

- **setScrollOffset(int offset)**
  Sets the scroll offset of the text.

- **setSpaceAbove(int spaceAbove)**
  Sets the line spacing above each line.

- **setSpaceBelow(int spaceBelow)**
  Sets the line space below each line.

- **setWidth(int width)**
  Sets the width of this Text.

- **textFits()**
  Checks if the contents can be fully rendered within the current bounds of this Text.

### Fields

- **ALIGN_CENTER**
  ```java
  public static final int ALIGN_CENTER
  Center each line of text horizontally within this Text.
  Constant value: 1
  Since: MIDP 3.0
  ```

- **ALIGN_DEFAULT**
  ```java
  public static final int ALIGN_DEFAULT
  Align each line using the default.
  Constant value: 4
  Since: MIDP 3.0
  ```

- **ALIGN_JUSTIFY**
  ```java
  public static final int ALIGN_JUSTIFY
  Justify the each line both left and right.
  Constant value: 3
  Since: MIDP 3.0
  ```

- **ALIGN_LEFT**
  ```java
  public static final int ALIGN_LEFT
  Align each line of text to the left edge of this Text.
  Constant value: 0
  Since: MIDP 3.0
  ```
ALIGN_RIGHT

public static final int ALIGN_RIGHT

Align each line of text to the right edge of this Text.
Constant value: 2
Since: MIDP 3.0

DIRECTION_LTR

public static final int DIRECTION_LTR

Initial Left-to-Right display order.
Constant value: 10
Since: MIDP 3.0

DIRECTION_NEUTRAL

public static final int DIRECTION_NEUTRAL

Initial neutral display order.
Constant value: 12
Since: MIDP 3.0

DIRECTION_RTL

public static final int DIRECTION_RTL

Initial Right-to-Left display order.
Constant value: 11
Since: MIDP 3.0

Constructors

Text

public Text ()

Creates a new Text object. The contents are empty and the width and height are zero.
Since: MIDP 3.0

Text

public Text (String contents,
        int width,
        int height)

Creates a new Text object with the contents of a String and the specified width and height.

Parameters:
        contents - the String that is to be used as the contents of this Text
        width - the width of the bounding box, in pixels.
        height - the height of the bounding box, in pixels.

Throws:
        java.lang.IllegalArgumentException - if the width or height are less than 0.
        NullPointerException - if contents is null.
Since: MIDP 3.0
public Text(String contents,  
   int offset,  
   int length,  
   int width,  
   int height)

Creates a new Text object with the specified substring, width and height.

Parameters:
   contents - String containing the characters to be used as the contents of this Text object.
   offset - offset within the contents of the first new character.
   length - the number of characters to insert from the contents.
   width - the width of the bounding box, in pixels.
   height - the height of the bounding box, in pixels.

Throws:
   java.lang.IndexOutOfBoundsException - if offset and length do not specify a valid range within contents.
   java.lang.IllegalArgumentException - if the length, width or height are less than 0.
   NullPointerException - if contents is null.

Since: MIDP 3.0

Methods

delete

public void delete(int index,  
                   int length)

Deletes the range of characters in the text.

Parameters:
   index - the index of the first character to delete.
   length - the length of string to delete.

Throws:
   java.lang.IndexOutOfBoundsException - if index and length do not specify a valid range within this Text.

Since: MIDP 3.0

getAlignment

public int getAlignment()

Gets the alignment for this Text object.

Returns:
   the alignment mode for this Text object; one of ALIGN_LEFT, ALIGN_CENTER, ALIGN_RIGHT, 
   ALIGN_JUSTIFY, or ALIGN_DEFAULT.

See Also: setAlignment(int)
Since: MIDP 3.0

getBackgroundColor

public int getBackgroundColor()

Gets the color used to fill the background of the Text.

Returns:
   the ARGB color of the background color.
### getCharIndex

```java
public int getCharIndex(int x, int y)
```

Gets the index of the character nearest the location within the bounding box.

**Parameters:**
- `x` - the x offset.
- `y` - the y offset.

**Returns:**
- the index of the character nearest the location; -1 if there is no character near the location.

**Since:** MIDP 3.0

### getFont

```java
public javax.microedition.lcdui.Font getFont()
```

Gets the default Font for this Text object. Per character fonts of regions can be retrieved with `getFont(int)`. The Font is used to determine the metrics of each character for the purposes of laying out the text within the bounding box. By default, the Font is set to the system default font, as returned by `Font.getDefaultFont()`.

**Returns:**
- the default Font being used for this Text object.

**See Also:** `setFont(Font)`

**Since:** MIDP 3.0
getFont

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

Gets the Font in use for the character at index. The Font is used to determine the metrics of each character for the purposes of laying out the text within the bounding box. The Font is also used to render the text.

Parameters:
index - the index of the character for which to query the font.

Returns:
the Font being used for this Text object; null is returned if the character does not have an per character font.

Throws:
java.lang.IndexOutOfBoundsException - if index is out of range for this Text.

See Also: setFont(Font, int, int)

getForegroundColor

public int getForegroundColor()

Gets the color used to draw any character that does not have a per character color. Until set with setForegroundColor(color), the value is set to the foreground color provided by Display.getColor(Display.COLOR_FOREGROUND).

Returns:
the ARGB color used to draw characters without a per character color.

See Also: setForegroundColor(int), setForegroundColor(int, int, int), getForegroundColor(int)

getForegroundColor

public int getForegroundColor(int index)

Gets the color used to draw the character at index.

Parameters:
index - the index to query for its ARGB color.

Returns:
the ARGB color of the character at index.

Throws:
java.lang.IndexOutOfBoundsException - if index is out of range for this Text.

See Also: setForegroundColor(int, int, int)

Since: MIDP 3.0

getHeight

public int getHeight()

Gets the height of this Text.

Returns:
height in pixels.
javax.microedition.lcdui.Text

See Also: getHeight(int)
Since: MIDP 3.0

getHighlightIndex

public int getHighlightIndex()

Gets the highlight index.

Returns: the first character highlighted; -1 if none.

See Also: getHighlightLength()
Since: MIDP 3.0

getHighlightLength

public int getHighlightLength()

Gets the highlight length.

Returns: the length of the highlight; zero if none.

See Also: getHeight(int)
Since: MIDP 3.0

getIndent

public int getIndent()

Gets the horizontal first line indent used at the beginning of each paragraph.

Returns: the indent in pixels.

See Also: getIndent(int)
Since: MIDP 3.0

getInitialDirection

public int getInitialDirection()

Gets the initial display direction.

Returns: the initial display direction; one of DIRECTION_LTR, DIRECTION_RTL, or DIRECTION_NEUTRAL.

See Also: setInitialDirection(int)
Since: MIDP 3.0

getRequiredHeight

public int getRequiredHeight()

Gets the height, in pixels, needed to fully render the text given its current width, font, and contents.

Returns: the height needed to render the contents of this Text.

Since: MIDP 3.0
getRequiredLineCount

public int getRequiredLineCount()

    Gets the number of lines needed to fully render this Text.

Returns:
    the number of lines

Since: MIDP 3.0

getScrollOffset

public int getScrollOffset()

    Gets the scroll offset for the text.

Returns:
    the scroll offset, in pixels.

See Also: setScrollOffset(int)

Since: MIDP 3.0

g.spaceAbove

public int getSpaceAbove()

    Gets the space above each line of text.

Returns:
    the number of pixels of additional space above each line; may be negative to tighten up the spacing.

See Also: setSpaceAbove(int), getSpaceBelow()

Since: MIDP 3.0

getSpaceBelow

public int getSpaceBelow()

    Gets the space below each line of text.

Returns:
    the number of pixels of additional space below each line; may be negative to tighten up the spacing.

See Also: setSpaceBelow(int), getSpaceAbove()

Since: MIDP 3.0

getText

public java.lang.String getText(int index, int length)

    Gets a substring from this Text.

Parameters:
    index - the index of the first character to return.
    length - the length of the substring to return.

Returns:
    the String containing the characters from index including length characters.

Throws:
    java.lang.IndexOutOfBoundsException - if index and length do not specify a valid range within this Text.
Since: MIDP 3.0

getTextLength

public int getTextLength()

Gets the length of the Text in characters.

Returns: the number of characters

Since: MIDP 3.0

getWidth

public int getWidth()

Gets the width of this Text.

Returns: width in pixels.

See Also: setWidth(int)

Since: MIDP 3.0

insert

public void insert(int index, String contents)

Inserts the contents into this Text object. The string is inserted just prior to the character indicated by the index parameter, where zero specifies the first character of the Text. If position is less than or equal to zero, the insertion occurs at the beginning of the Text, thus effecting a prepend operation. If position is greater than or equal to the current size of the Text, the insertion occurs immediately after the end of the Text, thus effecting an append operation. For example, text.insert(s, text.size()) always appends the string s to the current Text.

Parameters:
index - the index where the new contents are to be inserted.
contents - The String to be used as the contents.

Since: MIDP 3.0

lastRenderedIndex

public int lastRenderedIndex()

Gets the index of the last character that fits within the bounding box.

Returns: the index of the last character that fits within the bounding box.

Since: MIDP 3.0

moveCaret

public void moveCaret(int nlines)

Move caret forward or backward by one line in the display. The getCharExtent method is use to find the bounding box of the character identified getCursor. The vertical position is offset by the height of the line. The horizontal position maintained. The new caret position is set using the getCharIndex method with this new vertical offset.

Parameters:
nlines - the number of lines to move; +1 to move forward; -1 to move backward; zero to stay in place.
**setAlignment**

```java
public void setAlignment(int alignment)
```

Sets the alignment for this Text object. MUST be one of ALIGN_LEFT, ALIGN_CENTER, ALIGN_RIGHT, ALIGN_JUSTIFY, or ALIGN_DEFAULT.

**Parameters:**
alignment - the alignment mode to be used for laying out and rendering this Text.

**Throws:**
java.lang.IllegalArgumentException - if the value is not one of the listed values.

**See Also:** getAlignment()  Since: MIDP 3.0

**setBackgroundColor**

```java
public void setBackgroundColor(int color)
```

Sets the color used to fill the background of the Text. If the color is fully transparent and then it has no effect.

**Parameters:**
color - the ARGB color to use for the background.

**See Also:** getBackgroundColor()  Since: MIDP 3.0

**setCaret**

```java
public void setCaret(int index)
```

Sets the index of the caret. The caret can be used to indicate a position in the text. If characters are inserted, the caret index is increased by the number of characters inserted at indexes before or equal to the caret index. If characters are deleted, the caret index is decreased by the number of characters deleted from indexes before or equal to the caret index.

**Parameters:**
index - an integer between 0 and getTextLength(), inclusive, indicating where in the text to place the caret (0 being immediately before the first character and getTextLength() being immediately after the last); -1 if the caret is not to be displayed

**Throws:**
java.lang.IndexOutOfBoundsException - if index is not in the range -1 to getTextLength(), inclusive

**See Also:** setCaret(), insert(int, String), delete(int, int)  Since: MIDP 3.0

**setFont**

```java
public void setFont(Font font)
```

Sets the default Font for this Text object. Per character fonts of regions can be set with setFont(Font, int, int). The Font and per character fonts are used to determine the metrics of each character for the purposes of laying out the text and also used to render the text. By default, the default Font is set to the system default font, as returned by Font.getDefaultFont().

**Parameters:**
font - the Font to be used for this Text object.

**Throws:**
setFont

```java
public void setFont(Font font, int index, int length)
```

Sets the per character Font for a subrange of this Text object. The Font is used to determine the metrics of length characters starting at index for the purposes of laying out the text and it is also used to render the text. Any font previously set on each of the characters in the range is removed and replaced by the new font. Calling this method with a null value will reset the per character font so the default Text will be used.

**Parameters:**
- font - the Font to be used for this subrange of characters; null to revert to the default font.
- index - the first character to use the font.
- length - the number of characters in the subrange.

**Throws:**
- `java.lang.IndexOutOfBoundsException` - if index and length do not specify a valid range within this Text.

**See Also:** `getFont()`, `getFont(int)`

**Since:** MIDP 3.0

setForegroundColor

```java
public void setForegroundColor(int color)
```

Sets the color used to draw characters that do not have a per character foreground color.

**Parameters:**
- color - the ARGB color to use.

**See Also:** `getForegroundColor()`, `setForegroundColor(int, int, int)`, `getForegroundColor(int)`

**Since:** MIDP 3.0

setForegroundColor

```java
public void setForegroundColor(int color, int index, int length)
```

Sets the color used to draw length characters starting at index. The per character color is removed if the color value is zero.

**Parameters:**
- color - the ARGB color to use; zero to remove the color from each character in the range.
- index - the first character to use the color.
- length - the number of characters to use the color; must not be less than zero.

**Throws:**
- `java.lang.IndexOutOfBoundsException` - if index and length do not specify a valid range within this Text.

**See Also:** `getForegroundColor(int)`

**Since:** MIDP 3.0

setHeight

```java
public void setHeight(int height)
```
Sets the height of this Text.

**Parameters:**
- height - height in pixels.

**Throws:**
- IllegalArgumentException - if height is less than zero.

**See Also:** getHeight()

**Since:** MIDP 3.0

---

**setHighlight**

```java
public void setHighlight(int index, int length)
```

Sets the highlight on a range in the text.

**Parameters:**
- index - the index of the first character to be highlighted.
- length - the length in characters to be highlighted.

**Throws:**
- java.lang.IndexOutOfBoundsException - if index and length do not specify a valid range within this Text.

**See Also:** getHighlightIndex(), getHighlightLength()

**Since:** MIDP 3.0

---

**setIndent**

```java
public void setIndent(int indent)
```

Sets the horizontal first line indent used at the beginning of each paragraph.

**Parameters:**
- indent - the indent to be used, in pixels.

**Throws:**
- java.lang.IllegalArgumentException - if indent is less than 0.

**See Also:** getIndent()

**Since:** MIDP 3.0

---

**setInitialDirection**

```java
public void setInitialDirection(int direction)
```

Sets the initial display direction for Text.

**Parameters:**
- direction - the display direction; MUST be one of DIRECTION_LTR, DIRECTION_RTL, or DIRECTION_NEUTRAL.

**Throws:**
- IllegalArgumentException - if direction is invalid value.

**See Also:** getInitialDirection()

**Since:** MIDP 3.0

---

**setScrollOffset**

```java
public void setScrollOffset(int offset)
```
Sets the scroll offset of the text. The text displayed in the bounding box is moved up by the amount of the offset. All computations between character indexes and positions within the bounding box are affected by the scroll offset.

**Parameters:**
- `offset` - the scroll offset, in pixels.

**Throws:**
- `IllegalArgumentException` - if `offset` is less than zero.

**See Also:** `getScrollOffset()`, `getWidth()`

**Since:** MIDP 3.0

---

**setSpaceAbove**

```java
public void setSpaceAbove(int spaceAbove)
```

Sets the line spacing above each line.

**Parameters:**
- `spaceAbove` - the number of pixels to add above each line; may be negative to tighten up the spacing.

**See Also:** `getSpaceAbove()`, `getSpaceBelow()`

**Since:** MIDP 3.0

---

**setSpaceBelow**

```java
public void setSpaceBelow(int spaceBelow)
```

Sets the line space below each line.

**Parameters:**
- `spaceBelow` - the number of pixels to add below each line; may be negative to tighten up the spacing.

**See Also:** `getSpaceAbove()`, `getSpaceBelow()`

**Since:** MIDP 3.0

---

**setWidth**

```java
public void setWidth(int width)
```

Sets the width of this Text.

**Parameters:**
- `width` - width in pixels.

**Throws:**
- `IllegalArgumentException` - if `width` is less than zero.

**See Also:** `getWidth()`

**Since:** MIDP 3.0

---

**textFits**

```java
public boolean textFits()
```

Checks if the contents can be fully rendered within the current bounds of this Text.

**Returns:**
- true if the contents fit inside the bounds of this Text; otherwise false.

**Since:** MIDP 3.0
javax.microedition.lcdui

TextBox

Declaration

class TextBox extends Screen

Object
   +-javax.microedition.lcdui.Displayable
   |     +-javax.microedition.lcdui.Screen
   |          +-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>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>TextBox(String title, String text, int maxSize, int constraints)</td>
</tr>
<tr>
<td></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</td>
<td>delete(int offset, int length)</td>
</tr>
<tr>
<td></td>
<td>Deletes characters from the TextBox.</td>
</tr>
</tbody>
</table>
javax.microedition.lcdui.TextBox

- **int getCaretPosition()**
  Gets the current input position.

- **int getChars(char[] data)**
  Copies the contents of the TextBox into a character array starting at index zero.

- **int getConstraints()**
  Gets the current input constraints of the TextBox.

- **int getHeight()**
  Returns the height in pixels of the displayable area in the TextBox that is used to render the text without scrolling.

- **int getMaxSize()**
  Returns the maximum size (number of characters) that can be stored in this TextBox.

- **java.lang.String getString()**
  Gets the contents of the TextBox as a string value.

- **int getWidth()**
  Returns the width in pixels of the displayable area in the TextBox that is used to render the text without scrolling.

- **void insert(char[] data, int offset, int length, int position)**
  Inserts a subrange of an array of characters into the contents of the TextBox.

- **void insert(String src, int position)**
  Inserts a string into the contents of the TextBox.

- **void setCaret(int index)**
  Sets the index of the caret.

- **void setChars(char[] data, int offset, int length)**
  Sets the contents of the TextBox from a character array, replacing the previous contents.

- **void setConstraints(int constraints)**
  Sets the input constraints of the TextBox.

- **void setHighlight(int index, int length)**
  Sets the highlight on a range in the text.

- **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.

- **int setMaxSize(int maxSize)**
  Sets the maximum size (number of characters) that can be contained in this TextBox.

- **void setString(String text)**
  Sets the contents of the TextBox as a string value, replacing the previous contents.

- **int size()**
  Gets the number of characters that are currently stored in this TextBox.

**Methods inherited from class [javax.microedition.lcdui.Displayable]**
javax.microedition.lcdui.TextBox

Methods inherited from class Object

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

Methods

delete

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

Constructors

TextBox

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 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
**getCaretPosition**

```java
public int getCaretPosition()
```

Gets the current input position.

**Returns:**
the current caret position, 0 if at the beginning

**getChars**

```java
public int getChars(char[] data)
```

Copies the contents of the `TextBox` into a character array starting at index zero. Array elements beyond the characters copied are left unchanged.

**Parameters:**
- `data` - the character array to receive the value

**Returns:**
the number of characters copied

**Throws:**
- `ArrayIndexOutOfBoundsException` - if the array is too short for the contents
- `NullPointerException` - if `data` is null

**See Also:** `setChars(char[], int, int)`

**getConstraints**

```java
public int getConstraints()
```

Gets the current input constraints of the `TextBox`.

**Returns:**
the current constraints value (see input constraints)

**See Also:** `setConstraints(int)`

**getHeight**

```java
public int getHeight()
```

Returns the height in pixels of the displayable area in the `TextBox` that is used to render the text without scrolling. This displayable area only includes the area in which the text can be displayed. This area MUST NOT include any `Title`, `flicker`, or any other artifact such as the scrollbar which does not render the text.

**Returns:**
the height in pixels available to display text in the `TextBox`

**Since:** MIDP 3.0

**getMaxSize**

```java
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:** `setMaxSize(int)`
getString

public java.lang.String getString()

Gets the contents of the TextBox as a string value.

Returns:
the current contents

See Also: setString(String)

getWidth

public int getWidth()

Returns the width in pixels of the displayable area in the TextBox that is used to render the text without scrolling. This displayable area only includes the area in which the text can be displayed. This area MUST NOT include any Title,Ticker, or any other artifact such as the scrollbar which does not render the text.

Parameters:

Returns:
the width in pixels available to display text in the TextBox

Since: MIDP 3.0

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

Throws:
- ArrayIndexOutOfBoundsException - if offset and length do not specify a valid range within the data array
- IllegalArgumentException - if the resulting contents would be illegal for the current input constraints
- IllegalArgumentException - if the insertion would exceed the current maximum capacity
- NullPointerException - if data is null

insert

public void insert(String src,
int position)
Since: MIDP 3.0

setChars

Sets the contents of the TextBox from a character array, replacing the previous contents. Characters are copied from the region of the data array starting at array index offset and running for length characters. If the data array is null, the TextBox is set to be empty and the other parameters are ignored.

The offset and length parameters must specify a valid range of characters within the character array data. The offset parameter must be within the range [0..(data.length)], inclusive. The length parameter must be a non-negative integer such that (offset + length) <= data.length.

Parameters:

data - the source of the character data
offset - the beginning of the region of characters to copy
length - the number of characters to copy

Throws:

ArrayIndexOutOfBoundsException - if offset and length do not specify a valid range within the data array
IllegalArgumentException - if data is illegal for the current input constraints
IllegalArgumentException - if the text would exceed the current maximum capacity

See Also: `getChars(char[])`

---

**setConstraints**

```java
public void setConstraints(int constraints)
```

Sets the input constraints of the TextBox. If the current contents of the TextBox do not match the new constraints, the contents are set to empty.

**Parameters:**
- `constraints` - see input constraints

**Throws:**
- IllegalArgumentException - if the value of the constraints parameter is invalid

See Also: `getConstraints()`

---

**setHighlight**

```java
public void setHighlight(int index, int length)
```

Sets the highlight on a range in the text.

**Parameters:**
- `index` - the index of the first character to be highlighted.
- `length` - the length in characters to be highlighted.

**Since:** MIDP 3.0

---

**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: 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: getString()

**size**

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

Gets the number of characters that are currently stored in this TextBox.

**Returns:**
- the number of characters
javax.microedition.lcdui

TextEditor

Declaration

public class TextEditor extends CanvasItem

Object


javax.microedition.lcdui.CanvasItem

Description

A TextEditor is an editable text component that is drawn on a parent object; in LCDUI Canvas or CustomItem (including IdleItem), TextEditor inherits from CanvasItem. The TextEditor may be set to Canvas through setParent(canvas) method and removed through method call setParent(null). The TextEditor must be presented so that the Canvas painting happens independently regardless of the TextEditor editing or focus. If multiple TextEditors are added on the same instance of Canvas or CustomItem and their positions overlap, the initial z-order is the same as the adding order of the TextEditors; i.e. the first added TextEditor is the lowest in the stack. For usability applications should avoid adding overlapping TextEditors.

The implementation should initially present the TextEditor with minimal decoration; only the caret should be shown at the requested text insertion position. It is then the responsibility of the application to draw any additional decoration like focus highlight, border or scroll bar. The animation of the caret, e.g. blinking, is handled by the Java platform implementation.

TextEditor supports input constraints identically to TextField. See input constraints section in the TextField for the definition of these constants. In addition TextEditor has the same concept of actual contents and displayed contents as TextField; these are described in the same input constraints section.

TextEditor supports input modes identically to TextField. See input modes section in the TextField for the definition of these constants.

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.

The implementation should provide necessary interaction for example for pen input (handwriting recognition) on touch screen devices and selection of additional characters. It should be noted that the implementation provided visuals, e.g. character input panels, may require additional windows on top of the text editor and may obscure both the editor and its parent partly or completely.

Application can add a TextEditorChangeListener to the TextEditor for example for keeping track of user navigation (caret movement) and other content changes such as text selection. The events are sent on all occasions that cause the caret position to change (including but not limited to text typing by user, programmatic text insertion, navigation within the TextEditor content, and text deletion). The events must be sent to the application after they have effect on the editor - for example an event indicating caret movement must be available for the application after the implementation actually moves the caret in the editor.

When the TextEditor has focus all the key events that are not consumed by TextEditor with the current constraints, or mapped to Commands by the implementation, are sent to Canvas.

The implementation must scroll the TextEditor content automatically on user interaction and when the application calls setCaretPosition or setSelection methods. For example if the user clicks down on the last visible row the TextEditor content is scrolled accordingly by one row. However the Java platform implementation should not draw any scroll bars, this is left to the application.
Since: 3.0  
See Also: CanvasItem, Canvas, CustomItem

### Constructor Summary

<table>
<thead>
<tr>
<th>Method ID</th>
<th>Description</th>
</tr>
</thead>
</table>
| public | TextEditor(String text, int maxSize, int constraints, int width, int height)  
  Creates a new TextEditor object with the given initial contents, maximum size in characters, constraints and editor size in pixels. |

### Method Summary

<table>
<thead>
<tr>
<th>Method ID</th>
<th>Description</th>
</tr>
</thead>
</table>
| void | delete(int offset, int length)  
  Deletes characters from the TextEditor. |
| int | getBackgroundColor()  
  Gets the current background color and alpha of this TextEditor. |
| int | getCaretPosition()  
  Gets the current position of the caret in the editor. |
| int | getConstraints()  
  Gets the current input constraints of this TextEditor. |
| int | getContentHeight()  
  Gets the whole content height in this TextEditor in pixels. |
| boolean | getFocus()  
  Gets the current focus state of the TextEditor. |
| javax.microedition.lcdui.Font | getFont()  
  Gets the font being used in rendering the text content in this TextEditor. |
| int | getForegroundColor()  
  Gets the current foreground color and alpha of this TextEditor. |
| int | getHighlightColor()  
  Gets the current highlight color for this TextEditor. |
| int | getLineMarginHeight()  
  Gets the possible line margin height that this editor has in addition to the normal font height as returned by Font.getHeight(). |
| int | getMaxSize()  
  Returns the maximum size (number of characters) that can be stored in this TextEditor. |
| java.lang.String | getSelection()  
  Gets the currently selected content in the TextEditor. |
| java.lang.String | getString()  
  Gets a string representing the current content in the TextEditor. |
| boolean | getVisible()  
  Gets the visibility value of the TextEditor. |
| int | getVisibleContentPosition()  
  Gets the topmost pixel y-position of the topmost visible line in the editor. |
<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>void</strong></td>
<td><code>insert(String text, int position)</code></td>
<td>Inserts a string into the content of the TextEditor.</td>
</tr>
<tr>
<td><strong>void</strong></td>
<td><code>setBackgroundColor(int alpha, int red, int green, int blue)</code></td>
<td>Sets the background color and alpha of this TextEditor to the specified values.</td>
</tr>
<tr>
<td><strong>void</strong></td>
<td><code>setCaretPosition(int index)</code></td>
<td>Sets the index of the caret.</td>
</tr>
<tr>
<td><strong>void</strong></td>
<td><code>setConstraints(int constraints)</code></td>
<td>Sets the input constraints of this TextEditor.</td>
</tr>
<tr>
<td><strong>void</strong></td>
<td><code>setFocus(boolean focused)</code></td>
<td>Sets this TextEditor focused or removes focus.</td>
</tr>
<tr>
<td><strong>void</strong></td>
<td><code>setFont(Font font)</code></td>
<td>Sets the application preferred font for rendering the text content in this TextEditor.</td>
</tr>
<tr>
<td><strong>void</strong></td>
<td><code>setForegroundColor(int alpha, int red, int green, int blue)</code></td>
<td>Sets the foreground color and alpha of this TextEditor to the specified values.</td>
</tr>
<tr>
<td><strong>void</strong></td>
<td><code>setHighlightColor(int alpha, int red, int green, int blue)</code></td>
<td>Sets the highlight color for this TextEditor to the specified values.</td>
</tr>
<tr>
<td><strong>void</strong></td>
<td><code>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 TextEditor; i.e.</td>
</tr>
<tr>
<td><strong>void</strong></td>
<td><code>setMaxSize(int maxSize)</code></td>
<td>Sets the maximum size (number of characters) that can be contained in this TextEditor.</td>
</tr>
<tr>
<td><strong>void</strong></td>
<td><code>setSelection(int index, int length)</code></td>
<td>Sets a selection on a range of text in the TextEditor content.</td>
</tr>
<tr>
<td><strong>void</strong></td>
<td><code>setString(String text)</code></td>
<td>Sets the content of the TextEditor as a string.</td>
</tr>
<tr>
<td><strong>void</strong></td>
<td><code>setTextEditorListener(TextEditorChangeListener listener)</code></td>
<td>Sets a listener for content changes in this TextEditor, replacing any previous TextEditorChangeListener.</td>
</tr>
<tr>
<td><strong>void</strong></td>
<td><code>setVisible(boolean visible)</code></td>
<td>Sets the visibility value of the TextEditor.</td>
</tr>
<tr>
<td><strong>int</strong></td>
<td><code>size()</code></td>
<td>Gets the number of characters that are currently stored in this TextEditor.</td>
</tr>
</tbody>
</table>

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


**Methods inherited from class** `Object`

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

TextEditor

public TextEditor(String text,
                   int maxSize,
                   int constraints,
                   int width,
                   int height)

Creates a new TextEditor object with the given initial contents, maximum size in characters, constraints and editor size in pixels.

If the text parameter is null, the TextEditor is created empty.

The maxSize parameter must be greater than zero. An IllegalArgumentException is thrown if the length of the initial contents string exceeds the maximum size of the editor. 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 as returned by getMaxSize, the contents are truncated from the end in order to fit, and no exception is thrown.

Parameters:
  text - the initial contents, or null if the TextEditor is to be created empty
  maxSize - the maximum capacity in characters
  constraints - see the input constraints in TextField
  width - the width of the bounding box, in pixels
  height - the height of the bounding box, in pixels

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 maximum size of the editor as returned by getMaxSize
  IllegalArgumentException - if the width or height is less than one pixel

Methods

delete

public void delete(int offset,
                    int length)

Deletes characters from the TextEditor.

The offset and length parameters must specify a valid range of characters within the contents of the TextEditor. 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 content of the TextEditor
**getBackgroundColor**

```java
public int getBackgroundColor()
```

Gets the current background color and alpha of this TextEditor.

**Returns:**
the current color and alpha value encoded using the 32-bit ARGB format

**getCaretPosition**

```java
public int getCaretPosition()
```

Gets the current position of the caret in the editor.

**Returns:**
the current caret position, 0 if at the beginning

**getConstraints**

```java
public int getConstraints()
```

Gets the current input constraints of this TextEditor.

**Returns:**
the current constraints value (see input constraints)

**getContentHeight**

```java
public int getContentHeight()
```

Gets the whole content height in this TextEditor in pixels. The returned value must include the height of the whole content in the editor, not just the height of the visible content.

**Returns:**
the height of the whole content in the editor in pixels

**getFocus**

```java
public boolean getFocus()
```

Gets the current focus state of the TextEditor.

**Returns:**
true if the TextEditor has focus, false otherwise

**Throws:**
IllegalStateException - If the TextEditor is not added to a parent with setParent

**getFont**

```java
public javax.microedition.lcdui.Font getFont()
```

Gets the font being used in rendering the text content in this TextEditor.

**Returns:**
the font being used in this TextEditor
getForegroundColor

```java
public int getForegroundColor()
```

Gets the current foreground color and alpha of this TextEditor.

**Returns:**
the current color and alpha value encoded using the 32-bit ARGB format

getHighlightColor

```java
public int getHighlightColor()
```

Gets the current highlight color for this TextEditor.

**Returns:**
the current color and alpha value encoded using the 32-bit ARGB format

getLineMarginHeight

```java
public int getLineMarginHeight()
```

Gets the possible line margin height that this editor has in addition to the normal font height as returned by Font.getHeight(). The returned value is in pixels, and the returned value is 0 (zero) if the editor does not have any additional line margins to the text height returned from getHeight() of Font set to the editor.

**Returns:**
the line margin height in pixels, 0 if none

getMaxSize

```java
public int getMaxSize()
```

Returns the maximum size (number of characters) that can be stored in this TextEditor. The implementation may have assigned a smaller maximum size than requested by application with setMaxSize or in the constructor.

**Returns:**
the maximum size in characters

getSelection

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

Gets the currently selected content in the TextEditor. A selection may have been set with setSelection method or by user interaction.

**Returns:**
the currently selected content, null if no selection has been set

getString

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

Gets a string representing the current content in the TextEditor.

**Returns:**
the current content of the editor

getVisible

```java
public boolean getVisible()
```
javax.microedition.lcdui.TextEditor

getVisibleContentPosition

public int getVisibleContentPosition()

Gets the topmost pixel y-position of the topmost visible line in the editor. The returned y coordinate value must be relative to the whole content height, not just the visible part.

This method can be used by the application together with the getContentHeight, getLineMarginHeight and getCaretPosition methods in drawing custom visual cues like a scroll bar or other content sensitive pop-ups.

Returns:
the topmost pixel position of the visible content

insert

public void insert(String text, int position)

Inserts a string into the content of the TextEditor. The string is inserted just prior to the character indicated by the position parameter, where zero specifies the first character of the content in the TextEditor. If position is less than or equal to zero, the insertion occurs at the beginning of the content. If position is greater than or equal to the current size of the content, the insertion occurs immediately after the end of the content.

The current size of the contents is increased by the number of inserted characters. The resulting string must fit within the current maximum capacity as returned by getMaxSize.

Parameters:
text - the String to be inserted
position - the position at which insertion is to occur

Throws:
IllegalArgumentException - if the resulting content would be illegal for the current input constraints
IllegalArgumentException - if the insertion would exceed the current maximum capacity of the editor as returned by getMaxSize
NullPointerException - if text is null

setBackgroundColor

public void setBackgroundColor(int alpha, int red, int green, int blue)

Sets the background color and alpha of this TextEditor to the specified values. The set values apply also to any other related implementation provided input visuals; for example to the implementation provided additional character table or handwriting recognition panel.

Parameters:
alpha - the alpha component of the color being set within range 0-255
javax.microedition.lcdui.TextEditor

red - the red component of the color being set within range 0-255
green - the green component of the color being set within range 0-255
blue - the blue component of the color being set within range 0-255

Throws:
IllegalArgumentException - if any of the parameters is outside the range of 0-255

setCaretPosition

public void setCaretPosition(int index)

Sets the index of the caret. The caret can be used to indicate a position in the text and MUST be visible when the TextEditor is in focus. If characters are inserted, the caret index is increased by the number of characters inserted at indexes before or equal to the caret index. If characters are deleted, the caret index is decreased by the number of characters deleted from indexes before or equal to the caret index.

The Java platform implementation must scroll the content of the TextEditor automatically so that the caret is within the visible area. If the caret is set above the current position the content should be scrolled so that the caret is on the top most visible row. If the caret is set below the current position the content should be scrolled so that the caret is on the lowest visible row.

Parameters:
index - an integer between 0 and size(), inclusive, indicating where in the text to place the caret (0 being immediately before the first character and size() being immediately after the last)

Throws:
IndexOutOfBoundsException - if index is not in the range 0 to size(), inclusive

setConstraints

public void setConstraints(int constraints)

Sets the input constraints of this TextEditor. If the current content of this TextEditor do not match the new constraints, the content is set to empty.

Parameters:
constraints - see input constraints

Throws:
IllegalArgumentException - if the value of the constraints parameter is invalid

setFocus

public void setFocus(boolean focused)
Sets this `TextEditor` focused or removes focus.

Calling `setFocus(true)` enables text editing from keys as the delivery of needed key events will be targeted to the `TextEditor` instead of the parent object (e.g. `Canvas`). On touch screen devices a on-screen keyboard should be shown to enable text editing. Focus needs to be explicitly set to the `TextEditor` by application. Only in touch enabled devices user may tap to the editor which will result editor to gain also the focus automatically. If `Canvas` or `CustomItem` does not have any focused `TextEditor`, all key and pointer events are delivered to normal key delivery methods of the parent. Necessary key events are captured by `TextEditor` only when it has focus.

The `setFocus` method affects to the parent key event delivery in following way: If all `TextEditor`s are unfocused the key event delivery of the parent functions as if there were no `TextEditor`s in it. Only when a focus is set via this method to a `TextEditor` the key event delivery is modified: the focused editor starts to capture the necessary key events and these key events are not delivered to parent. The set of key events captured depends on the Java platform implementation and device hardware, but in most implementations nearly all device keys are captured by the editor for text insertion, input mode changes and caret move functionalities. Applications should not assume to get key events from keys mapped to game actions as most probably the same keys are used to navigate the text caret within the editor. The keys that are used for Command launching in `Canvas` are available for applications on focused `TextEditor` and these keys either launch commands or send low-level key events as normally.

Setting focus does not cause any visual focus indication by the implementation other than showing the caret. Any other change in the visual appearance of the editor in focused state is the responsibility of the application; this can be done for example by drawing a focus border around the `TextEditor` or by changing the background color or transparency of the editor.

If this `TextEditor` has already been focused earlier and the editor contents has not changed after previous unfocusing, then after calling `setFocus` again the caret position should be retained.

If there already is another focused `TextEditor` on the `Canvas`, the focus is first removed from that `TextEditor` before setting this `TextEditor` focused.

Calling `setFocus(false)` disables key based text editing and returns the delivery of key events and pointer events to the underlying parent (e.g. `Canvas`).

**Parameters:**
- `focused` - true to set focus, false to remove focus

**Throws:**
- `IllegalStateException` - If the `TextEditor` is not added to a parent with `setParent`
- `IllegalStateException` - If the `TextEditor` is not in visible state as returned by `getVisible`

---

**setFont**

```java
public void setFont(Font font)
```

Sets the application preferred font for rendering the text content in this `TextEditor`. Setting the font is a hint to the implementation, and the implementation may disregard the requested 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 text content.

The font change should be applied to the already visible editor and content as soon as possible. If setting the font changes for example the line wrapping of the already visible content, the implementation should maintain the caret visible.

**Parameters:**
- `font` - the application preferred font to be used in this `TextEditor`
javax.microedition.lcdui.TextEditor

**setForegroundColor**

```java
public void setForegroundColor(int alpha,
    int red,
    int green,
    int blue)
```

Sets the foreground color and alpha of this TextEditor to the specified values. The content, e.g. text, in the editor must be drawn with this color.

**Parameters:**
- alpha - the alpha component of the color being set within range 0-255
- red - the red component of the color being set within range 0-255
- green - the green component of the color being set within range 0-255
- blue - the blue component of the color being set within range 0-255

**Throws:**
- IllegalArgumentException - if any of the parameters is outside the range of 0-255

**setHighlightColor**

```java
public void setHighlightColor(int alpha,
    int red,
    int green,
    int blue)
```

Sets the highlight color for this TextEditor to the specified values. The highlight areas, e.g. a selection on a range of text, in the editor must be drawn with this color.

**Parameters:**
- alpha - the alpha component of the color being set within range 0-255
- red - the red component of the color being set within range 0-255
- green - the green component of the color being set within range 0-255
- blue - the blue component of the color being set within range 0-255

**Throws:**
- IllegalArgumentException - if any of the parameters is outside the range of 0-255

**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 TextEditor; i.e. when the TextEditor becomes focused. 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

**setMaxSize**

```java
public int setMaxSize(int maxSize)
```

Sets the maximum size (number of characters) that can be contained in this TextEditor. If the current content of the TextEditor is larger than the new maxSize, the content is truncated to fit. The implementation may assign a smaller maximum size than requested.

**Parameters:**
- maxSize - the new maximum size

**Returns:**
- assigned maximum capacity - may be smaller than requested.
javax.microedition.lcdui.TextEditor

setSelection

public void setSelection(int index, int length)

Sets a selection on a range of text in the TextEditor content. The implementation should highlight the selection visually. A selection may be set with this method or by user interaction. If there already is a selection set, it is replaced by this new selection.

The caret must be automatically set directly after the set selection. The Java platform implementation must scroll the content of the TextEditor automatically so that the caret is within the visible area and as much as possible of the selection is visible in the TextEditor.

Parameters:
  index - the index of the first character to be selected.
  length - the length of the selection in characters.

Throws:
  StringIndexOutOfBoundsException - if index and length do not specify a valid range within the content of the TextEditor

setString

public void setString(String text)

Sets the content of the TextEditor as a string. The set string replaces any previous content in the editor.

Parameters:
  text - the new content of the TextEditor as string, null empties the TextEditor

Throws:
  IllegalArgumentException - if text is illegal for the current input constraints
  IllegalArgumentException - if the given text would exceed the current maximum capacity of the editor as returned by getMaxSize

setTextEditorListener

public void setTextEditorListener(TextEditorChangeListener listener)

Sets a listener for content changes in this TextEditor, replacing any previous TextEditorChangeListener.

A null reference is allowed and has the effect of removing any existing TextEditorChangeListener from this TextEditor.

Callbacks to the TextEditorChangeListener may throw exceptions, but they must be ignored.

Parameters:
  listener - the new listener, or null

setVisible

public void setVisible(boolean visible)
Sets the visibility value of the TextEditor. Initially the TextEditor is not visible so it must be explicitly set to visible in order to appear on the user interface.

Setting visibility to true shows the editor with its content, but without the caret (calling setFocus(true) shows the caret). If the editor does not have any visible content and does not have any background color set then this method effectively does not cause any visual change in the display. If the editor is already visible, calling setVisible(true) does nothing.

Setting the visibility to false hides the editor and its content. If the TextEditor has focus then the focus is removed. If the editor is already hidden calling setVisible(false) does nothing.

**Parameters:**
visible - true to set visible, false to set invisible

---

**size**

public int **size()**

Gets the number of characters that are currently stored in this TextEditor.

**Returns:**
the number of characters
javax.microedition.lcdui

TextEditorChangeListener

Declaration

public interface TextEditorChangeListener

Description

A listener for receiving notification of content changes and other editor events that have been invoked on TextEditor objects.

When an editor event happens, the application is notified by calling the method on the TextEditorChangeListener that had been set on the TextEditor with a call to TextEditor.setTextEditorListener() method.

Since: 3.0

Field Summary

<table>
<thead>
<tr>
<th>public static final</th>
<th>ACTION_CARET_MOVE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Indicates that the caret in this TextEditor has moved.</td>
</tr>
<tr>
<td></td>
<td>Value: 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>ACTION_CONTENT_CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Indicates that the content of this TextEditor has changed.</td>
</tr>
<tr>
<td></td>
<td>Value: 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>ACTION_DIRECTION_CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Indicates that the direction of the writing-language has changed.</td>
</tr>
<tr>
<td></td>
<td>Value: 64</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>ACTION_INPUT_MODE_CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Indicates that the current input-mode has changed.</td>
</tr>
<tr>
<td></td>
<td>Value: 128</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>ACTION_LANGUAGE_CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Indicates that the current input-language has changed.</td>
</tr>
<tr>
<td></td>
<td>Value: 256</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>ACTION_TRAVERSE_NEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Indicates that the user tries to exit this TextEditor downwards.</td>
</tr>
<tr>
<td></td>
<td>Value: 16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>ACTION_TRAVERSE_PREVIOUS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Indicates that the user tries to exit this TextEditor upwards.</td>
</tr>
<tr>
<td></td>
<td>Value: 8</td>
</tr>
</tbody>
</table>

Method Summary

void inputAction(TextEditor textEditor, int actions)

This method is called by the Java platform implementation to notify the client about events in a TextEditor.
Fields

**ACTION_CARET_MOVE**

```java
public static final int ACTION_CARET_MOVE
```

Indicates that the caret in this TextEditor has moved.
Constant value: 4

**ACTION_CONTENT_CHANGE**

```java
public static final int ACTION_CONTENT_CHANGE
```

Indicates that the content of this TextEditor has changed.
Constant value: 1

**ACTION_DIRECTION_CHANGE**

```java
public static final int ACTION_DIRECTION_CHANGE
```

Indicates that the direction of the writing-language has changed.
Constant value: 64

**ACTION_INPUT_MODE_CHANGE**

```java
public static final int ACTION_INPUT_MODE_CHANGE
```

Indicates that the current input-mode has changed.
Constant value: 128

**ACTION_LANGUAGE_CHANGE**

```java
public static final int ACTION_LANGUAGE_CHANGE
```

Indicates that the current input-language has changed.
Constant value: 256

**ACTION_TRAVERSE_NEXT**

```java
public static final int ACTION_TRAVERSE_NEXT
```

Indicates that the user tries to exit this TextEditor downwards.
Constant value: 16

**ACTION_TRAVERSE_PREVIOUS**

```java
public static final int ACTION_TRAVERSE_PREVIOUS
```

Indicates that the user tries to exit this TextEditor upwards.
Constant value: 8

Methods
inputAction

public void inputAction(TextEditor textEditor, int actions)

This method is called by the Java platform implementation to notify the client about events in a TextEditor. A call to this method may represent more than one event. The events are masked into the actions parameter. An application can determine which events have occurred for example via code like:

if( (actions&TextEditorListener.ACTION_CONTENT_CHANGE)!=0), and

if( (actions&TextEditorListener.ACTION_CARET_CHANGE)!=0)

Parameters:
- textEditor - the TextEditor instance where the event occurred
- actions - the events that occurred
javax.microedition.lcdui

TextField

Declaration

public class TextField extends Item

Object

|--javax.microedition.lcdui.Item

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
- CURRENCY

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 the input of at least the following set of characters:

<table>
<thead>
<tr>
<th>Unicode value</th>
<th>Character name</th>
<th>Glyph</th>
</tr>
</thead>
<tbody>
<tr>
<td>U+0020</td>
<td>SPACE</td>
<td></td>
</tr>
<tr>
<td>U+0021</td>
<td>EXCLAMATION MARK</td>
<td>!</td>
</tr>
<tr>
<td>U+0022</td>
<td>QUOTATION MARK</td>
<td>&quot;</td>
</tr>
<tr>
<td>U+0023</td>
<td>NUMBER SIGN</td>
<td>#</td>
</tr>
<tr>
<td>U+0024</td>
<td>DOLLAR SIGN</td>
<td>$</td>
</tr>
<tr>
<td>U+0025</td>
<td>PERCENT SIGN</td>
<td>%</td>
</tr>
<tr>
<td>U+0026</td>
<td>AMPERSAND</td>
<td>&amp;</td>
</tr>
<tr>
<td>U+0027</td>
<td>APOSTROPHE</td>
<td>'</td>
</tr>
<tr>
<td>U+0028</td>
<td>LEFT PARENTHEsis</td>
<td>(</td>
</tr>
<tr>
<td>U+0029</td>
<td>RIGHT PARENTHEsis</td>
<td>)</td>
</tr>
<tr>
<td>U+002A</td>
<td>ASTERISK</td>
<td>*</td>
</tr>
<tr>
<td>U+002B</td>
<td>PLUS SIGN</td>
<td>+</td>
</tr>
<tr>
<td>U+002C</td>
<td>COMMA</td>
<td>,</td>
</tr>
<tr>
<td>U+002D</td>
<td>HYPHEN-MINUS</td>
<td>-</td>
</tr>
<tr>
<td>U+002E</td>
<td>FULL STOP (period)</td>
<td>.</td>
</tr>
<tr>
<td>U+002F</td>
<td>SOLIDUS (slash)</td>
<td>/</td>
</tr>
<tr>
<td>U+0030...U+0039</td>
<td>DIGIT ZERO through DIGIT NINE</td>
<td>0...9</td>
</tr>
<tr>
<td>U+003A</td>
<td>COLON</td>
<td>:</td>
</tr>
<tr>
<td>U+003B</td>
<td>SEMICOLON</td>
<td>;</td>
</tr>
<tr>
<td>U+003C</td>
<td>LESS-THAN SIGN</td>
<td>&lt;</td>
</tr>
<tr>
<td>U+003D</td>
<td>EQUALS SIGN</td>
<td>=</td>
</tr>
<tr>
<td>U+003E</td>
<td>GREATER-THAN SIGN</td>
<td>&gt;</td>
</tr>
<tr>
<td>U+003F</td>
<td>QUESTION MARK</td>
<td>?</td>
</tr>
<tr>
<td>U+0040</td>
<td>COMMERCIAL AT</td>
<td>@</td>
</tr>
<tr>
<td>U+0041...U+005A</td>
<td>LATIN CAPITAL LETTER A through LATIN CAPITAL LETTER Z</td>
<td>A...Z</td>
</tr>
<tr>
<td>U+005B</td>
<td>LEFT SQUARE BRACKET</td>
<td>[</td>
</tr>
</tbody>
</table>
Generally, any character that can be input using the input methods of the device SHOULD be supported.

Instances of TextField and TextBox with either of the constraints TextField.EMAILADDR and TextField.URL SHOULD allow the same characters to be input as are allowed for input constraint TextField.ANY. This is important particularly for entering URLs that are effectively IRIs (Internationalized Resource Identifiers, see [RFC3987]), which contain characters that would not traditionally be allowed in URLs. It is the responsibility of the application or the networking subsystem to encode the URLs into a format suitable for resource retrieval (using the mechanisms outlined in [RFC3987]).

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

<table>
<thead>
<tr>
<th>Unicode value</th>
<th>Character name</th>
<th>Glyph</th>
</tr>
</thead>
<tbody>
<tr>
<td>U+005C</td>
<td>REVERSE SOLIDUS (backslash)</td>
<td>\</td>
</tr>
<tr>
<td>U+005D</td>
<td>RIGHT SQUARE BRACKET</td>
<td>]</td>
</tr>
<tr>
<td>U+005E</td>
<td>CIRCUMFLEX ACCENT</td>
<td>^</td>
</tr>
<tr>
<td>U+005F</td>
<td>LOW LINE (underscore)</td>
<td>_</td>
</tr>
<tr>
<td>U+0060</td>
<td>GRAVE ACCENT</td>
<td>&lt;</td>
</tr>
<tr>
<td>U+0061...U+007A</td>
<td>LATIN SMALL LETTER A through LATIN SMALL LETTER Z</td>
<td>a...z</td>
</tr>
<tr>
<td>U+007B</td>
<td>LEFT CURLY BRACKET</td>
<td>{</td>
</tr>
<tr>
<td>U+007C</td>
<td>VERTICAL LINE</td>
<td></td>
</tr>
<tr>
<td>U+007D</td>
<td>RIGHT CURLY BRACKET</td>
<td>}</td>
</tr>
<tr>
<td>U+007E</td>
<td>TILDE</td>
<td>~</td>
</tr>
<tr>
<td>U+00A1</td>
<td>INVERTED EXCLAMATION MARK</td>
<td>i</td>
</tr>
<tr>
<td>U+00A3</td>
<td>POUND SIGN</td>
<td>£</td>
</tr>
<tr>
<td>U+00A4</td>
<td>CURRENCY SIGN</td>
<td>¢</td>
</tr>
<tr>
<td>U+00A5</td>
<td>YEN SIGN</td>
<td>¥</td>
</tr>
<tr>
<td>U+00A7</td>
<td>SECTION SIGN</td>
<td>§</td>
</tr>
<tr>
<td>U+00BF</td>
<td>INVERTED QUESTION MARK</td>
<td>¨</td>
</tr>
<tr>
<td>U+20AC</td>
<td>EURO SIGN</td>
<td>€</td>
</tr>
<tr>
<td>U+00A2</td>
<td>CENT SIGN</td>
<td>¢</td>
</tr>
<tr>
<td>U+00A8</td>
<td>DIAERESIS</td>
<td>¨</td>
</tr>
<tr>
<td>U+00A9</td>
<td>COPYRIGHT SIGN</td>
<td>©</td>
</tr>
<tr>
<td>U+00B0</td>
<td>DEGREE SIGN</td>
<td>°</td>
</tr>
<tr>
<td>U+00B4</td>
<td>ACUTE ACCENT</td>
<td>'</td>
</tr>
<tr>
<td>U+00B6</td>
<td>PILCROW SIGN</td>
<td>†</td>
</tr>
<tr>
<td>U+00B7</td>
<td>MIDDLE DOT</td>
<td>·</td>
</tr>
<tr>
<td>U+000A</td>
<td>LINE FEED</td>
<td></td>
</tr>
</tbody>
</table>
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 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:

<table>
<thead>
<tr>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCB_BASIC_LATIN</td>
</tr>
<tr>
<td>UCB_GREEK</td>
</tr>
<tr>
<td>UCB_CYRILLIC</td>
</tr>
<tr>
<td>UCB_ARABIC</td>
</tr>
<tr>
<td>UCB_DEVANAGARI</td>
</tr>
<tr>
<td>UCB_BENGALI</td>
</tr>
<tr>
<td>UCB_THAI</td>
</tr>
<tr>
<td>UCB_HIRAGANA</td>
</tr>
<tr>
<td>UCB_KATAKANA</td>
</tr>
<tr>
<td>UCB_HANGUL_SYLLABLES</td>
</tr>
</tbody>
</table>

"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:

<table>
<thead>
<tr>
<th>Subset</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS_FULLWIDTH_DIGITS</td>
</tr>
<tr>
<td>IS_FULLWIDTH_LATIN</td>
</tr>
<tr>
<td>IS_HALFWIDTH_KATAKANA</td>
</tr>
<tr>
<td>IS_HANJA</td>
</tr>
<tr>
<td>IS_KANJI</td>
</tr>
<tr>
<td>IS_LATIN</td>
</tr>
<tr>
<td>IS_LATIN_DIGITS</td>
</tr>
<tr>
<td>IS_SIMPLIFIED_HANZI</td>
</tr>
<tr>
<td>IS_TRADITIONAL_HANZI</td>
</tr>
</tbody>
</table>

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

**Field Summary**

<table>
<thead>
<tr>
<th>Field Summary</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public static final</code></td>
<td>ANY</td>
</tr>
<tr>
<td><code>public static final</code></td>
<td>CONSTRAINT_MASK</td>
</tr>
<tr>
<td><code>public static final</code></td>
<td>CURRENCY</td>
</tr>
<tr>
<td><code>public static final</code></td>
<td>DECIMAL</td>
</tr>
<tr>
<td><code>public static final</code></td>
<td>EMAILADDR</td>
</tr>
<tr>
<td><code>public static final</code></td>
<td>INITIAL_CAPS_SENTENCE</td>
</tr>
<tr>
<td><code>public static final</code></td>
<td>INITIAL_CAPS_WORD</td>
</tr>
</tbody>
</table>

- **ANY**
  - The user is allowed to enter any text.
  - Value: 0

- **CONSTRAINT_MASK**
  - The mask value for determining the constraint mode.
  - Value: 65535

- **CURRENCY**
  - The user is allowed to enter values that correspond to a currency amount.
  - Value: 6

- **DECIMAL**
  - The user is allowed to enter numeric values with optional decimal fractions, for example ".123", "0.123", or ".5".
  - Value: 5

- **EMAILADDR**
  - The user is allowed to enter an e-mail address.
  - Value: 1

- **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

- **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
<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NON_PREDICTIVE</td>
<td>524288</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>2</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>65536</td>
</tr>
<tr>
<td>PHONENUMBER</td>
<td>3</td>
</tr>
<tr>
<td>SENSITIVE</td>
<td>262144</td>
</tr>
<tr>
<td>UNEDITABLE</td>
<td>131072</td>
</tr>
<tr>
<td>URL</td>
<td>4</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**

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>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>int getMaxSize()</strong></td>
<td>Returns the maximum size (number of characters) that can be stored in this TextField.</td>
</tr>
<tr>
<td><strong>java.lang.String getString()</strong></td>
<td>Gets the contents of the TextField as a string value.</td>
</tr>
<tr>
<td><strong>void insert(char[] data, int offset, int length, int position)</strong></td>
<td>Inserts a subrange of an array of characters into the contents of the TextField.</td>
</tr>
<tr>
<td><strong>void insert(String src, int position)</strong></td>
<td>Inserts a string into the contents of the TextField.</td>
</tr>
<tr>
<td><strong>void setCaret(int index)</strong></td>
<td>Sets the index of the caret.</td>
</tr>
<tr>
<td><strong>void setChars(char[] data, int offset, int length)</strong></td>
<td>Sets the contents of the TextField from a character array, replacing the previous contents.</td>
</tr>
<tr>
<td><strong>void setConstraints(int constraints)</strong></td>
<td>Sets the input constraints of the TextField.</td>
</tr>
<tr>
<td><strong>void setHighlight(int index, int length)</strong></td>
<td>Sets the highlight on a range in the text.</td>
</tr>
<tr>
<td><strong>void setInitialInputMode(String characterSubset)</strong></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>
<tr>
<td><strong>int setMaxSize(int maxSize)</strong></td>
<td>Sets the maximum size (number of characters) that can be contained in this TextField.</td>
</tr>
<tr>
<td><strong>void setString(String text)</strong></td>
<td>Sets the contents of the TextField as a string value, replacing the previous contents.</td>
</tr>
<tr>
<td><strong>int size()</strong></td>
<td>Gets the number of characters that are currently stored in this TextField.</td>
</tr>
</tbody>
</table>

Methods inherited from class `javax.microedition.lcdui.Item`

- `addCommand`, `getCommands`, `getLabel`, `getLayout`, `getLayoutHint`, `getMinimumHeight`, `getMinimumWidth`, `getPreferredHeight`, `getPreferredWidth`, `notifyStateChanged`, `removeCommand`, `setCommand`, `setDefaultCommand`, `setItemCommandListener`, `setLabel`, `setLayout`, `setLayoutHint`, `setPreferredSize`

Methods inherited from class `Object`

- `equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`
The user is allowed to enter any text. Line breaks may be entered.

Constant 0 is assigned to ANY.
Constant value: 0

---

**CONSTRAINT_MASK**

```java
public static final int 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 0xFFFF is assigned to `CONSTRAINT_MASK`.
Constant value: 65535

---

**CURRENCY**

```java
public static final int CURRENCY
```

The user is allowed to enter values that correspond to a currency amount. The format of the amount is the same as with the `DECIMAL` constant, but the implementation SHOULD present the number as an amount in the currency attached to the locale of the device, as determined by the `microedition.locale` system property.

Information about the currency symbol and its position in relation to the number SHOULD be retrieved from the Common Locale Data Repository (CLDR, see [http://unicode.org/cldr/](http://unicode.org/cldr/)).

Examples of currency amount presentations for the number "12345.67" include:

<table>
<thead>
<tr>
<th>Locale</th>
<th>Presentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>fi-FI</td>
<td>12 345,67 €</td>
</tr>
<tr>
<td>en-US</td>
<td>$12,345.67</td>
</tr>
<tr>
<td>en-GB</td>
<td>£12,345.67</td>
</tr>
<tr>
<td>ja-JP</td>
<td>¥ 12,345.67</td>
</tr>
</tbody>
</table>

The implementation MAY defer the presentation until the focus leaves the input field.

Constant 6 is assigned to `CURRENCY`.
Constant value: 6

**Since:** MIDP 3.0

---

**DECIMAL**

```java
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 [JLS], 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 implementation MAY present the field with device-specific characters as part of a numeric 
string, such as spaces or commas for grouping separators. However, the only characters allowed in 
the actual contents of the text object are the optional minus sign, the fraction separator and the 
numeric digits.

The sign and separators 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**

```java
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
public static final int 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. 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
public static final int NUMERIC
```

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.

The implementation MAY present the field with device-specific characters as part of a numeric string, such as spaces or commas for grouping separators. However, the only characters allowed in the actual contents of the text object are the optional minus sign and the numeric digits.

Constant `2` is assigned to `NUMERIC`.  
Constant value: `2`

---

**PASSWORD**

```java
public static final int PASSWORD
```
javax.microedition.lcdui.TextField

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 be 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, but typically includes non-numeric characters, such as a "+" prefix character. At least numeric digits, the plus sign, star and hash MUST be supported. The input of other characters SHOULD be allowed. Any character MAY be ignored by the implementation.

Implementations 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

`public static final int URL`

The user is allowed to enter a URL.

Constant `4` is assigned to `URL`.  
Constant value: `4`

---

### Constructors

**TextField**

`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

---

### Methods

**delete**

`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  

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  

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: setChars(char[], int, int)  

getConstraints  

public int getConstraints()  

Gets the current input constraints of the TextField.  

Returns:  
the current constraints value (see input constraints)  

See Also: setConstraints(int)  

getMaxSize  

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: setSize(int)  

getString  

public java.lang.String getString()  

Gets the contents of the TextField as a string value.  

Returns:
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 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

insert

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 determine 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

setCaret

public void setCaret(int index)
Sets the index of the caret. The caret can be used to indicate a position in the text and MUST be visible when the TextField is in focus. If characters are inserted, the caret index is increased by the number of characters inserted at indexes before or equal to the caret index. If characters are deleted, the caret index is decreased by the number of characters deleted from indexes before or equal to the caret index.

**Parameters:**
- `index` - an integer between 0 and `size()`, inclusive, indicating where in the text to place the caret (0 being immediately before the first character and `size()` being immediately after the last)

**Throws:**
- `IndexOutOfBoundsException` - if `index` is not in the range 0 to `size()`, inclusive

**Since:** MIDP 3.0

### setChars

```java
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:** `getChars(char[])`

### setConstraints

```java
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:** `getConstraints()`

### setHighlight

```java
public void setHighlight(int index,
                         int length)
```

Sets the highlight on a range in the text.

**Parameters:**
- `index` - the index of the first character to be highlighted.
length - the length in characters to be highlighted.

**Since:** MIDP 3.0

### `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:** `getMaxSize()`

### `setString`

**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:** `getString()`

### `size`

**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

public class Ticker

Object

+-javax.microedition.lcdui.Ticker

Description

Implements a “ticker-tape”, a piece of text that runs continuously across the display. The direction and speed of scrolling are determined by the implementation. While animating, the ticker string scrolls continuously. That is, when the string finishes scrolling off the display, the ticker starts over at the beginning of the string.

There is no API provided for starting and stopping the ticker. The application model is that the ticker is always scrolling continuously. However, the implementation is allowed to pause the scrolling for power consumption purposes, for example, if the user doesn’t interact with the device for a certain period of time. The implementation should resume scrolling the ticker when the user interacts with the device again.

The text of the ticker may contain line breaks. The complete text MUST be displayed in the ticker; line break characters should not be displayed but may be used as separators.

The same ticker may be shared by several Displayable objects (“screens”). This can be accomplished by calling setTicker() 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>public Ticker(String str)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructs a new Ticker object, given its initial contents string.</td>
</tr>
</tbody>
</table>

Method Summary

| java.lang.String | getString() |
| --- |
| Gets the string currently being scrolled by the ticker. |

| void | setString(String str) |
| --- |
| Sets the string to be displayed by this ticker. |

Methods inherited from class Object

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

Ticker
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
public java.lang.String getString()

Gets the string currently being scrolled by the ticker.

Returns:
string of the ticker

See Also: setString(String)

setString
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: 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.

13.1 API Overview

Unless otherwise noted, passing a null argument to a constructor or method in any class or interface
in this package MUST cause a NullPointerException to be thrown.

The API is comprised of five classes:

13.1.1 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.

13.1.2 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.

13.1.3 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.

13.1.4 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.

13.1.5 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 a comprised of a grid of cells, and each cell can display one of several tiles that are provided by a single Image object. Cells can also be filled with animated tiles whose corresponding pixel data can be changed very rapidly; this feature is very useful for animating large groups of cells such as areas of water.

Since: MIDP 2.0
## Class Summary

<table>
<thead>
<tr>
<th>Classes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GameCanvas</td>
<td>The GameCanvas class provides the basis for a game user interface.</td>
</tr>
<tr>
<td>Layer</td>
<td>A Layer is an abstract class representing a visual element of a game.</td>
</tr>
<tr>
<td>LayerManager</td>
<td>The LayerManager manages a series of Layers.</td>
</tr>
<tr>
<td>Sprite</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>TiledLayer</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>
javax.microedition.lcdui.game

GameCanvas

Declaration

public abstract class GameCanvas extends javax.microedition.lcdui.Canvas

Object

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.microedition.lcdui.Displayable</td>
<td></td>
</tr>
<tr>
<td>javax.microedition.lcdui.Canvas</td>
<td></td>
</tr>
<tr>
<td>javax.microedition.lcdui.game.GameCanvas</td>
<td></td>
</tr>
</tbody>
</table>

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>Field Name</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>DOWN_PRESSED</td>
</tr>
<tr>
<td>FIRE_PRESSED</td>
</tr>
<tr>
<td>GAME_A_PRESSED</td>
</tr>
<tr>
<td>GAME_B_PRESSED</td>
</tr>
<tr>
<td>GAME_C_PRESSED</td>
</tr>
<tr>
<td>Method</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>public static final GAME_D_PRESSED</td>
</tr>
<tr>
<td>public static final LEFT_PRESSED</td>
</tr>
<tr>
<td>public static final RIGHT_PRESSED</td>
</tr>
<tr>
<td>public static final UP_PRESSED</td>
</tr>
</tbody>
</table>

### Fields inherited from class `javax.microedition.lcdui.Canvas`

- `ACTIONS_ALL`, `ACTIONS_NAVIGATION`, `ACTIONS_NONE`, `DOWN`, `FIRE`, `GAME_A`, `GAME_B`, `GAME_C`, `GAME_D`, `KEY_BACKSPACE`, `KEY_DELETE`, `KEY_DOWN`, `KEY_ENTER`, `KEY_ESCAPE`, `KEY_LEFT`, `KEY_NUM0`, `KEY_NUM1`, `KEY_NUM2`, `KEY_NUM3`, `KEY_NUM4`, `KEY_NUM5`, `KEY_NUM6`, `KEY_NUM7`, `KEY_NUM8`, `KEY_NUM9`, `KEY_POUND`, `KEY_RIGHT`, `KEY_SELECT`, `KEY_SPACE`, `KEY_STAR`, `KEY_TAB`, `KEY_UP`, `LEFT`, `RIGHT`, `UP`

### Constructor Summary

- protected `GameCanvas(boolean suppressKeyEvents)`
  Creates a new instance of a GameCanvas.
- protected `GameCanvas(boolean suppressKeyEvents, boolean preserveBuffer)`
  Creates a new instance of a GameCanvas.

### Method Summary

- 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.
Methods inherited from class Object

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**

```java
public static final int LEFT_PRESSED
```

The bit representing the LEFT key. This constant has a value of \(0x0004 \ (1 \ll \text{Canvas.LEFT})\).

Constant value: 4

**RIGHT_PRESSED**

```java
public static final int RIGHT_PRESSED
```

The bit representing the RIGHT key. This constant has a value of \(0x0020 \ (1 \ll \text{Canvas.RIGHT})\).

Constant value: 32

**UP_PRESSED**

```java
public static final int UP_PRESSED
```

The bit representing the UP key. This constant has a value of \(0x0002 \ (1 \ll \text{Canvas.UP})\).

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.

---

**GameCanvas**

```java
protected GameCanvas(boolean suppressKeyEvents,
                       boolean preserveBuffer)
```

Mobile Information Device Profile v3.0 - JSR 271
Java Community Process - Final Release
© Copyright Motorola, Inc. 2002-2009. All Rights Reserved.
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.

The developer can control preserving the buffer after flushGraphics call: if preserveBuffer == true then the buffer is preserved (MIDP 2.0 behavior), if preserveBuffer == false then after the flushGraphics call the buffer is undefined. In addition, calling flushGraphics(x,y,width,height) must throw an exception if preserveBuffer == false.

Parameters:
- suppressKeyEvents - true to suppress the regular key event mechanism for game keys, otherwise false.
- preserveBuffer - true to preserve the buffer after flushGraphics, false to indicate that after flushGraphics call the buffer is undefined.

See Also: flushGraphics(), flushGraphics(int, int, int, int)  
Since: MIDP 3.0

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: 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.
getGraphics

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: flushGraphics(), flushGraphics(int, int, int, int)

getKeyStates

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:** UP_PRESSED, DOWN_PRESSED, LEFT_PRESSED, RIGHT_PRESSED, FIRE_PRESSED, GAME_A_PRESSED, GAME_B_PRESSED, GAME_C_PRESSED, GAME_D_PRESSED

---

```java
public void paint(Graphics g)
```

Paints this GameCanvas. By default, this method renders 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 `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

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int getHeight()</code></td>
<td>Gets the current height of this layer, in pixels.</td>
</tr>
<tr>
<td><code>int getWidth()</code></td>
<td>Gets the current width of this layer, in pixels.</td>
</tr>
<tr>
<td><code>int getX()</code></td>
<td>Gets the horizontal position of this Layer's upper-left corner in the painter's coordinate system.</td>
</tr>
<tr>
<td><code>int getY()</code></td>
<td>Gets the vertical position of this Layer's upper-left corner in the painter's coordinate system.</td>
</tr>
<tr>
<td><code>boolean isVisible()</code></td>
<td>Gets the visibility of this Layer.</td>
</tr>
<tr>
<td><code>void move(int dx, int dy)</code></td>
<td>Moves this Layer by the specified horizontal and vertical distances.</td>
</tr>
<tr>
<td><code>abstract void paint(Graphics g)</code></td>
<td>Paints this Layer if it is visible.</td>
</tr>
<tr>
<td><code>void setPosition(int x, int y)</code></td>
<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><code>void setVisible(boolean visible)</code></td>
<td>Sets the visibility of this Layer.</td>
</tr>
</tbody>
</table>
Methods inherited from class java.lang.Object

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

Methods

getHeight

public final int getHeight()

Gets the current height of this layer, in pixels.

Returns:
the height in pixels

See Also: getWidth()

getWidth

public final int getWidth()

Gets the current width of this layer, in pixels.

Returns:
the width in pixels

See Also: getHeight()

getX

public final int getX()

Gets the horizontal position of this Layer's upper-left corner in the painter's coordinate system.

Returns:
the Layer's horizontal position.

See Also: getHeight(), setPosition(int, int), move(int, int)

getY

public final int getY()

Gets the vertical position of this Layer's upper-left corner in the painter's coordinate system.

Returns:
the Layer's vertical position.

See Also: getWidth(), setPosition(int, int), move(int, int)

isVisible

public final boolean isVisible()

Gets the visibility of this Layer.

Returns:
true if the Layer is visible, false if it is invisible.
move

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: setPosition(int, int), getX(), getY()
javax.microedition.lcdui.game

LayerManager

Declaration

public class LayerManager

Object

|--javax.microedition.lcdui.game.LayerManager

Description

The LayerManager manages a series of Layers. The LayerManager simplifies the process of rendering the Layers that have been added to it by automatically rendering the correct regions of each Layer in the appropriate order.

The LayerManager maintains an ordered list to which Layers can be appended, inserted and removed. A Layer's index correlates to its z-order; the layer at index 0 is closest to the user while a the Layer with the highest index is furthest away from the user. The indices are always contiguous; that is, if a Layer is removed, the indices of subsequent Layers will be adjusted to maintain continuity.

The LayerManager class provides several features that control how the game's Layers are rendered on the screen.

The view window controls the size of the visible region and its position relative to the LayerManager's coordinate system. Changing the position of the view window enables effects such as scrolling or panning the user's view. For example, to scroll to the right, simply move the view window's location to the right. The size of the view window controls how large the user's view will be, and is usually fixed at a size that is appropriate for the device's screen.

In this example, the view window is set to 85 x 85 pixels and is located at (52, 11) in the LayerManager's coordinate system. The Layers appear at their respective positions relative to the LayerManager's origin.

The paint(Graphics, int, int) 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.

![Game screenshot](image)

**Since:** MIDP 2.0

### Constructor Summary

<table>
<thead>
<tr>
<th>Public Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>public <strong>LayerManager()</strong></td>
</tr>
</tbody>
</table>
| Creates a new LayerManager.

### Method Summary

<table>
<thead>
<tr>
<th>Method Signature</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append(Layer l)</strong></td>
<td>Appends a Layer to this LayerManager.</td>
</tr>
<tr>
<td><strong>getLayerAt(int index)</strong></td>
<td>Gets the Layer with the specified index.</td>
</tr>
<tr>
<td><strong>getSize()</strong></td>
<td>Gets the number of Layers in this LayerManager.</td>
</tr>
<tr>
<td><strong>insert(Layer l, int index)</strong></td>
<td>Inserts a new Layer in this LayerManager at the specified index.</td>
</tr>
<tr>
<td><strong>paint(Graphics g, int x, int y)</strong></td>
<td>Renders the LayerManager's current view window at the specified location.</td>
</tr>
<tr>
<td><strong>remove(Layer l)</strong></td>
<td>Removes the specified Layer from this LayerManager.</td>
</tr>
<tr>
<td><strong>setViewWindow(int x, int y, int width, int height)</strong></td>
<td>Sets the view window on the LayerManager.</td>
</tr>
</tbody>
</table>

Methods inherited from class `Object`
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: insert(Layer, int), 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:
- 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 this LayerManager minus one and the Layer has already been added to this LayerManager.

See Also: append(Layer), remove(Layer)

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: setViewWindow(int, int, int, int)

public void remove(Layer l)
Remove the specified Layer from this LayerManager. This method does nothing if the specified Layer is not added to the this LayerManager.

Parameters:

- \( l \) - the Layer to be removed

Throws:

- NullPointerException - if the specified Layer is null

See Also: append(Layer), insert(Layer, int)

setViewWindow

```java
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 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
getRawFrameCount() returns the total number of raw frames.

A ScalableImage object MAY be used to create a Sprite or change its appearance using the setImage method. If such an Image is used, the implementation MUST behave as if a rasterized snapshot of the image is taken when the constructor or method is called; the dimensions of the snapshot MUST reflect the ScalableImage's current viewport size as reported by its getWidth and getHeight methods. Subsequent changes to the viewport or content of the ScalableImage have no effect on the size or appearance of the Sprite.

Frame Sequence

A Sprite's frame sequence defines an ordered list of frames to be displayed. The default frame sequence mirrors the list of available frames, so there is a direct mapping between the sequence index and the corresponding frame index. This also means that the length of the default frame sequence is equal to the number of raw frames. For example, if a Sprite has 4 frames, its default frame sequence is \{0, 1, 2, 3\}.

The developer must manually switch the current frame in the frame sequence. This may be accomplished by calling setFrame(int), prevFrame(), or 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 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 setPosition(x,y), getX(), and 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 *reference pixel*. The reference pixel is defined by specifying its location in the Sprite's untransformed frame using `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:

```
defineReferencePixel(25, 3)
```

`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:

```javascript
setRefPixelPosition(48, 22)
```

### 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 `getRefPixelX()` and `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:
javax.microedition.lcdui.game.Sprite

TRANS_NONE
TRANS_ROT180
TRANS_MIRROR
TRANS_MIRROR_ROT180
TRANS_ROT90
TRANS_MIRROR_ROT90
TRANS_MIRROR_ROT270
TRANS_ROT270
Sprite Drawing

Sprites can be drawn at any time using the `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

<table>
<thead>
<tr>
<th>Field Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public static final</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><code>public static final</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><code>public static final</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><code>public static final</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><code>public static final</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><code>public static final</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><code>public static final</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><code>public static final</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public Sprite(Image image)</code></td>
</tr>
<tr>
<td>Creates a new non-animated Sprite using the provided Image.</td>
</tr>
</tbody>
</table>
| 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     | **setCollisionAlpha** *(int alpha)*  
|          | Sets the alpha value for pixel level collision detection for this Sprite object. |
| void     | **setFrame** *(int sequenceIndex)*  
|          | Selects the current frame in the frame sequence. |
| void     | **setFrameSequence** *(int[] sequence)*  
|          | Set the frame sequence for this Sprite. |
javax.microedition.lcdui.game.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, 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**

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**

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**

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**

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
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
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
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

Sprite
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 image is null

Sprite
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 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

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 pixels with alpha values greater than or equal to the collision alpha threshold value (set by setCollisionAlpha() for this sprite) collide. That is, a pixel with alpha greater than or equal to the collision alpha threshold in this Sprite would have to collide with a pixel with alpha greater than or equal to the collision alpha threshold in the specified Sprite 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 pixels with alpha values greater than or equal to the collision alpha threshold value (set by setCollisionAlpha() for this sprite) collide. That is, a pixel with alpha greater than or equal to the collision alpha threshold in this Sprite would have to collide with a pixel with alpha greater than or equal to the collision alpha threshold in the specified 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 transform 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 pixels with alpha values greater than or equal to the collision alpha threshold value (set by setCollisionAlpha() for this sprite) collide. That is, a pixel with alpha greater than or equal to the collision alpha threshold in this Sprite would have to collide with a pixel with alpha greater than or equal to the collision alpha threshold in the specified Sprite 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**

```java
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 `getX()` and `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:** `setRefPixelPosition(int, int), getRefPixelX(), 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:** `setFrameSequence(int[]), 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: `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: `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: `defineReferencePixel(int, int), setRefPixelPosition(int, int), getRefPixelY()`

---

**getRefPixelY**

```java
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: `defineReferencePixel(int, int), setRefPixelPosition(int, int), getRefPixelX()`

---

**nextFrame**

```java
public void nextFrame()
```

Selects the next frame in the frame sequence.

The frame sequence is considered to be circular, i.e. if `nextFrame()` is called when at the end of the sequence, this method will advance to the first entry in the sequence.

See Also: `setFrameSequence(int[]), prevFrame()`

---

**paint**

```java
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
prevFrame

public void prevFrame()

Selects the previous frame in the frame sequence.

The frame sequence is considered to be circular, i.e. if prevFrame() is called when at the start of the sequence, this method will advance to the last entry in the sequence.

See Also: setFrameSequence(int[]), nextFrame()

setCollisionAlpha

public void setCollisionAlpha(int alpha)

Sets the alpha value for pixel level collision detection for this Sprite object. All pixel level collision detection operations performed, using the collidesWith() method, on this Sprite object will be performed with this collision alpha threshold value.

An alpha value of 255 is fully opaque, and a value of 0 is fully transparent. The alpha value is 255 (fully opaque) by default for detecting collisions unless changed by the use of this method.

Parameters:
alpha - the alpha value to be used for collision detection

Throws:
IllegalArgumentException - if the alpha value is outside the range of 0-255

Since: MIDP 3.0

setFrame

public void setFrame(int sequenceIndex)

Selects the current frame in the frame sequence.

The current frame is rendered when 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: setFrameSequence(int[]), getFrame()

setFrameSequence

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 getRawFrameCount()
- IllegalArgumentException - if the array has less than 1 element

**See Also:** nextFrame(), prevFrame(), setFrame(int), getFrame()

### setImage

```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 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 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 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

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: defineReferencePixel(int, int), getRefPixelX(), getRefPixelY()

setTransform

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 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 getRefPixelX() and 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: TRANS_NONE, TRANS_ROT90, TRANS_ROT180, TRANS_ROT270, TRANS_MIRROR, TRANS_MIRROR_ROT90, TRANS_MIRROR_ROT180, TRANS_MIRROR_ROT270
javax.microedition.lcdui.game

TiledLayer

Declaration

public class TiledLayer extends Layer

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 `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 `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 `setAnimatedTile(int, int)` method.

A ScalableImage object MAY be used to create a TiledLayer or change its appearance using the `setStaticTileSet` method. If such an Image is used, the implementation MUST behave as if a rasterized snapshot of the image is taken when the constructor or method is called; the dimensions of the snapshot MUST reflect the ScalableImage's current viewport size as reported by its `getWidth` and `getHeight` methods. Subsequent changes to the viewport or content of the ScalableImage have no effect on the size or appearance of the TiledLayer.

### Cells

The TiledLayer's grid is made up of equally sized cells; the number of rows and columns in the grid are specified in the constructor, and the physical size of the cells is defined by the size of the tiles.

The contents of each cell is specified by means of a tile index; a positive tile index refers to a static tile, and a negative tile index refers to an animated tile. A tile index of 0 indicates that the cell is empty; an empty cell is fully transparent and nothing is drawn in that area by the TiledLayer. By default, all cells contain tile index 0.

The contents of cells may be changed using `setCell(int, int, int)` and `fillCells(int, int, 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 Example](image_url)

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)`.

### Rendering a TiledLayer

```java
javax.microedition.lcdui.game.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

```java
public TiledLayer(int columns, int rows, Image image, int tileWidth, int tileHeight)

Creates a new TiledLayer.
```

### Method Summary

```java
tileIndex)

Creates a new animated tile and returns the index that refers to the new animated tile.

Fills a region cells with the specific tile.

Gets the tile referenced by an animated tile.

Gets the contents of a cell.

Gets the height of a single cell, in pixels.

Gets the width of a single cell, in pixels.

```

Fills a region cells with the specific tile.
```

```java
Gets the tile referenced by an animated tile.
```

```java
Gets the contents of a cell.
```

```java
`int getCellWidth()`

`int getCellHeight()`
```
javax.microedition.lcdui.game.TiledLayer

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void setAnimatedTile(int animatedTileIndex, int staticTileIndex)</td>
<td>Associates an animated tile with the specified static tile.</td>
</tr>
<tr>
<td>void setCell(int col, int row, int tileIndex)</td>
<td>Sets the contents of a cell.</td>
</tr>
<tr>
<td>void setStaticTileSet(Image image, int tileWidth, int tileHeight)</td>
<td>Change the static tile set.</td>
</tr>
</tbody>
</table>

Methods inherited from class `javax.microedition.lcdui.game.Layer`

- getHeight, getWidth, getX, getY, isVisible, move, setPosition, setVisible

Methods inherited from class `Object`

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

Constructors

```java
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 `setCell(int, int, int)` and `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 `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

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)

Returns:
the index of newly created animated tile

Throws:
IndexOutOfBoundsException - if the staticTileIndex is invalid

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: setCell(int, int, int), 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 referenced by the animated tile

Throws:
IndexOutOfBoundsException - if the animated tile index is invalid

See Also: setAnimatedTile(int, int)
getCell

```java
public int getCell(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:** `setCell(int, int, int), fillCells(int, int, int, int, int)`

getCellHeight

```java
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

```java
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

```java
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

```java
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

```java
public final void paint(Graphics g)
```
javax.microedition.lcdui.game.TiledLayer

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

See Also: `getAnimatedTile(int)`

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: `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: `getCell(int, int), fillCells(int, int, int, int, int)`

setStaticTileSet

public void `setStaticTileSet`(Image image, int tileWidth, int tileHeight)
javax.microedition.lcdui.game.TiledLayer

Change the static tile set.

Replaces the current static tile set with a new static tile set. See the constructor `TiledLayer(int, int, Image, int, int)` 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 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`
Chapter 14

Package
javax.microedition.media

Description

MIDP 3.0 requires the support of the JSR 135 Mobile Media API v1.1 specification. Compatible implementations of MIDP 3.0 MUST include a compatible implementation of JSR 135 v1.1 specification or a later Maintenance Release thereof. A later subsequent version of the Mobile Media API may also be used as long as the later version is backwards compatible with previous versions of the JSR 135 specification.

The design of MMAPI allows implementations to provide optional support for different media types and protocols. MIDP3.0 implementation MUST guarantee support for the following media types:

- Support for tone generation
- Support for wav format if the device supports sampled audio

Note: The MIDP 2.x specifications contained a specific building block which was a proper subset of [JSR135]. MIDP 3.0 requires the support of the full [JSR135] specification, therefore being backwards compatible with the building block included in MIDP 2.x specifications.

For detailed specification of this API, see [JSR135].

Since: MIDP 3.0
Package
javax.microedition.midlet

Description
The MIDlet package defines Mobile Information Device Profile (MIDP) 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.

Unless otherwise noted, passing a null argument to a constructor or method in any class or interface in this package MUST cause a java.lang.NullPointerException to be thrown.

15.1 MIDlets

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 execution environment is available to the MIDlet, and how it should behave so that the device can manage its resources. The application model defines how multiple MIDlets forming a suite can be packaged together.

A MIDlet is the entity that is launched by the Application Management Software (AMS). Each MIDlet consists of a class that extends the javax.microedition.midlet.MIDlet class and other classes as may be needed by the MIDlet. When a MIDlet 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 inform the MIDlet that it has been started or destroyed.

A MIDlet suite is a collection of MIDlets packaged in a JAR as described in MIDlet Suite Packaging. The manifest in the JAR contains attributes that are used during installation and execution of MIDlets.

Each MIDlet is executed in a separate execution environment that MUST not share any VM resources with any other MIDlet. All of the MIDlets in a MIDlet suite are subject to a single set of policies and controls. Refer to MIDP 3.0 MIDlet Concurrency for more requirements.

A MIDP application MUST use only functionality specified by the MIDP specification, the requested configuration, optional packages, and Licensee Open Classes as it is developed, tested, deployed, and run.

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.

15.2 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 introduced 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.

One aspect of the MIDlet security model is that each MIDlet suite runs in isolation, and cannot gain access to the variables, data, memory, etc. of a MIDlet in a different MIDlet. This is important because the integrity and privacy of the MIDlet data must be maintained and protected from rogue MIDlets either attempting to ascertain secret information or being destructive. With the LIBlet development model, where the LIBlet code may be developed and deployed independently (any by independent entities) from the MIDlet code, yet where the two are bound together to run in one execution environment, the security risks are significant. A LIBlet can gain access to MIDlet private data, can modify this data freely, and can even inherit the security privileges granted to the MIDlet.

- By itself, a LIBlet, unlike a MIDlet, is not associated with a domain.
- Permissions while executing LIBlet code are the permissions associated with the MIDlet's execution environment.
- Trust level (determining if the LIBlet has been tampered with) is handled by placing the LIBlet JAR hash in the dependent MIDlets JAD.

### 15.3 MIDP Execution Environment

The MIDP defines the execution environment provided to MIDlets. Each MIDlet executed receives its own execution environment. The loading of classes, class state, static variable data, and static initializers is managed per execution environment. The application management software initiates the applications and makes the following available to the MIDlet:

- Classes and native code that implement the supporting Java ME Configuration (CLDC or CDC), including a Java Virtual Machine
- Classes and native code that implement the MIDP specification
- Classes from the MIDlet suite JAR for execution
- Classes from each LIBlet required by the MIDlet suite
- Files from the MIDlet suite JAR as resources; class files SHOULD be available
- Files from each LIBlet required by the MIDlet suite as resources; class files SHOULD be available
- Contents of the application descriptor, when it is present
- Any other APIs available on the device such as implementations of additional Optional Packages, Licensee Open Classes, etc.

The CDC or CLDC and Java Virtual Machine provides 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 CDC, CLDC, MIDP, or optional packages.

The behavior of MIDlets in MIDP 3.0 is defined for two different Java ME configurations: CLDC 1.1.1 and CDC. Access to the functionality defined by configurations is controlled by the JAD attribute MicroEdition-Configuration. A MIDlet may choose the appropriate execution environment using the following attribute value definitions:

- If the value of the MicroEdition-Configuration attribute in the JAD file of the MIDlet suite is "CLDC-1.1.1", then execution environment MUST conform to the CLDC Specification,
version 1.1.1 or higher and the MIDlet SHALL have access to all of its functionality (within the restrictions imposed by the security policy).

All systems compliant with the MIDP3 Specification MUST provide support for the "CLDC-1.1.1" value of the MicroEdition-Configuration attribute, and the behavior associated with this attribute value.

- If the value of the MicroEdition-Configuration attribute in the JAD file of the MIDlet suite is "CDC-1.1", then execution environment MUST conform to the CDC Specification, version requested or higher and the MIDlet SHALL have access to all of its functionality (within the restrictions imposed by the security policy).

In the CDC environment, the sandbox security requirements defined in the CLDC Specification are superseded by the more fine-grained security model offered by the CDC Specification. Access to CDC-specific functionality is controlled by the security policy.

Support for the "CDC-1.1" values of the MicroEdition-Configuration attribute and the associated behavior is OPTIONAL.

If the system cannot honor a MIDlet's request for a particular execution environment, then the installation of the MIDlet MUST fail. Status Code 908 (Incompatible Configuration or Profile) MUST be used when reporting the installation failure.

The MIDP implementation 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 system packages.

A single JAR file contains all the classes of the MIDlet. The MIDlet may load and invoke methods from:

- any class in the MIDlet JAR,
- any class in each LIBlet the MIDlet specified a dependency on,
- any class in the MIDP,
- any class in the supporting Java ME Configuration (CLDC or CDC), and
- any class in optional packages.

All of the classes, class state, static variable data, and static initializers within these scopes are unique to the execution environment provided by the AMS for the MIDlet. All states accessible via those classes are available to any Java class running on behalf of the MIDlet.

When a MIDlet that depends on one or more LIBlets is launched, all the code of the MIDlet and LIBlets is executed in an execution environment, indistinguishable from a standalone MIDlet packaging all the MIDlet and LIBlet code in a single MIDlet JAR. That is, the MIDlet and LIBlets it depends on are executed on the same Java heap, in the same Protection Domain, and with the same security policy/manager.

The act of loading a MIDlet with all the associated LIBlets it declares dependencies on directly or indirectly into a single execution environment is called binding. Once bound, the LIBlet code is executed within the MIDlet execution environment as if it were packaged into the MIDlet JAR. Thus, although a LIBlet bound to different MIDlets may access shared persistent resources through RMS or other persistent storage, the execution of its code inside different bindings will be completely isolated from each other.

The class files of the MIDlet or dependent LIBlets MUST be available for execution and SHOULD be available as resources. The implementation MAY store and interpret the contents of the JAR in any manner suitable.
The files from the JAR that are not Java class files are made available via
`java.lang.Class.getResourceAsStream`; for example, the contents of the manifest file are
made 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(String key)`
method. This method can not be used to retrieve contents of the LIBlet descriptor file. The
`javax.microedition.midlet.MIDlet.getAppProperty(String name, String
vendor, String attributeName, String attributeDelimiter)` method is used to
retrieve and at the same time parse application property values from the application descriptor and
the manifest of the MIDlet or bound LIBlets. This method retrieves a property value for a specific
LIBlet or MIDlet, so the same property name is allowed to be used by multiple LIBlets without
ambiguity in the binding.

15.4 Screen Size Requirements

An application MAY define the minimum and/or maximum screen sizes that it is designed to work
with using the `MIDlet-Minimum-Canvas-Size` and/or `MIDlet-Maximum-Canvas-Size`
attributes. If either of the `MIDlet-Minimum-Canvas-Size` or `MIDlet-Maximum-Canvas-Size`
attributes are specified, and the primary display of the device is incapable of meeting the specified
size requirements, then the MIDlet suite installation MUST fail (see Installation for more details).

On devices that support multiple orientations, the MIDlet suite MAY be installed if the application's
size requirements can be met using one or more of its available orientations. Implementations
SHOULD select a display and orientation that meets the application's display requirements.

15.5 Screen Saver MIDlets

A screen saver MIDlet is an application that can be launched automatically when the device has
been in the idle state for a predetermined amount of time. The idle state is typically a state in which
the device is not actively being used; for example, if there is no call ongoing and the user has not
pressed any keys. The precise definition of the idle state depends on the device implementation.
The screen saver keeps running until it is deactivated by an external event, such as a user action or
by a platform dependent timeout.

15.5.1 Screen Saver Identification

Screen saver MIDlets are identified with the Application Attribute `MIDlet-<n>-Type`. The type
for identifying a screen saver MIDlet is `screensaver`. In order to indicate a screen saver MIDlet
in the MIDlet suite, either the JAD file or the JAR manifest MUST have a `MIDlet-<n>-Type:
screensaver` entry.

For example:

```
MIDlet-1: Cool3DSaver, /icon.png, ScreenSaverMIDlet
MIDlet-1-Type: screensaver
```

15.5.2 Screen Saver Provisioning

MIDlet suites that contain a screen saver MIDlet are provisioned in the same way as all other MIDlet
suites. The following additional requirements apply:

- If a MIDlet suite that is being updated contains a screen saver MIDlet that has been previously
  selected as the current screen saver, the user SHOULD be notified that the currently selected
  screen saver MIDlet is being updated, and the updated screen saver MIDlet MUST be set as
  the currently selected screen saver. If the new MIDlet suite does not contain a screen saver
MIDlet with the same name as the previously selected screen saver MIDlet, the user SHOULD be notified that the currently selected screen saver is no longer available.

- If a MIDlet suite that is being deleted contains a screen saver MIDlet that has been previously selected as the current screen saver, the user SHOULD be notified that the current screen saver MIDlet is being deleted.

15.5.3 Screen Saver MIDlet Execution

In order to receive events corresponding to activation and deactivation of the screen saver, a MIDlet must register a `EventDataListener` with `EventManager.addEventListener`. A screen saver MIDlet should register such a listener immediately after it has been started, as soon as it is ready to act as a screen saver. The event that corresponds to changes in screen saver state is `SCREENSAVER_MODE`. When a screen saver is activated or deactivated, the `SCREENSAVER_MODE` event MUST be sent to all registered screen saver applications. A MIDlet can then determine if it is the currently selected screen saver by querying `MIDlet.isSelectedScreenSaver`. A MIDlet can also query the state of the `SCREENSAVER_MODE` event with the `EventManager.getCurrent(event)` method. If the currently selected screen saver MIDlet is not running when the screen saver is activated, the AMS MUST start the currently selected screen saver MIDlet.

When a screen saver MIDlet is not running as the active screen saver, it runs like any other MIDlet. This provides the opportunity to allow configuration and preview of the screen saver. A screen saver MIDlet may also add content to the idle screen with `IdleItem`. In this case it MUST also announce itself as an active idle MIDlet through the `MIDlet-<n>-Type` attribute with value `idlescreen`. More information on the idle screen can be found in the `javax.microedition.lcdui` package.

15.5.4 Screen Saver Deactivation

Once in the active mode, a screen saver MIDlet MAY be deactivated by a key event. A device MAY have one or more keys that do not cause the screen saver MIDlet to be terminated in the active mode. These keys can be, for example, used to turn on a dimmed screen backlight. When the screen saver MIDlet is deactivated, the `SCREENSAVER_MODE` event with value `SCREENSAVER_MODE_DEACTIVATED` is sent to all registered applications.

If the screen saver MIDlet uses functionality that is protected with the security policy, the security prompts shown to the user when screen saver is activated are problematic. The security prompts SHOULD NOT deactivate the screen saver. However, this functionality may be impossible or at best difficult to implement. Therefore, sequence of events that the screen saver MIDlet MAY expect are as follows:

- When the screen saver MIDlet is activated, it receives `SCREENSAVER_MODE` event with value `SCREENSAVER_MODE_ACTIVATED`.
- The MIDlet's `Display` capabilities are changed to indicate no further user input events will be delivered to the screen saver MIDlet.
- When the screen saver MIDlet uses functionality that triggers a security dialog or user input widget, the `Display` capabilities are changed to deliver user input and a `SCREENSAVER_MODE` event with value `SCREENSAVER_MODE_DEACTIVATED` is sent to the screen saver MIDlet.
- After the user input is completed, the AMS SHOULD activate the screen saver again and a `SCREENSAVER_MODE` event with value `SCREENSAVER_MODE_ACTIVATED` is sent to the screen saver MIDlet. The `Display` capabilities are changed as appropriate.

An implementation MAY also use system screens to temporarily obscure a screen saver display.

The screen saver MIDlet MAY also be deactivated by implementation specific actions or timeouts; these may include:
javax.microedition.midlet - MIDP MIDlet Lifecycle

- Incoming call or message
- Launching of a MIDlet through PushRegistry
- Uncaught exception in the screen saver MIDlet
- Entering the power save mode of the device
- Activation of device key guard functionality
- MIDlet terminating itself with the `MIDlet.notifyDestroyed()` method

15.5.5 Screen Saver Security

From the security point of view, a screen saver MIDlet is handled as any other MIDlet. As such, if the screen saver MIDlet satisfies security requirements for specific APIs, those APIs MUST be accessible to the MIDlet.

15.6 Idle Screen MIDlets

Most mobile platforms feature some concept of an idle screen - a default display which is presented to the user when no other activity is taking place on the device. Typically the idle screen is not empty but features UI components which provide status information about the device (signal strength, battery levels), information about services on the device (number of unread emails, calendar event information, headlines from a news feed) and shortcuts or controls for popular applications (music players controls). MIDlets can offer the same capability by setting the `IdleItem` for a Display. [Figure 15-2 below](#) shows how idle screen MIDlets can be placed into a device’s idle screen.

| Figure 15-2 : Idle Screen MIDlet Example |
A MIDlet that wants to add content to the idle screen is identified with an application attribute `MIDlet-<n>-Type` with the value `idlescreen`. For example:

```
MIDlet-1: NewsTicker, /icon.png, NewsTickerMIDlet
MIDlet-1-Type: idlescreen
```

More than one MIDlet on the device can add content to the idle screen.

### 15.7 Auto Start MIDlets

MIDlets that are auto-started by the device on power-up and restarted (on exit of the MIDlet) are called "Auto Start MIDlets". The AMS MUST attempt to start the MIDlet on power-up; however, the system MAY fail start such MIDlets due to resource constraints at power-up. In addition, the AMS MUST attempt to restart Auto Start MIDlets on exit/termination of the MIDlet; however, the system may fail to restart the MIDlet due to resource limitations or other system failures. In both these failure cases, the AMS MUST make at least one attempt to restart the MIDlet.

A MIDlet developer marks a MIDlet as Auto Start using the `MIDlet-<n>-Type` attribute. If the attribute value includes `autostart`, the MIDlet assumes the Auto Start MIDlet behavior. The MIDlet
suite must have the `javax.microedition.midlet.AutoStartPermission` for a MIDlet to assume this behavior.

## 15.8 Controlling User actions on a MIDlet Suite

A MIDlet developer may want to restrict the actions allowed on a MIDlet by the user. User actions on a MIDlet are controlled using the `MIDlet-<n>-UserDenied` attribute. Actions include the ability for a user to launch a MIDlet and the ability to stop a running MIDlet from an AMS screen. User actions on a MIDlet Suite are controlled using the `MIDlet-UserDenied` attribute. User actions on a Suite include the ability to update or delete a MIDlet suite.

MIDlet Suites that declare attributes to control user action require the `ActionsDeniedPermission` Permission. If any of the user actions attributes are present and this Permission is not granted, the installation of the MIDlet Suite MUST fail.

If the `MIDlet-<n>-UserDenied` attribute is not present in the JAD/Manifest, no user actions are denied on this MIDlet. If the `MIDlet-UserDenied` attribute is not present in the JAD/Manifest, none of allowed user actions are denied on the MIDlet Suite.

### 15.8.1 Persistent MIDlet Suites

A MIDlet Suite that cannot be deleted by the user is a Persistent MIDlet Suite. A MIDlet developer marks a suite as Persistent using the `MIDlet-UserDenied` attribute. If the attribute value includes `delete`, the suite is a Persistent MIDlet Suite as the user is denied the ability to delete the MIDlet Suite. When a Persistent MIDlet suite is being installed, the user MUST be prompted to confirm that the MIDlet Suite installation can proceed, and that the user will not be able to delete the MIDlet suite after installation.

## 15.9 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 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 [JPVS].

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.

## 15.10 MIDlet Classes

All Java classes needed by the MIDlet are be placed in the JAR 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 with the name `com/sun/microedition/Test.class`.

## 15.11 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. If a MIDlet class identified by `MIDlet-<n>` has a `public static void main(String[])` method, it MUST be ignored by...
the application management software. Unlike other Java application environments, MIDP does not use the public static void main(String[]) method as the entry point of the application. Instead, the methods of the initial MIDlet class are called as defined by the MIDlet lifecycle model.

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.

15.12 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

15.12.1 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.

15.13 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 manufacturers and users expect; specifically:

• The perceived startup latency of an application SHOULD be very short.
• It SHOULD be possible to destroy an application at any time.

ThePaused state described below is included for backward compatibility with previous versions. For MIDP 3.0 MIDlet suites, the AMS does not call MIDlet.pauseApp after it enters the Active state.

The valid states for MIDlets are defined in Table 15-1 below.

| Table 15-1 : MIDlet States |
### javax.microedition.midlet - MIDP MIDlet Lifecycle

<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. |
| 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. 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.

Figure 15-1 below illustrates the state transitions and requirements related to the MIDlet lifecycle.

![Figure 15-1 : MIDlet Lifecycle State Diagram](image-url)
15.14 MIDlet Lifecycle Model

A typical sequence of MIDlet execution is presented in Table 15-2:

<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 MIDlet.</td>
<td>The default (no argument) constructor for the MIDlet is called; it is in the Paused state.</td>
</tr>
<tr>
<td>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.</td>
<td>The MIDlet acquires any resources it needs and begins to perform its service.</td>
</tr>
<tr>
<td>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 to be destroyed by calling the MIDlet.destroyApp method.</td>
<td>If it has been designed to do so, the MIDlet saves state or user preferences and performs clean up.</td>
</tr>
</tbody>
</table>
15.15 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.

15.16 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
 * it is done.
 * Refer to the startApp and destroyApp
 * methods so see how it handles each requested transition.
 */
public class MethodTimes extends MIDlet implements Runnable {
    // The state for the timing thread.
    Thread thread;

    /**
     * Start creates the thread to do the timing.
     * It should return immediately to keep the dispatcher
     * from hanging.
     */
    public void startApp() {
        thread = new Thread(this);
        thread.start();
    }

    /**
     * Destroy must cleanup everything. The thread is signaled
     * to stop and no result is produced.
     */
    public void destroyApp(boolean unconditional) {
        thread = null;
    }

    /**
     * Run the timing test, measure how long it takes to
     * call a empty method 1000000 times.
     * Terminate early if the current thread should no longer be running.
     */
    public void run() {
        Thread curr = Thread.currentThread();

        // Remember which thread is current
        long start = System.currentTimeMillis();
        for (int i = 0; i < 1000000 && thread == curr; i++) {
```
```java
empty();
}
long end = System.currentTimeMillis();

// Check if timing was aborted, if so just exit
// The rest of the application has already become quiescent.
if (thread != curr) {
    return;
}

long millis = end - start;

// Reporting the elapsed time is outside the scope of this example.
// All done cleanup and quit
destroyApp(true);
notifyDestroyed();

/**
 * An Empty method.
 */
void empty() {
}

15.17 Splash Screens

An application may specify a splash screen in its manifest (see Packaging section for more details).

The splash screen is shown only when the application is started by the user. The splash screen is shown on the primary Display.

The implementation should show the splash screen as soon as possible after the MIDlet is launched, but it must guarantee that it is visible to the user by the time the MIDlet's startApp method is called.

Once shown, the splash screen will remain visible until one of the following events occurs:

- The MIDlet shows a Displayable by calling setCurrent on its primary Display
- The MIDlet's startApp method returns
- The MIDlet is no longer in the foreground

Since: MIDP 1.0
### Class Summary

#### Classes

- **ActionsDeniedPermission**
  - ActionsDeniedPermission is required by MIDlets that want to restrict the actions allowed by an user on a MIDlet or a MIDlet Suite.

- **AutoStartPermission**
  - AutoStartPermission allows MIDlets in a MIDlet suite to assume the Auto Start MIDlet behavior.

- **MIDlet**
  - A MIDlet is a Mobile Information Device Profile application.

- **MIDletIdentity**
  - This class represents the identity of a MIDlet.

#### Exceptions

- **MIDletStateChangeException**
  - Signals that a requested MIDlet state change failed.
javax.microedition.midlet

ActionsDeniedPermission

Declaration

public final class ActionsDeniedPermission extends BasicPermission

Object

+--Permission

+--BasicPermission

+--javax.microedition.midlet.ActionsDeniedPermission

Description

ActionsDeniedPermission is required by MIDlets that want to restrict the actions allowed by an user on a
MIDlet or a MIDlet Suite. ActionsDeniedPermission is used only during the installation and hence the user
permission modes oneshot, session and blanket do not apply to this. Therefore,
ActionsDeniedPermission MUST have only two options: allowed or not allowed. ActionsDeniedPermission
class does not support any names and actions.

Since: MIDP 3.0

Constructor Summary

public ActionsDeniedPermission()

Creates a new ActionsDeniedPermission object

Method Summary

boolean implies(Permission permission)

Checks if the specified permission is "implied" by this object.

Methods inherited from class BasicPermission

equals, getActions, hashCode, newPermissionCollection

Methods inherited from class Permission

equals, getActions, getName, hashCode, newPermissionCollection, toString

Methods inherited from class Object

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

Constructors

ActionsDeniedPermission

public ActionsDeniedPermission()

Creates a new ActionsDeniedPermission object
### implies

```java
public boolean implies(Permission permission)
```

Checks if the specified permission is "implied" by this object. Returns `true` iff:

- permission's class is the same as this object's class.

**Parameters:**

- `permission` - the permission to check against.

**Returns:**

- `true` if the specified permission is implied by this object, otherwise `false`. 
AutoStartPermission

Declaration

public final class AutoStartPermission extends BasicPermission

Object

+-Permission
  +-BasicPermission
   +-javax.microedition.midlet.AutoStartPermission

Description

AutoStartPermission allows MIDlets in a MIDlet suite to assume the Auto Start MIDlet behavior. AutoStartPermission is used only during installation and hence the user permission modes oneshot, session and blanket do not apply to this. Therefore, AutoStartPermission MUST have only two options: allowed or not allowed. AutoStartPermission class does not support any names and actions.

Since: MIDP 3.0

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor summary</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>public AutoStartPermission()</td>
<td></td>
</tr>
<tr>
<td>Creates a new AutoStartPermission object</td>
<td></td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method summary</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean implies(Permission permission)</td>
<td></td>
</tr>
<tr>
<td>Checks if the specified permission is &quot;implied&quot; by this object.</td>
<td></td>
</tr>
</tbody>
</table>

Methods inherited from class BasicPermission

equals, getActions, hashCode, newPermissionCollection

Methods inherited from class Permission

equals, getActions, getName, hashCode, newPermissionCollection, toString

Methods inherited from class object

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

Constructors

AutoStartPermission

<table>
<thead>
<tr>
<th>Constructor summary</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>public AutoStartPermission()</td>
<td></td>
</tr>
<tr>
<td>Creates a new AutoStartPermission object</td>
<td></td>
</tr>
</tbody>
</table>

See Also: MIDlet
Methods

implies

public boolean implies(Permission permission)

Checks if the specified permission is "implied" by this object. Returns true iff:

- permission's class is the same as this object's class

Parameters:
permission - the permission to check against.

Returns:
true if the specified permission is implied by this object, otherwise false.
javax.microedition.midlet
MIDlet

Declaration

public abstract class MIDlet

Object

+--javax.microedition.midlet.MIDlet

Description

A MIDlet is a Mobile Information Device 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, 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 a MIDlet. It can select which MIDlet is active at a given time by starting and destroying 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: Some of the methods on this class 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.

Since: MIDP 1.0

Constructor Summary

<table>
<thead>
<tr>
<th>protected MIDlet()</th>
</tr>
</thead>
</table>

Protected constructor for subclasses.

Method Summary

<table>
<thead>
<tr>
<th>int checkPermission(String permission)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deprecated. The permissions model in this specification does not use named permissions.</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>static java.lang.String[] getAppProperty(String name, String vendor, String attributeName, String attributeDelimiter)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provides a mechanism to retrieve and at the same time parse application property values from the application descriptor and the manifest of the MIDlet or bound LIBlets.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>javax.microedition.midlet.MIDletIdentity getMIDletIdentity()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gets a MIDletIdentity instance associated with this MIDlet, which can be used to obtain a MIDlet's name, vendor, version, domain, and authorization state.</td>
</tr>
</tbody>
</table>
long getSplashScreenTime()
    Gets the length of time that the application’s splash screen has been
    shown to the user.

boolean isSelectedScreenSaver()
    Checks if the MIDlet is the currently selected screen saver.

void notifyDestroyed()
    Used by an MIDlet to notify the application management software that it
    has entered into the Destroyed state.

void notifyPaused()
    Deprecated. The Paused state is deprecated; the application does not
    need to inform the implementation that it is quiescent.

void pauseApp()
    Deprecated. Previously calls to pauseApp did not provide reliable
    information about system state changes. To be informed of changes in
    resources available to the application, use the listeners in the respective
    APIs. For example, to be notified that the application is not in the
    foreground, register a DisplayListener. The EventManager can be used
    to listen for system state changes.

boolean platformRequest(String URL)
    Requests that the device handle (for example, display or install) the
    indicated URL.

void resumeRequest()
    Deprecated. MIDlets will not be paused and do not need to be resumed.

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.

Since: MIDP 2.0

Methods

checkPermission

public final int checkPermission(String permission)

    Deprecated. The permissions model in this specification does not use named permissions.
The Named Permissions of MIDP 2.x have been deprecated. For MIDP 3.0 applications, calling checkPermission throws an exception. For MIDP 2.x applications, the implementation must implement checkPermission.

Gets the status of the specified legacy named 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

Throws:
IllegalStateException - if the MIDlet suite is a MIDP 3.0 MIDlet suite.

Since: MIDP 2.0

destroyApp

protected abstract void destroyApp(boolean unconditional)
throws javax.microedition.midlet.MIDletStateChangeException

Parameters:
  unconditional - When this method is called, the MIDlet must cleanup and release all resources. The destroyApp method is always called with true to force termination. The use of destroyApp(false) has been deprecated.

Throws:
  MIDletStateChangeException - is thrown if the MIDlet wishes to continue to execute (not enter the Destroyed state). This exception is ignored and the MIDlet is destroyed.

Since: MIDP 1.0

getAppProperty

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 and the manifest. Refer to the MIDlet Suite Installation for the handling of attributes that appear in both application descriptor and manifest.

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.

Since: MIDP 1.0

getAppProperty

public final static java.lang.String[] getAppProperty(String name, String vendor, String attributeName, String attributeDelimiter)

Provides a mechanism to retrieve and at the same time parse application property values from the application descriptor and the manifest of the MIDlet or bound LIBlets. This method retrieves property value for a specific LIBlet or MIDlet, so the same property name is allowed to be used by multiple LIBlets without ambiguity in the same binding.
javax.microedition.midlet.MIDlet

Parameters:
- name - name of the LIBlet or MIDlet to retrieve named property information from. If both name
  and vendor are null, this retrieves information from the MIDlet JAD/Manifest
- vendor - the vendor of the LIBlet or MIDlet
- attributeName - the name of the property to retrieve
- attributeDelimiter - The characters in this parameter are the delimiters for separating the
  retrieved property value into tokens. Delimiter characters themselves will not be treated as
  tokens. If this is null, the value is not tokenized.

Returns:
An array of string with the value of the property tokenized. null is returned if no value is
available for the attributeName

Throws:
NullPointerException - is thrown if attributeName is null.

Since: MIDP 3.0

getMIDletIdentity

public final javax.microedition.midlet.MIDletIdentity getMIDletIdentity()

Gets a MIDletIdentity instance associated with this MIDlet, which can be used to obtain a
MIDlet's name, vendor, version, domain, and authorization state.

Returns:
a MIDletIdentity instance representing the identity of the MIDlet

Since: MIDP 3.0

getSplashScreenTime

public final long getSplashScreenTime()

Gets the length of time that the application's splash screen has been shown to the user.

An application may specify a splash screen in its manifest (see Packaging and the MIDlet package
description for more details regarding splash screens).

If the splash screen is still visible, this method returns the elapsed time since when it was first
shown. Implementations are required to provide a time value that is accurate to within +/-100
milliseconds, as measured by viewing the physical display of the device. A value of -1 is returned if
the splash screen has been hidden or if a valid splash screen image is not specified in the MIDlet's
manifest.

Returns:
the length of time (in milliseconds) that the currently visible splash screen has been shown, or -
1 if no splash screen is currently visible

Since: MIDP 3.0

isSelectedScreenSaver

public final boolean isSelectedScreenSaver()

Checks if the MIDlet is the currently selected screen saver. If the MIDlet is not a screen saver
MIDlet, this method MUST return false. Since the SCREENSAVER_MODE_ACTIVATED system event is
sent to all screen saver MIDlets, a MIDlet can use this method to determine if it is the selected
screen saver.

Returns:
true if the MIDlet is the selected screen saver, false otherwise.

Since: MIDP 3.0
**notifyDestroyed**

```java
public final void notifyDestroyed()
```

Used by a 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.

**Since:** MIDP 1.0

---

**notifyPaused**

```java
public final void notifyPaused()
```

**Deprecated.** The *Paused* state is deprecated; the application does not need to inform the implementation that it is quiescent.

Pausing of MIDlets is deprecated. For backward compatibility and to avoid problems with existing MIDlets, calling `notifyPaused` in the *Active* state MUST cause `startApp` to be called immediately. The MIDlet is expecting a call to `startApp` to restore it to the *Active* state.

Invoking this method MUST NOT have any effect if the MIDlet is not in the *Active* state. A misbehaving MIDlet that continuously calls `notifyPaused` may be destroyed.

**Since:** MIDP 1.0

---

**pauseApp**

```java
protected void pauseApp()
```

**Deprecated.** Previously calls to `pauseApp` did not provide reliable information about system state changes. To be informed of changes in resources available to the application, use the listeners in the respective APIs. For example, to be notified that the application is not in the foreground, register a `DisplayListener`. The `EventManager` can be used to listen for system state changes.

Pausing of MIDlets is deprecated. The application management software MUST NOT call the `pauseApp` method. The MIDlet is not required to implement the `pauseApp` method. Classes in a MIDlet suite may call the `pauseApp` method but the state of the MIDlet MUST NOT be changed.

**Since:** MIDP 1.0

---

**platformRequest**

```java
public final boolean platformRequest(String URL)
    throws javax.microedition.io.ConnectionNotFoundException
```


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), 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). The component of the platform that handles the request is responsible for protecting the user from unexpected financial impacts. For example, some 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**

```java
public final void resumeRequest()
```

Deprecated. **MIDlets will not be paused and do not need to be resumed.**

Pausing of MIDlets is deprecated. Invoking this method MUST have no effect. The application management software maintains the MIDlet in the Active state.

Since: MIDP 1.0

**startApp**

```java
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.

The use of MIDletStateChangeException to delay starting a MIDlet is deprecated. If MIDletStateChangeException exception is thrown it MUST be handled as a RuntimeException below.

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.

**Since:** MIDP 1.0
javax.microedition.midlet

MIDletIdentity

Declaration

public final class MIDletIdentity

Object

\[ \text{---javax.microedition.midlet.MIDletIdentity} \]

Description

Since: MiDP 3.0

Field Summary

<table>
<thead>
<tr>
<th>public static final</th>
<th>IDENTIFIED_THIRD_PARTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant for the Identified Third Party domain.</td>
<td>Value: IdentifiedThirdParty</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>MANUFACTURER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant for the Manufacturer domain.</td>
<td>Value: Manufacturer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>OPERATOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant for the Operator domain.</td>
<td>Value: Operator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public static final</th>
<th>UNIDENTIFIED_THIRD_PARTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant for the Unidentified Third Party domain.</td>
<td>Value: UnidentifiedThirdParty</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getName()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the name of the MIDlet</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getSecurityDomain()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the name of the security domain for the MIDlet associated with the specified MIDletIdentity.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getVendor()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get vendor name of the MIDlet</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getVersion()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get version of the MIDlet</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>boolean</th>
<th>isAuthorized()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checks whether the MIDlet associated with the specified MIDletIdentity is authorized to the current MIDlet's runtime execution environment via application level access authorization.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>toString()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns the string representation of this MIDletIdentity.</td>
<td></td>
</tr>
</tbody>
</table>
javax.microedition.midlet.MIDletIdentity

Methods inherited from class Object

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

Fields

IDENTIFIED_THIRD_PARTY

public static final java.lang.String IDENTIFIED_THIRD_PARTY

Constant for the Identified Third Party domain. Returned by getSecurityDomain() iff the MIDlet was bound to the Identified Third Party domain at install time.
Constant value: IdentifiedThirdParty

MANUFACTURER

public static final java.lang.String MANUFACTURER

Constant for the Manufacturer domain. Returned by getSecurityDomain() if the MIDlet was bound to the Manufacturer domain at install time.
Constant value: Manufacturer

OPERATOR

public static final java.lang.String OPERATOR

Constant for the Operator domain. Returned by getSecurityDomain() iff the MIDlet was bound to the Operator domain at install time.
Constant value: Operator

UNIDENTIFIED_THIRD_PARTY

public static final java.lang.String UNIDENTIFIED_THIRD_PARTY

Constant for the Unidentified Third Party domain. Returned by getSecurityDomain() iff the MIDlet was bound to the Unidentified Third Party domain at install time.
Constant value: UnidentifiedThirdParty

Methods

getName

public java.lang.String getName()

Get the name of the MIDlet

Returns:
Name of the MIDlet

getSecurityDomain

public java.lang.String getSecurityDomain()
javax.microedition.midlet.MIDletIdentity

Get the name of the security domain for the MIDlet associated with the specified MIDletIdentity. If the MIDlet was bound to one of the predefined security domains, then one of the following MUST be returned, as appropriate:

- MANUFACTURER
- OPERATOR
- IDENTIFIED_THIRD_PARTY
- UNIDENTIFIED_THIRD_PARTY

If the MIDlet was not bound to one of the predefined security domains, then some String value other than those listed above MUST be returned.

**Returns:**
Name of the security domain that the MIDlet associated with this MIDletIdentity was bound to when it was installed.

---

**getVendor**

public java.lang.String getVendor()

Get vendor name of the MIDlet

**Returns:**
Vendor name of the MIDlet

---

**getVersion**

public java.lang.String getVersion()

Get version of the MIDlet

**Returns:**
Version string of the MIDlet

---

**isAuthorized**

public boolean isAuthorized()

Checks whether the MIDlet associated with the specified MIDletIdentity is authorized to the current MIDlet's runtime execution environment via application level access authorization.

**Returns:**
true if the MIDlet associated with the specified MIDletIdentity is authorized to the current MIDlet's runtime execution environment, false otherwise.

---

**toString**

public java.lang.String toString()

Returns the string representation of this MIDletIdentity.

**Returns:**
the String concatenation of getVendor() + "; " + getName() + "; " + getVersion().
javax.microedition.midlet

MIDletStateChangeException

Declaration

public class MIDletStateChangeException extends Exception

Object

|-- Throwable
|   |-- Exception
|      |-- javax.microedition.midlet.MIDletStateChangeException

Description

Signals that a requested MIDlet state change failed. This exception is thrown by the MIDlet in response to state change calls into the application via the MIDlet interface.

Since: MIDP 1.0

See Also: MIDlet

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>MIDletStateChangeException()</td>
<td>Deprecated. Constructs an exception with no specified detail message.</td>
</tr>
<tr>
<td>public</td>
<td>MIDletStateChangeException(String s)</td>
<td>Deprecated. Constructs an exception with the specified detail message.</td>
</tr>
</tbody>
</table>

Methods inherited from class Throwable

getMessage, printStackTrace, printStackTrace, toString

Methods inherited from class Object

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

Constructors

MIDletStateChangeException

public MIDletStateChangeException()

Deprecated.

Constructs an exception with no specified detail message.

MIDletStateChangeException

public MIDletStateChangeException(String s)

Deprecated.

Constructs an exception with the specified detail message.
javax.microedition.midlet.MIDletStateChangeException

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.

Unless otherwise noted, passing a null argument to a constructor or method in any class or interface in this package MUST cause a NullPointerException to be thrown.

References
MIDP 3.0 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
- [RFC3280] - Internet X.509 Public Key Infrastructure

16.1 MIDP X.509 Certificate Profile

WAP-211-WAPCert-20010522-a [WAPCert] which is based on [RFC3280] Internet X.509 Public Key Infrastructure Certificate and CRL Profile.

Devices MUST conform to all mandatory requirements in [WAPCert]. Devices SHOULD conform to all optional requirements in [WAPCert]. Mandatory and optional requirements are listed in Appendix C of [WAPCert]. Devices SHOULD NOT conform to the requirements in the following sections:

- [WAPCert] § 6.2, User Certificates for Authentication
- [WAPCert] § 6.3, User Certificates for Digital Signatures

[RFC3280] 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 [RFC3280] Section 6.2 Extending Path Validation. Support for Policy Certificate Authority or policy attributes is not required.
16.1.1 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 [RFC3280] § 4.2.1.3), basic constraints (see [RFC3280] § 4.2.1.10).

Although a conforming device may not recognize the authority and subject key identifier (see [RFC3280] § 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.

16.1.2 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.

16.1.3 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.

16.1.4 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 [RFC3280] § 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

<table>
<thead>
<tr>
<th>Interfaces</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificate</td>
<td>Interface common to certificates.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exceptions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CertificateException</td>
<td>The CertificateException encapsulates an error that occurred while a Certificate is being used.</td>
</tr>
</tbody>
</table>
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

An X.509 distinguished name is a sequence of relative names. Each relative name is a sequence of attributes; each attribute is a sequence of an object ID and a value. For string comparison purposes, the following rules define a strict printable representation :

1. There is no added white space around separators.
2. The attributes are in the same order as in the certificate; attributes are not reordered.
3. If an object ID is in the table below, the label from the table will be substituted for the object ID, else the ID is formatted as a string using the binary printable representation above.
4. Each object ID or label and value within an attribute will be separated by a "=" (Unicode U+003D), even if the value is empty.
5. If value is not a string, then it is formatted as a string using the binary printable representation above.
6. Attributes will be separated by a ";" (Unicode U+003B)
7. Relative names will be separated by a "+" (Unicode U+002B)

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
javax.microedition.pki.Certificate

<table>
<thead>
<tr>
<th>Method</th>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getIssuer()</td>
<td>java.lang.String</td>
<td>Gets the name of this certificate’s issuer.</td>
</tr>
<tr>
<td>getNotAfter()</td>
<td>long</td>
<td>Gets the time after which this Certificate may not be used from the validity period.</td>
</tr>
<tr>
<td>getNotBefore()</td>
<td>long</td>
<td>Gets the time before which this Certificate may not be used from the validity period.</td>
</tr>
<tr>
<td>getSerialNumber()</td>
<td>java.lang.String</td>
<td>Gets the printable form of the serial number of this Certificate.</td>
</tr>
<tr>
<td>getSigAlgName()</td>
<td>java.lang.String</td>
<td>Gets the name of the algorithm used to sign the Certificate.</td>
</tr>
<tr>
<td>getSubject()</td>
<td>java.lang.String</td>
<td>Gets the name of this certificate’s subject.</td>
</tr>
<tr>
<td>getType()</td>
<td>java.lang.String</td>
<td>Get the type of the Certificate.</td>
</tr>
<tr>
<td>getVersion()</td>
<td>java.lang.String</td>
<td>Gets the version number of this Certificate.</td>
</tr>
</tbody>
</table>

## Methods

**getIssuer**

```java
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**

```java
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**

```java
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 [RFC3279] § 2.2.

**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 [RFC3280] it would be "3".

**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>Public Static Final</th>
<th>BAD_EXTENSIONS</th>
<th>Indicates a certificate has unrecognized critical extensions. Value: 1</th>
</tr>
</thead>
<tbody>
<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>Method</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>public static final ROOT_CA_EXPIRED</td>
<td>Indicates the root CA's public key is expired. Value: 12</td>
<td></td>
</tr>
<tr>
<td>public static final SITENAME_MISMATCH</td>
<td>Indicates a certificate does not contain the correct site name. Value: 7</td>
<td></td>
</tr>
<tr>
<td>public static final UNAUTHORIZED_INTERMEDIATE_CA</td>
<td>Indicates an intermediate certificate in the chain does not have the authority to be a intermediate CA. Value: 4</td>
<td></td>
</tr>
<tr>
<td>public static final UNRECOGNIZED_ISSUER</td>
<td>Indicates a certificate was issued by an unrecognized entity. Value: 8</td>
<td></td>
</tr>
<tr>
<td>public static final UNSUPPORTED_PUBLIC_KEY_TYPE</td>
<td>Indicates that type of the public key in a certificate is not supported by the device. Value: 13</td>
<td></td>
</tr>
<tr>
<td>public static final UNSUPPORTED_SIGALG</td>
<td>Indicates a certificate was signed using an unsupported algorithm. Value: 9</td>
<td></td>
</tr>
<tr>
<td>public static final VERIFICATION_FAILED</td>
<td>Indicates a certificate failed verification. Value: 14</td>
<td></td>
</tr>
</tbody>
</table>

**Constructor Summary**

```java
public CertificateException(Certificate certificate, byte status)
    Create a new exception with a Certificate and specific error reason.
```

```java
public CertificateException(String message, Certificate certificate, byte status)
    Create a new exception with a message, Certificate, and specific error reason.
```

**Method Summary**

```java
javax.microedition.pki.Certificate getCertificate()
    Get the Certificate that caused the exception.
```

```java
byte getReason()
    Get the reason code.
```

Methods inherited from class Throwable

getMessage, printStackTrace, printStackTrace, toString

Methods inherited from class Object

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

BAD_EXTENSIONS
public static final byte BAD_EXTENSIONS

Indicates a certificate has unrecognized critical extensions. The value is 1.
Constant value: 1

BROKEN_CHAIN
public static final byte BROKEN_CHAIN

Indicates a certificate in a chain was not issued by the next authority in the chain. The value is 11.
Constant value: 11

CERTIFICATE_CHAIN_TOO_LONG
public static final byte CERTIFICATE_CHAIN_TOO_LONG

Indicates the server certificate chain exceeds the length allowed by an issuer's policy. The value is 2.
Constant value: 2

EXPIRED
public static final byte EXPIRED

Indicates a certificate is expired. The value is 3.
Constant value: 3

INAPPROPRIATE_KEY_USAGE
public static final byte INAPPROPRIATE_KEY_USAGE

Indicates a certificate public key has been used in way deemed inappropriate by the issuer. The value is 10.
Constant value: 10

MISSING_SIGNATURE
public static final byte MISSING_SIGNATURE

Indicates a certificate object does not contain a signature. The value is 5.
Constant value: 5

NOT_YET_VALID
public static final byte NOT_YET_VALID

Indicates a certificate is not yet valid. The value is 6.
Constant value: 6

ROOT_CA_EXPIRED
public static final byte ROOT_CA_EXPIRED

Indicates the root CA's public key is expired. The value is 12.
Constant value: 12
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.
javax.microedition.pki.CertificateException

public class CertificateException

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

Get the Certificate that caused the exception.

getReason

Get the reason code.
Chapter 17

Package
javax.microedition.rms

Description
The Mobile Information Device Profile provides a mechanism for MIDlets to persistently store data and later retrieve it.

Unless otherwise noted, passing a null argument to a constructor or method in any class or interface in this package MUST cause a NullPointerException to be thrown.

17.1 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.

17.1.1 Record Store

A record store consists of a collection of records that will remain persistent across multiple invocations of a MIDlet. The implementation is responsible for making its best effort to maintain the integrity of the MIDlet's record stores throughout the normal use of an implementation, including device reboots, power loss, etc. The actual process of persisting record store data is the responsibility of the implementation, and MAY occur asynchronously, even as part of a cleanup process when the device restarts.

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. LIBlet-owned record stores (i.e. those provisioned via LIBlet-Persistent-Data-URL-<n>) will have a namespace based on the owning LIBlet. Note that multiple versions of the same LIBlet may exist on a device, and each of these LIBlet versions will have its own record store namespace. When a MIDlet suite is deleted from a platform, all record stores associated with its MIDlets MUST be deleted. When a LIBlet is deleted, all of its associated record stores MUST also be deleted. MIDlets within a MIDlet suite can access one another's record stores directly. The RecordStore APIs allow for the explicit sharing of record stores if the MIDlet creating the RecordStore chooses to give such permission.

17.1.1.1 Naming Record Stores

A MIDlet Suite's record stores are uniquely named using the unique name of the MIDlet suite plus the name of the record store. MIDlet suites are identified by the MIDlet-Vendor and MIDlet-Name attributes from the application descriptor.

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.

17.1.1.2 Provisioned Record Stores

Record stores MAY be created by the AMS during MIDlet suite or LIBlet installation. Record stores
associated with the MIDlet Suite or LIBlet are created if the MIDlet-Persistent-Data-URL-<n>
or LIBlet-Persistent-Data-URL-<n> attribute is present in their respective manifests. See
RMS Data Provisioning for more details.

For MIDlet Suites, the record store created may be private or shared based on the value of the
authmode field within the RMS data file pointed to by the MIDlet-Persistent-Data-URL-<n>
attribute. The value of the authmode field may be one of :

- AUTHMODE_ANY to allow access to any MIDlet suite,
- AUTHMODE_APPLEVEL to allow access only to certain identified MIDlet suites, or
- AUTHMODE_PRIVATE to allow access only to the current MIDlet suite

For LIBlets, using the LIBlet-Persistent-Data-URL-<n> attribute in the LIBlet manifest
is the only way to create a record store that is owned by the LIBlet. A record store owned by
a LIBlet MUST have its authmode set to AUTHMODE_PRIVATE. If the authmode field within
the RMS data file for the LIBlet is set to any value other than AUTHMODE_PRIVATE then the
installation of the LIBlet MUST fail. Record stores created dynamically at runtime by LIBlet code
are owned by the MIDlet suite of the execution environment in which the MIDlet is running. A
MIDlet MAY open a RecordStore owned by a LIBlet it declares a dependency on by passing in
the RecordStore name, LIBlet vendor, and LIBlet name to the RecordStore.openRecordStore
(plaintext) or RecordStore.openRecordStore (encrypted) method call. A MIDlet will not be able to
access a LIBlet's record stores if its MIDlet Suite did not declare a dependency on that LIBlet.

It is possible for a provisioned RMS data file to contain no records; that is, the record store may be
an empty record store and may be populated by a MIDlet at runtime. Provisioned record stores are
available for use immediately upon creation.

17.1.1.3 Shared Record Stores

Record store 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.

A MIDlet MAY control access to a shared record store using AUTHMODE_APPLEVEL and the
application level access control mechanism as described in Application Level Access Authorization.
If the MIDlet JAD/Manifest does not contain any of the access authorization attributes, the
AUTHMODE_APPLEVEL has no effect, and the authmode defaults to AUTHMODE_ANY.

Access controls are defined when record stores to be shared are created. Access controls are
enforced when record stores are opened. A MIDlet suite defines access control by using access
modes. The access modes allow private use or shareable with any other MIDlet suite. Shareable
record stores of two kinds can be created :

- All the other MIDlet suites have the same level of access privilege.
- The owning MIDlet suite identifies a set of MIDlet suites with which to share the record store
  using the application level access control.

Implementations MUST allow shared record stores to be opened concurrently by multiple
applications. Successful updates to records MUST be visible to all applications when the update is
complete. All RecordListeners to shared record stores must be notified after a record changes,
regardless of the MIDlet that registered the listener and regardless of which MIDlet made the record update, both within and across MIDlets.

17.1.1.4 Secure Record Stores

An application may optionally request that a RecordStore's record data be encrypted on the device. If requested, the implementation MUST encrypt the records before they are persisted and automatically decrypt them when they are fetched. Implementations MUST encrypt secure record store data using either a hardware or software based cryptographically strong algorithm; an example is a symmetric-key cipher such as AES, DES, or Blowfish. The encryption key MUST be derived from the password supplied. Encrypted record stores are only as secure as the handling of the key; if a MIDlet stores the password within its code, security is not a reasonable expectation. For improved security, the MIDlet should ask the user for the password on each invocation of the MIDlet. The password is a String that consists of Unicode characters with a recommended minimum length of eight characters.

Note: In RMS Interchange file format, the encryption algorithm and standards for key derivation are specified to ensure interoperability between devices, whereas for on-device encryption of record stores, the implementation may choose to use the same encryption standards as for the RMS interchange format or follow the guidelines mentioned above.

17.1.1.5 Atomicity of RecordStore Access

No locking operations are provided in this API. Record store implementations MUST ensure that all individual record store operations are atomic, synchronous, and serialized so that no corruption occurs with multiple accesses, from within or across execution environments. 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 RecordStore 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 an implementation performs transparent synchronization of a record store or other access, it is the implementation's responsibility to enforce exclusive access to the record store between the MIDlets and synchronization engine. The implementation MUST NOT serialize calls to RecordListeners across execution environments. The implementation MUST call the RecordListener callbacks in the order in which additions, deletions, or changes took place on a record. Implementations MAY coalesce record listener callbacks that resulted from multiple changes to a particular record. Implementations MUST NOT discard any record listener callbacks that resulted from record additions, deletions, or changes.

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.

17.1.1.6 Examples

The static method RecordStore.openRecordStore is overloaded to enable applications to open and create different kinds of RecordStores. Some code examples are given below.

- Private RecordStore:
  ```java
  RecordStore rstore = RecordStore.openRecordStore ( "MyRecordStore", true); // create one if not found
  ```

- Public RecordStore:

  ```java
  RecordStore rstore = RecordStore.openRecordStore
  ```
javax.microedition.rms - Record Management System for MIDP

("MyRecordStore", false); // open only an existing RecordStore.

- **Shared RecordStore**:
  ```java
  int authmode = AUTHMODE_ANY; // or AUTHMODE_APPLEVEL
  boolean writable = true; // or false for read-only
  RecordStore rstore = RecordStore.openRecordStore("MyRecordStore", true, AUTHMODE_ANY, writable);
  ```

- **Private Encrypted RecordStore**:
  ```java
  String password = getPasswordFromUser();
  boolean writable = true; // or false for read-only
  RecordStore rstore = RecordStore.openRecordStore("MyRecordStore", true, AUTHMODE_PRIVATE, writable, password);
  ```

- **Shared Encrypted RecordStore**:
  ```java
  String password = getPasswordFromUser();
  boolean writable = true; // or false for read-only
  RecordStore rstore = RecordStore.openRecordStore("MyRecordStore", true, AUTHMODE_APPLEVEL, writable, password);
  ```

- **Shared RecordStore owned by another MIDlet Suite or a LIBlet**:
  ```java
  RecordStore rstore = RecordStore.openRecordStore("OtherRecordStore", "OtherVendor", "OtherSuite");
  ```

- **Encrypted Shared RecordStore owned by another MIDlet Suite or a LIBlet**:
  ```java
  String password = getPasswordFromUser();
  RecordStore rstore = RecordStore.openRecordStore("OtherRecordStore", "OtherVendor", "OtherSuite", password);
  ```

### 17.1.2 Record Tags

In MIDP 2.0 there was no efficient way to limit the enumeration on a subset of records in a record store. The RecordComparator and RecordFilter are applied on all the records of the record store. For a larger record store, finding a particular record results in call backs on the RecordComparator and RecordFilter for all the records in the store, which is a lot of overhead. The record tags provide an option to the developer to reduce this overhead significantly.

Record tags allow MIDlet developers to associate an integer tag with each record. These tags are specified while calling `addRecord` or `setRecord` to the record store. The developer can now specify these tags when calling enumeration, and the implementation MUST only return those records for which the tags match.
As an example, if a record store has 100 records and the developer tags 10 records with the TAG value of 10. The developer can now call enumeration with tag value 10 and the implementation will only return those records with the tag value of 10. The developer has significantly reduced the number of records that need to be matched or compared.

When records are added with the legacy `addRecord` and `setRecord` API's, the default value of tag MUST be 0.

Tags are not required to be encrypted by the implementation when a record store is locally encrypted. Since record tags may not be encrypted before being written to persistent storage, MIDlet developers should avoid storing sensitive information in clear text in record tags.

### 17.1.3 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.

### 17.1.4 Persistent Data Interchange

Versions of the MIDP specification previous to MIDP 3.0 did not provide either for the provisioning or interchange of RMS record stores. This lack of a common RMS format resulted in limited application portability.

MIDP 3.0 introduces support for a standalone secure binary file format that can be provisioned to a device by URL reference from MIDlet suite's JAD or manifest file.

The `RecordStore` class supports serialization/deserialization of RMS data into this file format with optional encryption.

#### 17.1.4.1 RMS Interchange File Format

The RMS file format is used for persistent data provisioning as well as RMS interchange. Each RMS file MUST contain exactly one serialized record store.

RMS data MAY be provisioned along with the application via standalone files and/or files embedded in the application JAR. RMS files are listed in the `MIDlet-Persistent-Data-URL-<n>` attribute. See RMS Data Provisioning for details.

The recommended RMS file extension is ".rms" and a MIME type is "application/vnd.jcp.javame.midlet-rms".

The RMS data is stored on the device in implementation specific format. In order for RMS data to be exchanged between implementations, it MUST be serialized into this implementation independent format.

The following crypto algorithm, mode, padding scheme, message digest algorithms, and password based key derivation function MUST be supported:

- **Algorithms:** AES 128-bit
- **Algorithm Mode:** CBC
- **Padding Scheme:** PKCS5Padding
javax.microedition.rms - Record Management System for MIDP

- Message Digest Algorithm: SHA-1
- Password Based Key Derivation: PBKDF2 (part of PKCS#5 v2.0)

Cipher used for encryption is represented by a transformation string in the form of "algorithm" or "algorithm\(\text{mode/\text{padding}}\)" (e.g. "AES/CBC/PKCS5Padding" is required to be supported). Note: this format is the same as defined by the SATSA specification for the cipher transformation string. The standard names for algorithm, mode, padding scheme and message digest algorithm are defined in JCE specification.

The following tables define the format of a serialized RMS data file used for interchange of the RMS data. Both encrypted and unencrypted formats are supported. The file header contains information on whether the file is encrypted or not. For an encrypted file, the name of the Cipher Algorithm is placed in the Encryption Parameters portion of the file and contains the name of the Cipher Algorithm used. The AMS MAY use this information to discover the required decryption algorithm.

<table>
<thead>
<tr>
<th>Table 18-1 : RMS Interchange File Format</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Header</strong></td>
</tr>
<tr>
<td>EncryptionParameters</td>
</tr>
<tr>
<td>RecordStoreData</td>
</tr>
<tr>
<td>MessageDigestData</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Header</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FileIdentifier</td>
<td>6 bytes (0x4d, 0x49, 0x44, 0x52, 0x4d, 0x53 or “MIDRMS”)</td>
</tr>
<tr>
<td>VersionNumber</td>
<td>2 bytes First byte: major version Second byte: minor version</td>
</tr>
<tr>
<td>Encrypted</td>
<td>1 byte (boolean) 0: plain-text (not encrypted) file 1: encrypted file.</td>
</tr>
<tr>
<td>MessageDigestAlgorithm</td>
<td>UTF String as specified by DataOutputStream.writeUTF MessageDigestAlgorithmLength: 2 bytes MessageDigestAlgorithmString: variable size of MessageDigestAlgorithmLength bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>EncryptionParameters</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EncryptionAlgorithm</td>
<td>UTF String as specified by DataOutputStream.writeUTF EncryptionAlgorithmLength: 2 bytes</td>
</tr>
<tr>
<td>Cipher algorithm used in this file. Supported values are 1. “algorithm” 2. “algorithm(\text{mode/\text{padding}})” e.g. “AES/CBC/PKCS5Padding”</td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EncryptionAlgorithmString</td>
<td>variable size of NameLength bytes</td>
</tr>
<tr>
<td>IVLength</td>
<td>4 bytes, high byte first as specified by DataOutputStream.writeInt()</td>
</tr>
<tr>
<td></td>
<td>Cipher initialization vector length (if required by the cipher) in bytes. 0:</td>
</tr>
<tr>
<td></td>
<td>if initialization vector is not required by this algorithm</td>
</tr>
<tr>
<td>IV</td>
<td>Variable size of IVLength bytes</td>
</tr>
<tr>
<td></td>
<td>Cipher initialization vector as byte array. IV[0], ... IV[&lt;IVLength&gt;-1]</td>
</tr>
<tr>
<td>SaltLength</td>
<td>4 bytes, high byte first as specified by DataOutputStream.writeInt()</td>
</tr>
<tr>
<td></td>
<td>Password salt length in bytes</td>
</tr>
<tr>
<td>Salt</td>
<td>Variable size of SaltLength bytes</td>
</tr>
<tr>
<td></td>
<td>Password salt as byte array. Salt[0], ... Salt[&lt;SaltLength&gt;-1]</td>
</tr>
<tr>
<td>IterationCount</td>
<td>4 bytes, high byte first as specified by DataOutputStream.writeInt()</td>
</tr>
<tr>
<td></td>
<td>Password iteration count. e.g. 1000</td>
</tr>
<tr>
<td>KeyLength</td>
<td>4 bytes, high byte first as specified by DataOutputStream.writeInt()</td>
</tr>
<tr>
<td></td>
<td>Cipher key length. e.g. 128 bits</td>
</tr>
</tbody>
</table>

### RecordStoreData

<table>
<thead>
<tr>
<th>RecordStore</th>
<th>Record Store Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record 1</td>
<td>First record</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Record N</td>
<td>Last record</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RecordStore</th>
<th>UTF String as specified by DataOutputStream.writeUTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Record store name</td>
</tr>
<tr>
<td>NameLength</td>
<td>2 bytes Name String: variable size of NameLength bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LastModified</th>
<th>8 bytes, high byte first as specified by DataOutputStream.writeLong()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time of last modification As specified by System.currentTimeMillis()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version</th>
<th>4 bytes, high byte first as specified by DataOutputStream.writeInt()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Record store version</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AuthMode</th>
<th>4 bytes, high byte first as specified by DataOutputStream.writeInt()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Authorization mode As specified by RecordStore.setMode(int, int)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Writeable</th>
<th>1 byte (boolean) 0: not writeable by other MIDlet suites (false) 1:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>writeable by other MIDlet suites (true)</td>
</tr>
<tr>
<td></td>
<td>Write access</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>NumberOfRecords</strong></td>
<td>4 bytes, high byte first as specified by DataOutputStream.writeInt()</td>
</tr>
<tr>
<td></td>
<td><strong>Number of records in this record store</strong></td>
</tr>
<tr>
<td><strong>Record</strong></td>
<td></td>
</tr>
<tr>
<td><strong>RecordID</strong></td>
<td>4 bytes, high byte first as specified by DataOutputStream.writeInt()</td>
</tr>
<tr>
<td></td>
<td><strong>ID of this Record</strong></td>
</tr>
<tr>
<td><strong>Tag</strong></td>
<td>4 bytes, high byte first as specified by DataOutputStream.writeInt()</td>
</tr>
<tr>
<td></td>
<td><strong>Tag of this record 0</strong>: if this record has no tag</td>
</tr>
<tr>
<td><strong>RecordDataSize</strong></td>
<td>4 bytes, high byte first as specified by DataOutputStream.writeInt()</td>
</tr>
<tr>
<td></td>
<td><strong>Record size</strong></td>
</tr>
<tr>
<td><strong>RecordData</strong></td>
<td>Variable size of <strong>RecordDataSize</strong> bytes</td>
</tr>
<tr>
<td></td>
<td><strong>Record data byte array RecordData[0], ... RecordData[&lt;RecordDataSize&gt;-1]</strong></td>
</tr>
<tr>
<td><strong>MessageDigest</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MessageDigestLength</strong></td>
<td>4 bytes, high byte first as specified by DataOutputStream.writeInt()</td>
</tr>
<tr>
<td></td>
<td><strong>Message digest length in bytes</strong></td>
</tr>
<tr>
<td><strong>MessageDigest</strong></td>
<td>Variable size of <strong>MessageDigestLength</strong> bytes</td>
</tr>
<tr>
<td></td>
<td><strong>Message digest of the stream containing Encryption parameters and non-encrypted RecordStoreData as byte array: MessageDigest[0], ... MessageDigest[&lt;MessageDigestLength&gt;-1]</strong></td>
</tr>
</tbody>
</table>

Note: For Encrypted RMS Interchange Format files, the entire portion of the stream from RecordStore Data to the end of the Message Digest is encrypted.

### 17.1.5 RMS Example

The following 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;

/**
 * A class used for storing and showing game scores.
 */
public class RMSGameScores
    implements RecordFilter, RecordComparator {
```
javax.microedition.rms - Record Management System for MIDP

/*
 * The RecordStore used for storing the game scores.
 */
private RecordStore recordStore = null;

/*
 * The player name to use when filtering.
 */
public static String playerNameFilter = null;

/*
 * Part of the RecordFilter interface.
 */
public boolean matches(byte[] candidate)
throws IllegalArgumentException
{
    // If no filter set, nothing can match it.
    if (this.playerNameFilter == null) {
        return false;
    }

    ByteArrayInputStream bais = new ByteArrayInputStream(candidate);
    DataInputStream inputStream = new DataInputStream(bais);
    String name = null;
    try {
        int score = inputStream.readInt();
        name = inputStream.readUTF();
    }
    catch (EOFException eoe) {
        System.out.println(eoe);
        eoe.printStackTrace();
    }
    catch (IOException eoe) {
        System.out.println(eoe);
        eoe.printStackTrace();
    }
    return (this.playerNameFilter.equals(name));
}

/*
 * Part of the RecordComparator interface.
 */
public int compare(byte[] rec1, byte[] rec2)
{
    // Construct DataInputStreams for extracting the scores from
    // the records.
    ByteArrayInputStream bais1 = new ByteArrayInputStream(rec1);
    DataInputStream inputStream1 = new DataInputStream(bais1);
    ByteArrayInputStream bais2 = new ByteArrayInputStream(rec2);
    DataInputStream inputStream2 = new DataInputStream(bais2);
    int score1 = 0;
    int score2 = 0;
    try {
        // Extract the scores.
javax.microedition.rms - Record Management System for MIDP

score1 = inputStream1.readInt();
score2 = inputStream2.readInt();
}
catch (EOFException eoe) {
    System.out.println(eoe);
    eoe.printStackTrace();
}
catch (IOException eoe) {
    System.out.println(eoe);
    eoe.printStackTrace();
}

// Sort by score
if (score1 < score2) {
    return RecordComparator.PRECEDES;
} else if (score1 > score2) {
    return RecordComparator.FOLLOWS;
} else {
    return RecordComparator.EQUIVALENT;
}

/**
 * The constructor opens the underlying record store,
 * creating it if necessary.
 */
public RMSGameScores() {
    //
    // Create a new record store for this example
    //
    try {
        recordStore = RecordStore.openRecordStore("scores", true);
    }
    catch (RecordStoreException rse) {
        System.out.println(rse);
        rse.printStackTrace();
    }
}

/**
 * Add a new score to the storage.
 *
 * @param score the score to store.
 * @param playerName the name of the player achieving this score.
 */
public void addScore(int score, String playerName) {
    //
    // Each score is stored in a separate record, formatted with
    // the score, followed by the player name.
    //
    int recId; // returned by addRecord but not used
ByteArrayOutputStream baos = new ByteArrayOutputStream();
DataOutputStream outputStream = new DataOutputStream(baos);
try {
    // Push the score into a byte array.
    outputStream.writeInt(score);
    // Then push the player name.
    outputStream.writeUTF(playerName);
} catch (IOException ioe) {
    System.out.println(ioe);
    ioe.printStackTrace();
}

// Extract the byte array
byte[] b = baos.toByteArray();
// Add it to the record store
try {
    recId = recordStore.addRecord(b, 0, b.length);
} catch (RecordStoreException rse) {
    System.out.println(rse);
    rse.printStackTrace();
}

/**
 * A helper method for the printScores methods.
 */
private void printScoresHelper(RecordEnumeration re) {
    try {
        while (re.hasNextElement()) {
            int id = re.nextRecordId();
            ByteArrayInputStream bais = new ByteArrayInputStream(recordStore.getRecord(id));
            DataInputStream inputStream = new DataInputStream(bais);
            try {
                int score = inputStream.readInt();
                String playerName = inputStream.readUTF();
                System.out.println(playerName + " = " + score);
            } catch (EOFException eofe) {
                System.out.println(eofe);
                eofe.printStackTrace();
            }
        }
    }
    catch (RecordStoreException rse) {
        System.out.println(rse);
        rse.printStackTrace();
    }
    catch (IOException ioe) {
        System.out.println(ioe);
        ioe.printStackTrace();
    }
}
javax.microedition.rms - Record Management System for MIDP

/**
 * 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");
}
javax.microedition.rms - Record Management System for MIDP

Since: MIDP 1.0
# Class Summary

## Interfaces
- **RecordComparator**: An interface defining a comparator which compares two records (in an implementation-defined manner) to see if they match or what their relative sort order is.
- **RecordEnumeration**: An interface representing a bidirectional record store Record enumerator.
- **RecordFilter**: An interface defining a filter which examines a record to see if it matches (based on an application-defined criteria).
- **RecordListener**: A listener interface for receiving Record Changed/Added/Deleted events from a record store.

## Classes
- **RecordStore**: A class representing a record store.
- **RecordStoreInfo**: A class representing information about a RecordStore, including authorization mode, encryption status, writeable status, and size information.

## Exceptions
- **InvalidRecordIDException**: Thrown to indicate an operation could not be completed because the record ID was invalid.
- **RecordStoreException**: Thrown to indicate a general exception occurred in a record store operation.
- **RecordStoreFullException**: Thrown to indicate an operation could not be completed because the record store system storage is full.
- **RecordStoreNotFoundException**: Thrown to indicate an operation could not be completed because the record store could not be found.
- **RecordStoreNotOpenException**: Thrown to indicate that an operation was attempted on a closed record store.
- **SecureRecordStoreException**: Thrown to indicate that a problem occurred during the process of Encrypting or Decrypting data of a Secure RecordStore.
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 Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>InvalidRecordIDException(String message)</td>
</tr>
<tr>
<td>Constructs a new InvalidRecordIDException with the specified detail message.</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>InvalidRecordIDException()</td>
</tr>
<tr>
<td>Constructs a new InvalidRecordIDException with no detail message.</td>
<td></td>
</tr>
</tbody>
</table>

Methods inherited from class Throwable

getMessage, printStackTrace, 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>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>EQUIVALENT</code></td>
<td>EQUIVALENT means that in terms of search or sort order, the two records are the same. Value: 0</td>
</tr>
<tr>
<td><code>FOLLOWS</code></td>
<td>FOLLOWS means that the left (first parameter) record follows the right (second parameter) record in terms of search or sort order. Value: 1</td>
</tr>
<tr>
<td><code>PRECEDES</code></td>
<td>PRECEDES means that the left (first parameter) record precedes the right (second parameter) record in terms of search or sort order. 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>int getRecordId(int index)</td>
<td>Used to quickly find a record present at the specified index of the record enumeration.</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>
</tbody>
</table>
javax.microedition.rms.RecordEnumeration

**Methods**

**destroy**

```java
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.

**getRecordId**

```java
public int getRecordId(int index)
    throws java.lang.IllegalArgumentException
```

Used to quickly find a record present at the specified index of the record enumeration. It is recommended to use `keepUpdated = false` with this method. The behavior when `keepUpdated = true` is used is implementation dependent. Calling this method doesn't change the current iteration index of the enumeration.

**Parameters:**
index - The value of index is 0 to numRecords()-1.

Returns:
The recordId of the record present at the given index of the record enumeration.

Throws:
IllegalArgumentException - If the value of index is less than 0 or greater than numRecords()-1.

Since: MIDP 3.0

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: rebuild()

nextRecord
public byte[] nextRecord()

throws javax.microedition.rms.InvalidRecordIDException, javax.microedition.rms.RecordStoreNotOpenException, javax.microedition.rms.RecordStoreException
javax.microedition.rms.RecordEnumeration

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**

```java
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**

```java
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**

```java
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**

```java
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**

```java
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. Rebuild updates the existing enumeration and it doesn't create a new enumeration. This method is useful when keepUpdated is false.

**See Also:** keepUpdated(boolean)

---

**reset**

```java
public void reset()
```

Resets the iteration index back to 0, it does not rollback the changes if records were deleted.
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>Method</th>
<th>Return Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>matches</td>
<td>boolean</td>
<td>byte[] candidate</td>
<td>Returns true if the candidate matches the implemented criterion.</td>
</tr>
</tbody>
</table>

Methods

matches

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>void recordAdded(RecordStore recordStore, int recordId)</td>
<td>Called after a record has been added to a record store.</td>
</tr>
<tr>
<td>void recordChanged(RecordStore recordStore, int recordId)</td>
<td>Called after a record in a record store has been changed.</td>
</tr>
<tr>
<td>void 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 after 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. MIDlets can use this API to persistently store data and later retrieve it. A record store consists of a collection of records that will remain persistent across multiple invocations of a MIDlet. For further information and examples of usage, see the Persistent Storage section of the package summary.
Since: MIDP 1.0

Field Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final AUTHMODE_ANY</td>
<td>Authorization to allow access to any MIDlet suites. Value: 1</td>
</tr>
<tr>
<td>public static final AUTHMODE_APPLEVEL</td>
<td>Authorization to allow access to only certain identified MIDlets. Value: 2</td>
</tr>
<tr>
<td>public static final 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, and returns the recordId for this new record.</td>
</tr>
<tr>
<td>int addRecord(byte[] data, int offset, int numBytes, int tag)</td>
<td>Adds a new record to the record store and associates the record with the given tag.</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; the recordId for this record MUST NOT be reused.</td>
</tr>
<tr>
<td>static void deleteRecordStore(String recordStoreName)</td>
<td>Deletes the named record store.</td>
</tr>
<tr>
<td>Class/Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>javax.microedition.rms.RecordStore.enumerateRecords</code></td>
<td>Returns an enumeration for traversing a set of records in the record store in an optionally specified order.</td>
</tr>
<tr>
<td><code>javax.microedition.rms.RecordStore.enumerateRecords</code></td>
<td>Returns an enumeration for traversing a set of records in the record store in an optionally specified order.</td>
</tr>
<tr>
<td><code>static void exportRecordStore(OutputStream os, String recordStoreName, String internalPassword, String exportPassword)</code></td>
<td>Exports the record store to an RMS stream (see RMS Interchange Format).</td>
</tr>
<tr>
<td><code>long getLastModified()</code></td>
<td>Returns the last time the record store was modified, in the format used by System.currentTimeMillis().</td>
</tr>
<tr>
<td><code>java.lang.String getName()</code></td>
<td>Returns the name of this RecordStore.</td>
</tr>
<tr>
<td><code>int getNextRecordID()</code></td>
<td>Returns the recordId of the next record to be added to the record store.</td>
</tr>
<tr>
<td><code>int getNumRecords()</code></td>
<td>Returns the number of records currently in the record store.</td>
</tr>
<tr>
<td><code>byte[] getRecord(int recordId)</code></td>
<td>Returns a copy of the data stored in the given record.</td>
</tr>
<tr>
<td><code>int getRecord(int recordId, byte[] buffer, int offset)</code></td>
<td>Returns the data stored in the given record.</td>
</tr>
<tr>
<td><code>int getRecordSize(int recordId)</code></td>
<td>Returns the size (in bytes) of the MIDlet data available in the given record.</td>
</tr>
<tr>
<td><code>javax.microedition.rms.RecordStoreInfo getRecordStoreInfo()</code></td>
<td>Gets a RecordStoreInfo instance for this RecordStore.</td>
</tr>
<tr>
<td><code>int getSize()</code></td>
<td>Deprecated. Previously calls to getSize() could not accommodate a RecordStore larger than Integer.MAX_VALUE. To query the size of a RecordStore, use getRecordStoreInfo() and RecordStoreInfo.getSize(), which returns a long and can accommodate RecordStore sizes up to Long.MAX_VALUE.</td>
</tr>
<tr>
<td><code>int getSizeAvailable()</code></td>
<td>Deprecated. Previously calls to getSizeAvailable() could not accommodate a RecordStore larger than Integer.MAX_VALUE. To query the available size of a RecordStore, use getRecordStoreInfo() and RecordStoreInfo.getSizeAvailable(), which returns a long and can accommodate RecordStore sizes up to Long.MAX_VALUE.</td>
</tr>
<tr>
<td><code>int getTag(int recordId)</code></td>
<td>Returns the tag associated with the recordId.</td>
</tr>
<tr>
<td><code>int getVersion()</code></td>
<td>Each time a record store is modified (by addRecord, setRecord, or deleteRecord methods) its version is incremented.</td>
</tr>
</tbody>
</table>
### javax.microedition.rms.RecordStore

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>static</strong> importRecordStore(InputStream is, String importPassword, String internalPassword)</td>
<td>Imports the record store from an encrypted or plaintext RMS stream (see <a href="#">RMS Interchange Format</a>).</td>
</tr>
<tr>
<td><strong>static</strong> listRecordStores()</td>
<td>Returns an array of the names of record stores owned by the MIDlet suite.</td>
</tr>
<tr>
<td><strong>static</strong> openRecordStore(String recordStoreName, boolean createIfNecessary)</td>
<td>Open (and possibly create) a record store associated with the current MIDlet suite.</td>
</tr>
<tr>
<td><strong>static</strong> openRecordStore(String recordStoreName, boolean createIfNecessary, int authmode, boolean writeable)</td>
<td>Open (and possibly create) a record store that can be shared with other MIDlet suites.</td>
</tr>
<tr>
<td><strong>static</strong> openRecordStore(String recordStoreName, boolean createIfNecessary, int authmode, boolean writeable, String password)</td>
<td>Open (and possibly create) a record store that can be shared with other MIDlet suites.</td>
</tr>
<tr>
<td><strong>static</strong> openRecordStore(String recordStoreName, String vendorName, String suiteName)</td>
<td>Open a record store associated with the named MIDlet suite.</td>
</tr>
<tr>
<td><strong>static</strong> openRecordStore(String recordStoreName, String vendorName, String suiteName, String password)</td>
<td>Open a record store associated with the named MIDlet suite.</td>
</tr>
<tr>
<td><strong>void</strong> removeRecordListener(RecordListener listener)</td>
<td>Removes the specified RecordListener.</td>
</tr>
<tr>
<td><strong>void</strong> setMode(int authmode, boolean writeable)</td>
<td>Changes the access mode for this RecordStore.</td>
</tr>
<tr>
<td><strong>void</strong> setRecord(int recordId, byte[] newData, int offset, int numBytes)</td>
<td>Sets the data in the given record to the provided new data.</td>
</tr>
<tr>
<td><strong>void</strong> setRecord(int recordId, byte[] newData, int offset, int numBytes, int tag)</td>
<td>Sets the data in the given record to the provided new data.</td>
</tr>
</tbody>
</table>

#### Methods inherited from class **Object**

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

### Fields

**AUTHMODE_ANY**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final int</td>
<td>AUTHMODE_ANY</td>
</tr>
</tbody>
</table>

Authorization to allow access to any MIDlet suites. AUTHMODE_ANY has a value of 1.

Constant value: 1
AUTHMODE_APPLEVEL

public static final int AUTHMODE_APPLEVEL

Authorization to allow access to only certain identified MIDlets. These MIDlets are identified using the application level access control mechanism. AUTHMODE_APPLEVEL has a value of 2.

Constant value: 2

Since: MIDP 3.0

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

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, and returns the recordId for this new record. This is a blocking atomic operation. The record data MUST be queued to be persisted before the method returns, and MUST be available for retrieval after the method returns. However, the record data MAY be written to persistent storage by the implementation after the method returns. If this method is used to add records to the record store, the records have a tag value of 0.

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
  ArrayIndexOutOfBoundsException - if either offset or numBytes is negative, or if offset + numBytes > data.length

addRecord

public int addRecord(byte[] data, int offset, int numBytes, int tag)
throws javax.microedition.rms.RecordStoreNotOpenException,
javax.microedition.rms.RecordStoreException,
javax.microedition.rms.RecordStoreFullException

Adds a new record to the record store and associates the record with the given tag. This is a blocking atomic operation. The record data MUST be queued to be persisted before the method returns, and MUST be available for retrieval after the method returns. However, the record data MAY be written to persistent storage by the implementation after 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)
- `tag` - a tag value to identify the record.

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
- `ArrayIndexOutOfBoundsException` - if either `offset` or `numBytes` is negative, or if `offset + numBytes > data.length`

Since: MIDP 3.0

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. If the record store is not open, this method does nothing.

Parameters:
- `listener` - the RecordChangedListener

See Also: `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
```

The record is deleted from the record store; the recordId for this record MUST NOT be reused. This is a blocking atomic operation. The deletion operation MUST be queued to be persisted before the method returns, and the record associated with the recordId MUST NOT be available for retrieval after the method returns. However, the record deletion MAY be reflected in persistent storage by the implementation after the method returns. All the information related to the tag associated with the recordId MUST be removed as a result of this method call.

Parameters:
- `recordId` - the ID of the record to delete
Throws:
- `javax.microedition.rms.RecordStoreNotOpenException` - if the record store is not open
- `javax.microedition.rms.InvalidRecordIDException` - if the recordID is invalid
- `javax.microedition.rms.RecordStoreException` - if a general record store exception occurs
- `java.security.SecurityException` - if the MIDlet has read-only access to the RecordStore

**deleteRecordStore**

```java
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:
- `javax.microedition.rms.RecordStoreException` - if a record store-related exception occurred
- `javax.microedition.rms.RecordStoreNotFoundException` - if the record store could not be found

**enumerateRecords**

```java
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 are null, the enumeration will traverse all records in the record store in an undefined order. This is the most efficient way to traverse all of the records in a record store. If a filter is used with a null comparator, the enumeration will traverse the filtered records in an undefined order. The first call to `RecordEnumeration.nextRecord()` returns the record data from the first record in the sequence. Subsequent calls to `RecordEnumeration.nextRecord()` return the next consecutive record's data. To return the record data from the previous consecutive from any given point in the enumeration, call `previousRecord()`. On the other hand, if after creation the first call is to `previousRecord()`, the record data of the last element of the enumeration will be returned. Each subsequent call to `previousRecord()` will step backwards through the sequence.

Parameters:
- `filter` - if non-null, will be used to determine what subset of the record store records will be used
- `comparator` - if non-null, will be used to determine the order in which the records are returned
- `keepUpdated` - if true, the enumerator will keep its enumeration current with any changes in the records of the record store. Use with caution as there are possible performance consequences. If false the enumeration will not be kept current and may return recordIds for records that have been deleted or miss records that are added later. It may also return records out of order that have been modified after the enumeration was built. Note that any changes to records in the record store are accurately reflected when the record is later retrieved, either directly or through the enumeration. The thing that is risked by setting this parameter false is the filtering and sorting order of the enumeration when records are modified, added, or deleted.

Returns:
- an enumeration for traversing a set of records in the record store in an optionally specified order

Throws:
enumerateRecords

public javax.microedition.rms.RecordEnumeration enumerateRecords(RecordFilter filter,
    RecordComparator comparator,
    boolean keepUpdated,
    int[] tags)

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 are 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 record 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.
- tags - An integer array of the tags that need to be matched. This is effectively an OR operation, since each record can contain only one tag. If tags is null, the filter and comparator are applied on all the records of the record store. If tags is empty, no records are selected and an empty RecordEnumeration is returned. For all other cases, the filter and comparator are applied on any record that matches any one of the tags in the tags array.

Returns:
an enumeration for traversing a set of records in the record store in an optionally specified order

Throws:
javax.microedition.rms.RecordStoreNotOpenException - if the record store is not open

See Also: RecordEnumeration.rebuild()

Since: MIDP 3.0
exportRecordStore

public static void exportRecordStore(OutputStream os,
String recordStoreName,
String internalPassword,
String exportPassword)
throws java.io.IOException,
javax.microedition.rms.RecordStoreException,
java.lang.IllegalArgumentException,
javax.microedition.rms.RecordStoreNotFoundException,
javax.microedition.rms.SecureRecordStoreException

Exports the record store to an RMS stream (see RMS Interchange Format). The RMS-interchange-formatted stream MUST be encrypted using the AES block cipher, if a non-null value for exportPassword is passed in. The key for encrypting the stream MUST be derived from the parameter "exportPassword" as specified by PKCS#5 v 2.0. The parameter "internalPassword" is to be used for decrypting the locally encrypted RecordStore (see Locally Encrypted Record Stores).

Parameters:
os - Output stream where serialized input is to be written
recordStoreName - Name of the record store to serialize
internalPassword - Password for decryption of the record store entries. If the record store is not encrypted the parameter should be null. If a non-null value is passed in for a plaintext record store, this parameter will be ignored.
exportPassword - Password for encryption of the serialized stream. If null, the stream is not encrypted.

Throws:
IOException - if there was an stream output error
RecordStoreException - if record store cannot be read
java.lang.IllegalArgumentException - - if the recordStoreName is invalid.
RecordStoreNotFoundException - if a recordstore with name recordStoreName cannot be found
SecureRecordStoreException - if there is an error while locally decrypting a record store or encrypting an OutputStream.

See Also: importRecordStore(InputStream, String, String)
Since: MIDP 3.0

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**

```java
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**

```java
public int getNumRecords()
    throws javax.microedition.rms.RecordStoreNotOpenException
```

Returns the number of records currently in the record store. The maximum number of records that a `RecordStore` can contain is `Integer.MAX_VALUE`.

**Returns:**
the number of records currently in the record store.

**Throws:**
- `RecordStoreNotOpenException` - if the record store is not open

---

**getRecord**

```java
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:** `setRecord(int, byte[]`, `int, int)`

---

**getRecord**

```java
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.
javax.microedition.rms.RecordStore

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, or if offset is negative or greater than or equal to the buffer length

See Also: 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

getRecordStoreInfo

public javax.microedition.rms.RecordStoreInfo getRecordStoreInfo()
throws javax.microedition.rms.RecordStoreNotOpenException

Gets a RecordStoreInfo instance for this RecordStore.

Returns:
RecordStoreInfo object for the RecordStore

Throws:
RecordStoreNotOpenException - if the record store is not open

Since: MIDP 3.0

getSize

public int getSize()
throws javax.microedition.rms.RecordStoreNotOpenException

Deprecated. Previously calls to getSize() could not accommodate a RecordStore larger than
Integer.MAX_VALUE. To query the size of a RecordStore, use getRecordStoreInfo() and
RecordStoreInfo.getSize(), which returns a long and can accommodate RecordStore sizes up
to Long.MAX_VALUE.

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, tags etc.

Returns:
public int getSizeAvailable() throws javax.microedition.rms.RecordStoreNotOpenException

Deprecated. Previously calls to getSizeAvailable() could not accommodate a RecordStore larger than Integer.MAX_VALUE. To query the available size of a RecordStore, use getRecordStoreInfo() and RecordStoreInfo.getSizeAvailable(), which returns a long and can accommodate RecordStore sizes up to Long.MAX_VALUE.

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, tag data etc.

Returns:
the amount of additional room (in bytes) available for this record store to grow. If the amount of additional room available for this record store to grow exceeds java.lang.Integer.MAX_VALUE, then java.lang.Integer.MAX_VALUE MUST be returned.

Throws:
RecordStoreNotOpenException - if the record store is not open

public int getTag(int recordId) throws javax.microedition.rms.RecordStoreNotOpenException, javax.microedition.rms.InvalidRecordIDException, javax.microedition.rms.RecordStoreException

Returns the tag associated with the recordId.

Parameters:
recordId - the ID of the record to use in this operation

Returns:
the tag associated with the recordId. If the record was added by addRecord(data,offset,numBytes), then 0 is returned.

Throws:
RecordStoreNotOpenException - if the record store is not open
InvalidRecordIDException - if the recordId is invalid
RecordStoreException - if a general record store exception occurs

Since: MIDP 3.0

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
importRecordStore

public static javax.microedition.rms.RecordStore importRecordStore(InputStream is,
            String importPassword,
            String internalPassword)
throws java.io.IOException,
        javax.microedition.rms.RecordStoreException,
        javax.microedition.rms.SecureRecordStoreException

Imports the record store from an encrypted or plaintext RMS stream (see RMS Interchange Format). The Key for decrypting the stream is derived from the provided "importPassword" parameter as specified by PKCS#5 v 2.0. If a non-null value is passed in for internalPassword, then the RecordStore MUST be encrypted locally on device. See Locally Encrypted Record Stores. If a record store with the same name already exists, a RecordStoreException will be thrown. Record store import is an atomic operation: if an exception occurs while reading from the stream or decrypting the data, no record store will be created.

Parameters:

is - Input stream with the encrypted content
importPassword - password to be used in decrypting the serialized stream. If the stream is not encrypted the parameter should be null. For unencrypted streams, if a non-null value for this parameter is passed in, it will be ignored.
internalPassword - password to be used to encrypt entries in the new record store. If null, the record store will not be encrypted

Returns:
RecordStore object for the record store

Throws:
IOException - is the stream cannot be read
SecureRecordStoreException - if there is any error while decrypting a Secure inputStream or locally encrypting the RecordStore (e.g. cipher is unsupported or key has invalid length).
RecordStoreException - if record store cannot be created (for example if a record store with the same name already exists)

See Also: exportRecordStore(OutputStream, String, String, String)
Since: MIDP 3.0

listRecordStores

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

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 current MIDlet suite. If this method is called by the MIDlet when the record store is already open by the same MIDlet, 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.
javax.microedition.rms.RecordStore

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

public static javax.microedition.rms.RecordStore openRecordStore(String recordStoreName, boolean createIfNecessary, int authmode, boolean writeable)
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.
- AUTHMODE_APPLEVEL - Allows certain identified MIDlets to access the RecordStore. These MIDlets are identified using the Application level access control mechanism. The owning MIDlet suite in its JAD/Manifest declares criteria for authorizing MIDlets for access. The criteria are based on the accessing MIDlets domain, signer and/or vendor. MIDlets that do not match this criteria will not be granted access to this RecordStore. See Application Level Access Authorization).

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 the same MIDlet, 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, AUTHMODE_APPLEVEL, or AUTHMODE_ANY. This argument is ignored if the RecordStore exists.
writeable - true if the RecordStore is to be writeable 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
RecordStoreNotFoundException - if the record store could not be found
RecordStoreFullException - if the operation cannot be completed because the record store is full
IllegalArgumentException - if authmode or recordStoreName is invalid
**openRecordStore**

```java
public static javax.microedition.rms.RecordStore openRecordStore(String
  recordStoreName,
  boolean createIfNecessary,
  int authmode,
  boolean writeable,
  String password)
throws javax.microedition.rms.RecordStoreException,
javax.microedition.rms.RecordStoreFullException,
javax.microedition.rms.RecordStoreNotFoundException,
javax.microedition.rms.SecureRecordStoreException
```

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 encryption password for the record store is provided as a String. The implementation MUST
use the provided password to derive a key and encrypt records before they are persisted. Retrieving a record MUST decrypt the data using the same key. See [Locally Encrypted Record Stores](javascript:void(0)).

No copies of the records in plaintext form will be retained by the implementation. The MIDlet suite
accessing the secure record store should also release references to plaintext records as soon as possible.

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 the same MIDlet, this
method returns a reference to the same RecordStore object.

**Parameters:**
- `recordStoreName` - The MIDlet suite unique name for the record store, consisting of between 1 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, AUTHMODE_APPLEVEL, or AUTHMODE_ANY. This argument is ignored if the RecordStore exists.
- `writeable` - true if the RecordStore is to be writeable by other MIDlet suites that are granted access. This argument is ignored if the RecordStore exists.
- `password` - The password to be used for generation of an encryption key for encryption of records, consisting of Unicode characters with a recommended minimum length of eight characters.

**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
- `SecureRecordStoreException` - If there is an internal error while making the record store secure (e.g. an error occurs while initializing cryptographic functions)
- `SecurityException` - If the `recordStoreName` corresponds to an existing `RecordStore` in the current MIDlet Suite and the `password` does not match that of the existing `RecordStore`
- `IllegalArgumentException` - if `recordStoreName` is invalid

**Since:** MIDP 3.0
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. If authorization mode is used:

- **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.
- **AUTHMODE_APPLEVEL** - Succeeds only if vendorName and suiteName identify a MIDlet suite
whose access authorization requirements are satisfied by the current MIDlet suite.

If this method is called by a MIDlet when the record store is already open by the same MIDlet, 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)

This may be used to open a plaintext record store owned by a LIBlet.

**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

closeRecordStore

public static void closeRecordStore(javax.microedition.rms.RecordStore recordStore)
throws javax.microedition.rms.RecordStoreException

Closes the specified record store. The record store is no longer accessible to any MIDlet.

**Parameters:**

- recordStore - the record store

**Throws:**

- RecordStoreException - if a record store-related exception occurred
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.

If this method is called by a MIDlet when the record store is already open by the same MIDlet, 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, AUTHMODE_PRIVATE, true, password). (Note: The values of authmode and writeable are ignored for an existing record store)

This may be used to open an encrypted record store owned by a LIBlet.

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
- password - the password for use in decrypting the record store.

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 the calling MIDlet suite is not allowed to open the specified RecordStore, or if the recordStoreName corresponds to an existing RecordStore in the current MIDlet Suite and the password does not match that of the existing RecordStore
- IllegalArgumentException - if any of the vendor name, suite name or record store name are invalid
- SecureRecordStoreException - If there is an internal error while making the record store secure (e.g. an error occurs while initializing cryptographic functions)

Since: MIDP 3.0

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: addRecordListener(RecordListener)

setMode

public void setMode(int authmode, boolean writeable)

throws javax.microedition.rms.RecordStoreException
Changes the access mode for this RecordStore. Please use this method carefully. 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.
- **AUTHMODE_APPLEVEL** - Succeeds only if vendorName and suiteName identify a MIDlet suite whose access authorization requirements are satisfied by the current MIDlet suite. See Application Level Access Authorization.

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.

This method can only be called if this recordstore is NOT open by a MIDlet in this suite or in a different MIDlet suite. If this recordStore is open by any MIDlet, an IllegalStateException will be thrown by the method.

**Parameters:**
- `authmode` - the mode under which to check or create access. Must be one of AUTHMODE_PRIVATE, AUTHMODE_APPLEVEL, or AUTHMODE_ANY.
- `writeable` - true if the RecordStore is to be writeable 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
- `IllegalStateException` - if any MIDlet from this MIDlet suite or a different MIDlet suite has this RecordStore open.

**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 the provided new data. After this method returns, a call to `getRecord(int recordId)` will return an array of numBytes size containing the data supplied here. The record data MUST be queued to be persisted before the method returns, and MUST be available for retrieval after the method returns. However, the record data MAY be written to persistent storage by the implementation after the method returns. The tag value associated with the record is unchanged when this method is used to set the record data.

**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
- `ArrayIndexOutOfBoundsException` - if either `offset` or `numBytes` is negative, or if `offset + numBytes > newData.length`
setRecord

public void setRecord(int recordId, byte[] newData, int offset, int numBytes, int tag)
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 the provided new data. After this method returns, a call to getRecord(int recordId) will return an array of numBytes size containing the data supplied here. The record data MUST be queued to be persisted before the method returns, and MUST be available for retrieval after the method returns. However, the record data MAY be written to persistent storage by the implementation after the method returns. The existing tag is replaced with the new tag value.

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
- tag - a tag value to identify the 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
- ArrayIndexOutOfBoundsException - if either offset or numBytes is negative, or if offset + numBytes > newData.length

See Also: getRecord(int, byte[], int)

Since: MIDP 3.0
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,
    javax.microedition.rms.SecureRecordStoreException

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>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>RecordStoreException(String message)</td>
</tr>
<tr>
<td></td>
<td>Constructs a new RecordStoreException with the specified detail message.</td>
</tr>
<tr>
<td>public</td>
<td>RecordStoreException()</td>
</tr>
<tr>
<td></td>
<td>Constructs a new RecordStoreException with no detail message.</td>
</tr>
</tbody>
</table>

Methods inherited from class Throwable

getMessage, printStackTrace, 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>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>RecordStoreFullException(String message)</td>
</tr>
<tr>
<td>Constructs a new RecordStoreFullException with the specified detail message.</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>RecordStoreFullException()</td>
</tr>
<tr>
<td>Constructs a new RecordStoreFullException with no detail message.</td>
<td></td>
</tr>
</tbody>
</table>

Methods inherited from class Throwable

getMessage, printStackTrace, 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
RecordStoreInfo

Declaration
public final class RecordStoreInfo

Object
    +--javax.microedition.rms.RecordStoreInfo

Description
A class representing information about a RecordStore, including authorization mode, encryption status, writeable status, and size information. The RecordStoreInfo information remains up to date for as long as the associated RecordStore remains open.
Since: MIDP 3.0

Method Summary
<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>int getAuthMode()</td>
<td>int getAuthMode()</td>
</tr>
<tr>
<td>long getSize()</td>
<td>long getSize()</td>
</tr>
<tr>
<td>Returns the amount of space, in bytes, that the record store occupies.</td>
<td></td>
</tr>
<tr>
<td>long getSizeAvailable()</td>
<td>long getSizeAvailable()</td>
</tr>
<tr>
<td>Returns the amount of additional room (in bytes) available for this record store to grow.</td>
<td></td>
</tr>
<tr>
<td>boolean isEncrypted()</td>
<td>boolean isEncrypted()</td>
</tr>
<tr>
<td>Return a boolean indicating if the RecordStore associated with this RecordStoreInfo instance is encrypted.</td>
<td></td>
</tr>
<tr>
<td>boolean isWriteable()</td>
<td>boolean isWriteable()</td>
</tr>
<tr>
<td>Return the boolean writeable state for the RecordStore associated with this RecordStoreInfo instance.</td>
<td></td>
</tr>
</tbody>
</table>

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

Methods

getAuthMode
public int getAuthMode()
throws javax.microedition.rms.RecordStoreNotOpenException

Returns:
javax.microedition.rms.RecordStoreInfo

- RecordStore.AUTHMODE_PRIVATE
- RecordStore.AUTHMODE_ANY
- RecordStore.AUTHMODE_APPLEVEL

Throws:
RecordStoreNotOpenException - if the record store is not open

getSize

public long 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, tags etc.

Returns:
the size of the record store in bytes. If the size of the record store exceeds java.lang.Long.MAX_VALUE, then java.lang.Long.MAX_VALUE MUST be returned.

Throws:
RecordStoreNotOpenException - if the record store is not open

getSizeAvailable

public long 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, tag data etc.

Returns:
the amount of additional room available for this record store to grow. If the amount of additional room available for this record store to grow exceeds java.lang.Long.MAX_VALUE, then java.lang.Long.MAX_VALUE MUST be returned.

Throws:
RecordStoreNotOpenException - if the record store is not open

isEncrypted

public boolean isEncrypted() throws javax.microedition.rms.RecordStoreNotOpenException

Return a boolean indicating if the RecordStore associated with this RecordStoreInfo instance is encrypted.

Returns:
true if the RecordStore is encrypted, false otherwise

Throws:
RecordStoreNotOpenException - if the record store is not open

isWriteable

public boolean isWriteable() throws javax.microedition.rms.RecordStoreNotOpenException

Return the boolean writeable state for the RecordStore associated with this RecordStoreInfo instance.
javax.microedition.rms.RecordStoreInfo

**Returns:**
true if the RecordStore is to be writable by other MIDlet suites that are granted access, false otherwise

**Throws:**
*RecordStoreNotOpenException* - if the record store is not open
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>public</th>
<th>RecordStoreNotFoundException(String message)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructs a new RecordStoreNotFoundException with the specified detail message.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public</th>
<th>RecordStoreNotFoundException()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructs a new RecordStoreNotFoundException with no detail message.</td>
<td></td>
</tr>
</tbody>
</table>

Methods inherited from class Throwable
getMessage, printStackTrace, 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>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RecordStoreNotOpenException(String message)</td>
<td>Constructs a new RecordStoreNotOpenException with the specified detail message.</td>
</tr>
<tr>
<td>RecordStoreNotOpenException()</td>
<td>Constructs a new RecordStoreNotOpenException with no detail message.</td>
</tr>
</tbody>
</table>

Methods inherited from class Throwable

getMessage, printStackTrace, 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
javax.microedition.rms

SecureRecordStoreException

Declaration

public class SecureRecordStoreException extends RecordStoreException

Object

+- Throwable
  | +- Exception
  |     +--javax.microedition.rms.RecordStoreException
  |     |     +--javax.microedition.rms.SecureRecordStoreException

Description

Thrown to indicate that a problem occurred during the process of Encrypting or Decrypting data of a Secure RecordStore.

Since: MIDP 3.0

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public SecureRecordStoreException(String message)</td>
<td>Constructs a new SecureRecordStoreException with the specified detail message.</td>
</tr>
<tr>
<td>public SecureRecordStoreException()</td>
<td>Constructs a new SecureRecordStoreException with no detail message.</td>
</tr>
</tbody>
</table>

Methods inherited from class Throwable

gMessage, printStackTrace, printStackTrace, toString

Methods inherited from class Object

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

Constructors

SecureRecordStoreException

public SecureRecordStoreException()

Constructs a new SecureRecordStoreException with no detail message.

SecureRecordStoreException

public SecureRecordStoreException(String message)

Constructs a new SecureRecordStoreException with the specified detail message.

Parameters:
javax.microedition.rms.SecureRecordStoreException

message - the detail message
Package
javax.microedition.event

Description
Package to provide events for application to application communication and provide access to and events for changes in system states. The event package is comprised of one interface and three classes.

The javax.microedition.event.EventManager class is the starting point in the events package. It provides methods to query system states, register listeners to be notified on events, and to register the application to be launched on events. The events are delivered through javax.microedition.event.EventDataListener interface that the applications implement. The event notifications encapsulate the information about the event into the javax.microedition.event.EventData objects. The EventData class defines the names of system events. Access to events is controlled by the javax.microedition.event.EventPermission class.

Since: MIDP 3.0
### Class Summary

#### Interfaces

**EventDataListener**

With this interface applications are able to receive events from the system and from other applications.

#### Classes

**EventData**

The **EventData** class provides methods to create events and to retrieve information from events.

**EventManager**

The **EventManager** class provides methods to post and listen for events, to determine the events supported by the device, and to register applications to be automatically launched in response to events.

**EventPermission**

**EventPermission** allows access to system events.
javax.microedition.event

EventData

Description

The `EventData` class provides methods to create events and to retrieve information from events. The predefined system events are also defined in this class.

Each event has a name, a typed value, a source and may have a message and an information object. The source of the event is either system or another application. The typed value must be one of long, double, boolean, or `String`. Events can be constructed from any primitive type but are converted to one of the types above using the normal Java Language Specification rules for type conversions (see [JLS] Chapter 5, Conversions and Promotions). The methods `getInt`, `getLong`, `getFloat`, `getDouble`, `getBoolean`, `getString` return the value by converting the value to the type requested as defined in each method. The method `getValue` returns the value as an Object.

The information object supports a limited set of types to keep the implementation simple. The valid types allowed for the `info` object is restricted to: `java.lang.String`, `java.lang.Boolean`, `java.lang.Double`, `java.lang.Long`, and `byte[]`.

The `EventData` class and `EventManager` implementations MUST support passing the event name, value, message and info object totaling at least 1024 bytes. Each character of the `String` objects, including event name, value, and message is counted as two bytes. The number of bytes counted for the primitive value types are: boolean: 1, double: 8, and long: 8. The info object size is counted as the array length if it is a byte array; for all other info object types, it is considered to be exactly the same size as its corresponding primitive data type. The implementation overhead exceeding the previously defined sizes for a particular field MUST NOT be counted towards the 1024 byte requirement.

System Property Events

Some system properties have values that may change at runtime. If notification of changes to system properties is desired then the specification of each such system property must define the associated event of the same name. The system property event values are `String` objects and will return the current value of the system property as the value of the event. The implementation MUST ensure that when the system property is set that the corresponding event is posted. The event name for each such system property MUST be returned from the `EventManager.getSystemEvents` method.

Since: MIDP 3.0

Field Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>APPLICATION_RELAUNCH_PREFIX</code></td>
<td>Prefix of system event names that indicate an attempt was made to relaunch a target MIDlet that is already running.</td>
<td>Value: <code>APPLICATION_RELAUNCH</code>:</td>
</tr>
<tr>
<td>Method</td>
<td>Constant</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>public static final</td>
<td>AUDIO_MUTE</td>
<td>Indicates whether all audio output devices under the control of the implementation are muted. Value: AUDIO_MUTE</td>
</tr>
<tr>
<td>public static final</td>
<td>BACKLIGHT</td>
<td>Indicates the backlight illumination level. Value: BACKLIGHT</td>
</tr>
<tr>
<td>public static final</td>
<td>BACKLIGHT_DIM</td>
<td>Indicates the backlight is dim. Value: 2</td>
</tr>
<tr>
<td>public static final</td>
<td>BACKLIGHT_OFF</td>
<td>Indicates the backlight is off. Value: 0</td>
</tr>
<tr>
<td>public static final</td>
<td>BACKLIGHT_ON</td>
<td>Indicates the backlight is on. Value: 1</td>
</tr>
<tr>
<td>public static final</td>
<td>BATTERY_CHARGING</td>
<td>Indicates whether the battery is currently charging. Value: BATTERY_CHARGING</td>
</tr>
<tr>
<td>public static final</td>
<td>BATTERY_LEVEL</td>
<td>Indicates the amount of charge remaining in the battery as a percent represented by int values of 0 to 100 inclusive. Value: BATTERY_LEVEL</td>
</tr>
<tr>
<td>public static final</td>
<td>BATTERY_LOW</td>
<td>Indicates if the battery is low. Value: BATTERY_LOW</td>
</tr>
<tr>
<td>public static final</td>
<td>DATA_NETWORK</td>
<td>Indicates the available data networks. Value: DATA_NETWORK</td>
</tr>
<tr>
<td>public static final</td>
<td>EXTERNAL_POWER</td>
<td>Indicates whether the device is connected to an external power source. Value: EXTERNAL_POWER</td>
</tr>
<tr>
<td>public static final</td>
<td>FLIGHT_MODE</td>
<td>Indicates that the device is in flight mode. Value: FLIGHT_MODE</td>
</tr>
<tr>
<td>public static final</td>
<td>NETWORK_3GPP_CSD</td>
<td>Data network technology name for circuit switched data (CSD). Value: 3GPP_CSD</td>
</tr>
<tr>
<td>public static final</td>
<td>NETWORK_3GPP_PD</td>
<td>Data network technology name for packet data in the GSM network. Value: 3GPP_PD</td>
</tr>
<tr>
<td>public static final</td>
<td>NETWORK_3GPP_PD_3G</td>
<td>Data network technology name for packet data in the 3G(UMTS) network. Value: 3GPP_3G</td>
</tr>
<tr>
<td>Method</td>
<td>Definition</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------------</td>
<td></td>
</tr>
<tr>
<td>NETWORK_3GPP_PD_EDGE</td>
<td>Data network technology name for packet data in the EDGE network. Value: 3GPP_EDGE</td>
<td></td>
</tr>
<tr>
<td>NETWORK_3GPP_PD_HSDPA</td>
<td>Data network technology name for high speed download packet data in the 3G network (HSDPA). Value: 3GPP_HSDPA</td>
<td></td>
</tr>
<tr>
<td>NETWORK_802DOT11</td>
<td>Data network technology name for the 802.11 WiFi family of interfaces. Value: 802.11</td>
<td></td>
</tr>
<tr>
<td>NETWORK_802DOT16</td>
<td>Data network technology name for the 802.16 WiMax interface standards. Value: 802.16</td>
<td></td>
</tr>
<tr>
<td>NETWORK_CDMA</td>
<td>Data network technology name for the CDMA/3GPP2 defined set of radio interface technologies. Value: CDMA</td>
<td></td>
</tr>
<tr>
<td>PROFILE_ACTIVATED</td>
<td>Indicates that a named profile was activated. Value: PROFILE_ACTIVATED</td>
<td></td>
</tr>
<tr>
<td>PROFILE_GENERAL</td>
<td>Constant for the General profile. Value: GENERAL</td>
<td></td>
</tr>
<tr>
<td>PROFILE_MEETING</td>
<td>Constant for the Meeting profile. Value: MEETING</td>
<td></td>
</tr>
<tr>
<td>PROFILE_OFFLINE</td>
<td>Constant for the Offline profile (also known as &quot;Flight Mode&quot;). Value: OFFLINE</td>
<td></td>
</tr>
<tr>
<td>PROFILE_OUTDOOR</td>
<td>Constant for the Outdoor profile. Value: OUTDOOR</td>
<td></td>
</tr>
<tr>
<td>PROFILE_PAGER</td>
<td>Constant for the Pager profile. Value: PAGER</td>
<td></td>
</tr>
<tr>
<td>PROFILE_SILENT</td>
<td>Constant for the Silent profile. Value: SILENT</td>
<td></td>
</tr>
<tr>
<td>PROFILE_SYSTEM1</td>
<td>Constant for first system-reserved profile. Value: SYSTEM1</td>
<td></td>
</tr>
<tr>
<td>PROFILE_SYSTEM2</td>
<td>Constant for second system-reserved profile. Value: SYSTEM2</td>
<td></td>
</tr>
<tr>
<td>Field Name</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>PROFILE_SYSTEM3</td>
<td>Constant for third system-reserved profile. Value: SYSTEM3</td>
<td></td>
</tr>
<tr>
<td>PROFILE_SYSTEM4</td>
<td>Constant for fourth system-reserved profile. Value: SYSTEM4</td>
<td></td>
</tr>
<tr>
<td>PROFILE_USER1</td>
<td>Constant for first user-defined profile. Value: USER1</td>
<td></td>
</tr>
<tr>
<td>PROFILE_USER2</td>
<td>Constant for second user-defined profile. Value: USER2</td>
<td></td>
</tr>
<tr>
<td>PROFILE_USER3</td>
<td>Constant for third user-defined profile. Value: USER3</td>
<td></td>
</tr>
<tr>
<td>PROFILE_USER4</td>
<td>Constant for fourth user-defined profile. Value: USER4</td>
<td></td>
</tr>
<tr>
<td>SCREENSAVER_MODE</td>
<td>Indicates the changes in the screen saver execution as a String value. Value: SCREENSAVER_MODE</td>
<td></td>
</tr>
<tr>
<td>SCREENSAVER_MODE_ACTIVATED</td>
<td>Indicates the screen saver is being activated. Value: ACTIVATED</td>
<td></td>
</tr>
<tr>
<td>SCREENSAVER_MODE_DEACTIVATED</td>
<td>Indicates the screen saver is being deactivated. Value: DEACTIVATED</td>
<td></td>
</tr>
<tr>
<td>SYSTEM_STATE</td>
<td>Indicates the system state as a String value. Value: SYSTEM_STATE</td>
<td></td>
</tr>
<tr>
<td>SYSTEM_STATE_NORMAL</td>
<td>Indicates the system is running applications normally. Value: NORMAL</td>
<td></td>
</tr>
<tr>
<td>SYSTEM_STATE_SHUTDOWN</td>
<td>Indicates the system is shutting down. Value: SHUTDOWN</td>
<td></td>
</tr>
<tr>
<td>SYSTEM_STATE_STANDBY</td>
<td>Indicates the device is entering a potentially low power state in which no applications are running. Value: STANDBY</td>
<td></td>
</tr>
<tr>
<td>SYSTEM_STATE_STARTUP</td>
<td>Indicates the system is starting up. Value: STARTUP</td>
<td></td>
</tr>
</tbody>
</table>
public static final VOICE_CALL
Indicates whether there are voice calls currently active.
Value: VOICE_CALL

### Constructor Summary

<table>
<thead>
<tr>
<th>public</th>
<th>EventData(String event, String value, String message, Object info)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Creates a new instance of EventData with a String value.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public</th>
<th>EventData(String event, long value, String message, Object info)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Creates a new instance of EventData with a long value.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public</th>
<th>EventData(String event, double value, String message, Object info)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Creates a new instance of EventData with a double value.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public</th>
<th>EventData(String event, boolean value, String message, Object info)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Creates a new instance of EventData with a boolean value.</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>boolean</th>
<th>equals(Object object)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Checks if another object is &quot;equal&quot; to this one.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>boolean</th>
<th>getBoolean()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets the value as a boolean.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>double</th>
<th>getDouble()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets the value as a double.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>float</th>
<th>getFloat()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets the value as a float.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.Object</th>
<th>getInfo()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets information regarding the event.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int</th>
<th>getInt()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets the value as an int.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>long</th>
<th>getLong()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets the value as a long.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getMessage()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets the message for this event.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getName()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets the name of the event.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>javax.microedition.midlet.MIDletIdentity</th>
<th>getSourceInfo()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets the information about the source of this event.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getString()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets the value as a String.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>long</th>
<th>getTimestamp()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets the time when the event was posted.</td>
</tr>
</tbody>
</table>
javax.microedition.event.EventData

<table>
<thead>
<tr>
<th>java.lang.Object</th>
<th>getValue()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets the value as a java.lang.Object.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int</th>
<th>hashCode()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets the hash code value for this object.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>toString()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets the value as a String.</td>
</tr>
</tbody>
</table>

Methods inherited from class java.lang.Object

- getClass
- notify
- notifyAll
- wait
- wait
- wait

Fields

**APPLICATION_RELAUNCH_PREFIX**

```java
public static final java.lang.String APPLICATION_RELAUNCH_PREFIX
```

Prefix of system event names that indicate an attempt was made to relaunch a target MIDlet that is already running. The full event name includes the prefix, the vendor, name, and version of the target MIDlet formatted according to MIDletIdentity.toString(). Specifically, it is the concatenation of APPLICATION_RELAUNCH_PREFIX + target.getMIDletIdentity().toString() where target is an placeholder for the target MIDlet. The type of the value is boolean and the value is always true.

To be notified of relaunch attempts the MIDlet should concatenate this prefix with the string returned from MIDletIdentity.toString() and register to listen for the event.

Constant value: **APPLICATION_RELAUNCH**

**AUDIO_MUTE**

```java
public static final java.lang.String AUDIO_MUTE
```

Indicates whether all audio output devices under the control of the implementation are muted. A boolean value of true indicates all audio output devices are muted and false otherwise.

Constant value: **AUDIO_MUTE**

**BACKLIGHT**

```java
public static final java.lang.String BACKLIGHT
```

Indicates the backlight illumination level. The value of this state MUST NOT be changed due to calls to Display.flashBacklight(int). The value SHOULD NOT be changed due to any programmatic flashing of the backlight through any other API. The value is an integer with values: BACKLIGHT_ON, BACKLIGHT_OFF, or BACKLIGHT_DIM.

Constant value: **BACKLIGHT**

**BACKLIGHT_DIM**

```java
public static final int BACKLIGHT_DIM
```

Indicates the backlight is dim. Any illumination level other than fully on or off MUST be reported as dim.

Constant value: 2

See Also: BACKLIGHT

**BACKLIGHT_OFF**

```java
public static final int BACKLIGHT_OFF
```

Indicates the backlight is off.
Indicates the backlight is off.
Constant value: 0

See Also: BACKLIGHT

---

**BACKLIGHT_ON**

```java
public static final int BACKLIGHT_ON
```

Indicates the backlight is on.
Constant value: 1

See Also: BACKLIGHT

---

**BATTERY_CHARGING**

```java
public static final java.lang.String BATTERY_CHARGING
```

Indicates whether the battery is currently charging. A boolean value of true indicates that the battery is charging and is false otherwise.
Constant value: BATTERY_CHARGING

---

**BATTERY_LEVEL**

```java
public static final java.lang.String BATTERY_LEVEL
```

Indicates the amount of charge remaining in the battery as a percent represented by int values of 0 to 100 inclusive. Implementations SHOULD map the native battery levels to an aggregate percentile of all batteries in the device. Only changes in battery level detectable by the implementation must be reported.
Constant value: BATTERY_LEVEL

---

**BATTERY_LOW**

```java
public static final java.lang.String BATTERY_LOW
```

Indicates if the battery is low. The value is a boolean, if true the battery is low, otherwise the battery is ok. If true MIDlets should save their work and conserve power.
Constant value: BATTERY_LOW

---

**DATA_NETWORK**

```java
public static final java.lang.String DATA_NETWORK
```

Indicates the available data networks. This event is sent to the requested applications whenever there is a change in the list of available data networks. This notification MUST be sent if a new data network appears or a data network is lost. The event contains the list of all currently available data networks. The data networks are indicated as a semicolon separated list of the technology names in the EventData value field. The data network technology names MUST be the following:

- NETWORK_3GPP_CSD
- NETWORK_3GPP_PD
- NETWORK_3GPP_PD_EDGE
- NETWORK_3GPP_PD_3G
- NETWORK_3GPP_PD_HSDPA
- NETWORK_802DOT11
- NETWORK_802DOT16
- NETWORK_CDMA

Constant value: DATA_NETWORK
javax.microedition.event.EventData

EXTERNAL_POWER

public static final java.lang.String EXTERNAL_POWER

Indicates whether the device is connected to an external power source. A boolean value of true indicates that the device is connected, otherwise the value is false.
Constant value: EXTERNAL_POWER

FLIGHT_MODE

public static final java.lang.String FLIGHT_MODE

Indicates that the device is in flight mode. For example, the device’s radio(s) are powered off. The value is a boolean, true indicates that flight mode is active; otherwise false.
Constant value: FLIGHT_MODE

NETWORK_3GPP_CSD

public static final java.lang.String NETWORK_3GPP_CSD

Data network technology name for circuit switched data (CSD). This is part of the 3GPP defined set of radio interface technologies.
Constant value: 3GPP_CSD

NETWORK_3GPP_PD

public static final java.lang.String NETWORK_3GPP_PD

Data network technology name for packet data in the GSM network. This is part of the 3GPP defined set of radio interface technologies.
Constant value: 3GPP_PD

NETWORK_3GPP_PD_3G

public static final java.lang.String NETWORK_3GPP_PD_3G

Data network technology name for packet data in the 3G(UMTS) network. This is part of the 3GPP defined set of radio interface technologies.
Constant value: 3GPP_3G

NETWORK_3GPP_PD_EDGE

public static final java.lang.String NETWORK_3GPP_PD_EDGE

Data network technology name for packet data in the EDGE network. This is part of the 3GPP defined set of radio interface technologies.
Constant value: 3GPP_EDGE

NETWORK_3GPP_PD_HSDPA

public static final java.lang.String NETWORK_3GPP_PD_HSDPA

Data network technology name for high speed download packet data in the 3G network (HSDPA). This is part of the 3GPP defined set of radio interface technologies.
Constant value: 3GPP_HSDPA

NETWORK_802DOT11

public static final java.lang.String NETWORK_802DOT11
Data network technology name for the 802.11 WiFi family of interfaces. This value includes access technology variants of 802.11a, 802.11b, 802.11g and 802.11n, along with support for any of the optional QoS and hand-over extensions.

Constant value: 802.11

**NETWORK_802DOT16**

public static final java.lang.String NETWORK_802DOT16

Data network technology name for the 802.16 WiMax interface standards. This is primarily for the 802.16e mobile WiMax specification, but can also be used for the fixed 802.16 specification.

Constant value: 802.16

**NETWORK_CDMA**

public static final java.lang.String NETWORK_CDMA

Data network technology name for the CDMA/3GPP2 defined set of radio interface technologies. This includes CDMA, CDMA-2000, CDMA 1X, CDMA-EvDO and EVDO rev A(DOrA).

Constant value: CDMA

**PROFILE_ACTIVATED**

public static final java.lang.String PROFILE_ACTIVATED

Indicates that a named profile was activated. Only one profile at a time can be active, so that when a profile is activated, the previously active profile is deactivated, but there is no specific event for this implicit operation. The initial profile is implementation dependent. The type of the event is String. The event String value MUST be one of the following:

- PROFILE_GENERAL
- PROFILE_SILENT
- PROFILE_MEETING
- PROFILE_OUTDOOR
- PROFILE_PAGER
- PROFILE_OFFLINE
- PROFILE_USER1
- PROFILE_USER2
- PROFILE_USER3
- PROFILE_USER4
- PROFILE_SYSTEM1
- PROFILE_SYSTEM2
- PROFILE_SYSTEM3
- PROFILE_SYSTEM4

USER1...4 and SYSTEM1...4 are placeholders for user-defined profiles and existing system profiles that do not fit the defined semantics. As event information, the implementation SHOULD set a String containing the display name of the activated profile, if any. Applications MUST NOT be able to post PROFILE_ACTIVATED events to other applications or the system.

Constant value: PROFILE_ACTIVATED

**PROFILE_GENERAL**

public static final java.lang.String PROFILE_GENERAL

Constant for the General profile. All tones are set to the default, and there is no need for the application to restrict its audio output.

Constant value: GENERAL

**PROFILE_MEETING**

public static final java.lang.String PROFILE_MEETING
Constant for the Meeting profile. The application is expected to issue only brief audio segments. Visual cues are preferred. Constant value: `MEETING`

**PROFILE_OFFLINE**

```java
public static final java.lang.String PROFILE_OFFLINE
```

Constant for the Offline profile (also known as “Flight Mode”). Typically no network connectivity is available when this profile is active, but applications SHOULD NOT rely on this to find out about network availability. In terms of audio, this profile is like General. Constant value: `OFFLINE`

**PROFILE_OUTDOOR**

```java
public static final java.lang.String PROFILE_OUTDOOR
```

Constant for the Outdoor profile. Audio output does not need to be restricted, and it could even be louder than usual. Constant value: `OUTDOOR`

**PROFILE_PAGER**

```java
public static final java.lang.String PROFILE_PAGER
```

Constant for the Pager profile. The application is expected to issue only alert tones, and audio playback is not desirable. Constant value: `PAGER`

**PROFILE_SILENT**

```java
public static final java.lang.String PROFILE_SILENT
```

Constant for the Silent profile. The application is not expected to play any audio when this profile is active. Constant value: `SILENT`

**PROFILE_SYSTEM1**

```java
public static final java.lang.String PROFILE_SYSTEM1
```

Constant for first system-reserved profile. Used if the profile does not match any of the predefined profile semantics. Constant value: `SYSTEM1`

**PROFILE_SYSTEM2**

```java
public static final java.lang.String PROFILE_SYSTEM2
```

Constant for second system-reserved profile. Constant value: `SYSTEM2`

**PROFILE_SYSTEM3**

```java
public static final java.lang.String PROFILE_SYSTEM3
```

Constant for third system-reserved profile. Constant value: `SYSTEM3`
PROFILE_SYSTEM4
public static final java.lang.String PROFILE_SYSTEM4

Constant for fourth system-reserved profile.
Constant value: SYSTEM4

PROFILE_USER1
public static final java.lang.String PROFILE_USER1

Constant for first user-defined profile.
Constant value: USER1

PROFILE_USER2
public static final java.lang.String PROFILE_USER2

Constant for second user-defined profile.
Constant value: USER2

PROFILE_USER3
public static final java.lang.String PROFILE_USER3

Constant for third user-defined profile.
Constant value: USER3

PROFILE_USER4
public static final java.lang.String PROFILE_USER4

Constant for fourth user-defined profile.
Constant value: USER4

SCREENSAVER_MODE
public static final java.lang.String SCREENSAVER_MODE

Indicates the changes in the screen saver execution as a String value. Valid values are:

- SCREENSAVER_MODE_ACTIVATED - The screen saver is activated.
- SCREENSAVER_MODE_DEACTIVATED - The screen saver is deactivated.

Constant value: SCREENSAVER_MODE

SCREENSAVER_MODE_ACTIVATED
public static final java.lang.String SCREENSAVER_MODE_ACTIVATED

Indicates the screen saver is being activated.
Constant value: ACTIVATED
See Also: SCREENSAVER_MODE

SCREENSAVER_MODE_DEACTIVATED
public static final java.lang.String SCREENSAVER_MODE_DEACTIVATED

Indicates the screen saver is being deactivated.
Constant value: `DEACTIVATED`

**See Also:** [SCREENSAVER_MODE](#)

---

**SYSTEM_STATE**

```java
public static final java.lang.String SYSTEM_STATE
```

Indicates the system state as a `String` value. Valid values are `SYSTEM_STATE_STARTUP`, `SYSTEM_STATE_NORMAL`, `SYSTEM_STATE_SHUTDOWN`, and `SYSTEM_STATE_STANDBY`.

Events for each system state are delivered when the system enters the state. The current system state can be queried with `EventManager.getCurrent`. When the system is initialized the STARTUP state is entered and startup applications are automatically launched. When the system is ready to begin running MIDlets under user control or resuming from the standby state the NORMAL state is entered. When the system is in a low power or standby state the STANDBY state is entered. When the system is shutting down applications the SHUTDOWN state is entered.

Constant value: `SYSTEM_STATE`

---

**SYSTEM_STATE_NORMAL**

```java
public static final java.lang.String SYSTEM_STATE_NORMAL
```

Indicates the system is running applications normally.

Constant value: `NORMAL`

**See Also:** `SYSTEM_STATE`

---

**SYSTEM_STATE_SHUTDOWN**

```java
public static final java.lang.String SYSTEM_STATE_SHUTDOWN
```

Indicates the system is shutting down.

Constant value: `SHUTDOWN`

**See Also:** `SYSTEM_STATE`

---

**SYSTEM_STATE_STANDBY**

```java
public static final java.lang.String SYSTEM_STATE_STANDBY
```

Indicates the device is entering a potentially low power state in which no applications are running.

Constant value: `STANDBY`

**See Also:** `SYSTEM_STATE`

---

**SYSTEM_STATE_STARTUP**

```java
public static final java.lang.String SYSTEM_STATE_STARTUP
```

Indicates the system is starting up.

Constant value: `STARTUP`

**See Also:** `SYSTEM_STATE`

---

**VOICE_CALL**

```java
public static final java.lang.String VOICE_CALL
```

Indicates whether there are voice calls currently active. A `boolean` value of `true` indicates that there are voice calls active while a value of `false` indicates that no voice calls are active.

Constant value: `VOICE_CALL`
Constructors

EventData

public EventData(String event,
                boolean value,
                String message,
                Object info)

Creates a new instance of EventData with a boolean value. The event name may be either a custom application specific event name or one of the predefined system events.

Parameters:
  event - The name of the event.
  value - The boolean value for this event.
  message - The message text explaining the current event, or null if this event has no message.
  info - Additional information about this event, as an Object or null for no information.

Throws:
  NullPointerException - If the event parameter is null.
  IllegalArgumentException - If the Java type of info is not one of the valid types.

EventData

public EventData(String event,
                double value,
                String message,
                Object info)

Creates a new instance of EventData with a double value. The event name may be either a custom application specific event name or one of the predefined system events.

Parameters:
  event - The name of the event.
  value - The value for this event; the full range of double values MUST be stored.
  message - The message text explaining the current event, or null if this event has no message.
  info - Additional information about this event, as an Object or null for no information.

Throws:
  NullPointerException - If the event parameter is null.
  IllegalArgumentException - If the Java type of info is not one of the valid types.

EventData

public EventData(String event,
                long value,
                String message,
                Object info)

Creates a new instance of EventData with a long value. The event name may be either a custom application specific event name or one of the predefined system events.

Parameters:
  event - The name of the event.
  value - The value for this event; the full range of long values MUST be stored.
  message - The message text explaining the current event, or null if this event has no message.
  info - Additional information about this event, as an Object or null for no information.

Throws:
  NullPointerException - If event parameter is null
  IllegalArgumentException - If the Java type of info is not one of the valid types.

See Also: getLong(), getInt()
**EventData**

```java
class EventData {
    public EventData(String event, 
                     String value, 
                     String message, 
                     Object info)
```

Creates a new instance of EventData with a String value. The event name may be either a custom application specific event name or one of the predefined system events.

**Parameters:**
- `event` - The name of the event.
- `value` - The String value for this event
- `message` - The message text explaining the current event, or `null` if this event has no message.
- `info` - Additional information about this event, or `null` for no information.

**Throws:**
- `NullPointerException` - If `event` or `value` parameter is `null`.
- `IllegalArgumentException` - If the Java type of `info` is not one of the valid types.

**Methods**

**equals**

```java
public boolean equals(Object object)
```

Checks if another object is "equal" to this one. An EventData object is equal to another object if the other object is an instance EventData and the event name, event typed value, and the message are all equal. The additional information is ignored. The hashcode must meet the general requirements of `java.lang.Object.equals`.

**Parameters:**
- `object` - an object to compare

**Returns:**
- `true` if the object is an instance of the same class and the event name, event typed value, and the message are all equal.

**getBoolean**

```java
public boolean getBoolean()
```

Gets the value as a boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Convert to boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>(value == 0 ? false : true)</td>
</tr>
<tr>
<td>double</td>
<td>(value == 0.0f ? false : true)</td>
</tr>
<tr>
<td>boolean</td>
<td>value</td>
</tr>
<tr>
<td>String</td>
<td>value.equals(&quot;true&quot;)</td>
</tr>
</tbody>
</table>

**Returns:**
- The value as a boolean.

**getDouble**

```java
public double getDouble() 
```

Throws `java.lang.NumberFormatException`
java.microedition.event.EventData

Gets the value as a `double`.

<table>
<thead>
<tr>
<th>Value</th>
<th>Convert to float</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>(double)value</td>
</tr>
<tr>
<td>double</td>
<td>value</td>
</tr>
<tr>
<td>boolean</td>
<td>(value ? 1.0d : 0.0d)</td>
</tr>
<tr>
<td>String</td>
<td><code>java.lang.Double.parseDouble(value)</code></td>
</tr>
</tbody>
</table>

**Returns:**

The value as a `double`.

**Throws:**

`NumberFormatException` - If the value can not be parsed as a `double` value.

---

### getFloat

```java
public float getFloat()
    throws java.lang.NumberFormatException
```

Gets the value as a `float`.

<table>
<thead>
<tr>
<th>Value</th>
<th>Convert to float</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>(float)value</td>
</tr>
<tr>
<td>double</td>
<td>(float)value</td>
</tr>
<tr>
<td>boolean</td>
<td>(value ? 1.0f : 0.0f)</td>
</tr>
<tr>
<td>String</td>
<td><code>java.lang.Float.parseFloat(value)</code></td>
</tr>
</tbody>
</table>

**Returns:**

The value as a `float`.

**Throws:**

`NumberFormatException` - If the value is a `boolean` or can not be parsed as a `float` value.

---

### getInfo

```java
public java.lang.Object getInfo()
```

Gets information regarding the event. This information may include more data about the event. The instance returned will be of one of the `valid types` types. It is up to the application to cast the object to a usable type. The `instanceof` operator or the object’s `Class` may be used to determine the type of the value.

**Returns:**

Information regarding the event, or `null` if there is no further information about this event.

---

### getInt

```java
public int getInt()
    throws java.lang.NumberFormatException
```

Gets the value as an `int`.

<table>
<thead>
<tr>
<th>Value</th>
<th>Convert to int</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>(int)value</td>
</tr>
<tr>
<td>double</td>
<td>(int)value</td>
</tr>
<tr>
<td>boolean</td>
<td>(value ? 1 : 0)</td>
</tr>
<tr>
<td>String</td>
<td>java.lang.Integer.parseInt(value)</td>
</tr>
</tbody>
</table>

**Returns:**
- The value as an `int`.

**Throws:**
- `NumberFormatException` - If the value cannot be parsed as an `int`.

---

**getLong**

```java
public long getLong() throws java.lang.NumberFormatException
```

Gets the value as an `long`.

<table>
<thead>
<tr>
<th>Value</th>
<th>Convert to int</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>value</td>
</tr>
<tr>
<td>double</td>
<td>(long)value</td>
</tr>
<tr>
<td>boolean</td>
<td>(value ? 1L : 0L)</td>
</tr>
<tr>
<td>String</td>
<td>java.lang.Long.parseLong(value)</td>
</tr>
</tbody>
</table>

**Returns:**
- The value as an `long`.

**Throws:**
- `NumberFormatException` - If the value cannot be parsed as an `long`.

---

**getMessage**

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

Gets the message for this event. The event message is typically a single sentence or phrase explaining the current event.

**Returns:**
- The message text explaining the current event, or `null` if this event has no message.

---

**getName**

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

Gets the name of the event.

**Returns:**
- The name of the event.

---

**getSourceInfo**

```java
public javax.microedition.midlet.MIDletIdentity getSourceInfo()
```
java.microedition.event.EventData

Gets the information about the source of this event. If the event was posted by a MIDlet then the return value MUST be non-null and identify the MIDlet, otherwise null is returned.

Returns:
an MIDletIdentity instance containing information about the source of the event; null if this EventData was not posted by a MIDlet.

See Also: MIDletIdentity

getString

public java.lang.String getString()

Gets the value as a String. The value of the event is converted to a String depending on the type of the value:

<table>
<thead>
<tr>
<th>Value</th>
<th>Convert to String</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>java.lang.Long.toString()</td>
</tr>
<tr>
<td>double</td>
<td>java.lang.Double.toString()</td>
</tr>
<tr>
<td>boolean</td>
<td>java.lang.Boolean.toString()</td>
</tr>
<tr>
<td>String</td>
<td>value</td>
</tr>
</tbody>
</table>

Returns: The value as a String.

getTimestamp

public long getTimestamp()

Gets the time when the event was posted. The time is in the same format as java.lang.System.currentTimeMillis.

Returns: the time when the event was posted. If the event has not been posted zero is returned.

getValue

public java.lang.Object getValue()

Gets the value as a java.lang.Object. The value of the event is converted to an Object depending on the type of the value:

<table>
<thead>
<tr>
<th>Value</th>
<th>Convert to Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>new java.lang.Long(value)</td>
</tr>
<tr>
<td>double</td>
<td>new java.lang.Double(value)</td>
</tr>
<tr>
<td>boolean</td>
<td>new java.lang.Boolean(value)</td>
</tr>
<tr>
<td>String</td>
<td>new java.lang.String(value)</td>
</tr>
</tbody>
</table>

Returns: The value as an Object.

hashCode

public int hashCode()

Gets the hash code value for this object. The hashcode must meet the general requirements of java.lang.Object.hashCode.
Returns:
The hashcode is computed from the event name, event typed value, and the message.

toString

public java.lang.String toString()

Gets the value as a String. The value of the event is using getString.

Returns:
The value as a String.
javax.microedition.event
EventDataListener

Declaration

public interface EventDataListener

Description

With this interface applications are able to receive events from the system and from other applications. The EventDataListener interface MUST be implemented by an application and registered with the EventManager to receive events.

The application MUST NOT make any assumptions about synchronization of calls to the EventDataListener. The listener may be called at any time. The application is responsible for handling concurrency.

From the call to handleEvent until it returns the event system will not change the state of the EventData instance. After handleEvent returns the state of the EventData instance is unpredictable.

Since: MIDP 3.0
See Also: EventManager

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void handleEvent (EventData value)</td>
<td>Called with the EventData for the event to which this listener has been registered.</td>
</tr>
</tbody>
</table>

Methods

handleEvent

public void handleEvent (EventData value)

Called with the EventData for the event to which this listener has been registered. This method will be called when the value has changed.

Parameters:

value - An EventData object containing the value that triggered the event.
javax.microedition.event

EventManager

Declaration

public final class EventManager

Description

The EventManager class provides methods to post and listen for events, to determine the events supported by the device, and to register applications to be automatically launched in response to events. The system publishes some events and applications publish other events. Each event has a name, a value, and may contain an informational message and extra information about the event. When an event is posted it is given a timestamp and the identity of the MIDlet, if any, that posted it. See the EventData class for the definition of events.

Event Naming Conventions

The EventData class defines a set of commonly available system events. Individual devices may also support unique system events in addition to those defined in EventData. Event names are limited to the characters allowed by the Java Language Specification [JLS] for identifiers. Event names are case sensitive; for example, "COM.SUN" and "com.sun" are different names. In order to avoid name collisions names defined by applications and specific implementations SHOULD use the common reverse domain notation and SHOULD contain at least one ".". For example com.YourCompanyName.SomeCustomEvent.

When registering and unregistering listeners and for application launch, events may be requested using a prefix and an optional single final asterisk "*". The prefix must be at least 1 character long. For example, the pattern "com.mot.*" will match any and all event names that begin with "com.mot."

Security of the Event API

Some events may convey privileged information to the application and therefore the application must have the appropriate permission to access such events. Additionally, the ability to post events or to register applications to launch automatically on events must be restricted to applications with the appropriate permissions. See the EventPermission class for more details.

Application Level Access Authorization

It is possible to limit operations on the events by using the application level access restrict mechanism as described in "Application Level Access Authorization" section. The following restrictions can be done by setting the authmode to true and by using the application level access authorization:

- Only authorized source post events and notify listeners
- Only authorized source can launch applications
- Only authorized recipient can receive an event

See more details in the next sections.
Event Posting and Notification of Listeners

All events posted MUST be delivered to all valid recipients in each runtime environment on the device that implements this specification. The recipients register themselves to receive the events with `addEventListener` methods. The event API is asynchronous, events are delivered via a listener and event posts MUST NOT block the application execution. Each event MUST be delivered to each listener exactly once. Each event MUST successfully launch the application only once. An event matches a recipient if all of the following are true:

1. The event name MUST match the event name for `addEventListener` or `registerApplication` including wildcards.
2. The type of the event value MUST match the type of the value requested for `addEventListener` or `registerApplication`.
3. The source MUST permit delivery to the recipient and the recipient MUST accept delivery from a source by fulfilling the requirements set forth:
   a. The event was posted with `authmode` equal `false` or the event was posted with `authmode` equals `true` and the recipient is authorized and
   b. The recipient was requested with `authmode` equal `false` or the event was posted with `authmode` equals `true` and the source is authorized.
4. The event value must match the conditions, if any, specified by `addEventListener` or `registerApplication`.

Events that are not matched to a recipient are silently discarded. This includes the case if authorization of the source or recipient fails.

Events are discarded after they delivered to all the valid recipients. If no valid recipients can be identified the event is discarded immediately except in the case that an event caused the launch of an application the event MUST be retained until the application returns from the `MIDlet.startApp` method. This insures that the application is given a chance to listen for the event before the event is discarded. Once the listener has been registered the event is retained until it can be delivered to the listener.

For system events, as soon as a listener is added or an application is registered to be launched the current value is checked. If the current value is matched, the listener MUST be immediately notified or the application is launched. Performing the match in `addEventListener` and `registerApplication` removes a race condition that could exist between those actions and the application needing to poll the current value and check the condition.

Application Launch on Events

Applications may be launched automatically in response to events. The registration is done with `registerApplication` methods. The application will not be launched if it is already running. The application will be launched when the event value matches the requested value and the `authmode` constraints are fulfilled. Otherwise the application will not be launched. An application can register to launch on a particular event either via static jar attribute or dynamically through methods in the `EventManager` class.

Static Registration of Launch on Event Handlers

Static registration to application launch on events is done by adding the `MIDlet-Event-Launch-<n>` attribute to the JAR Manifest. The attribute consists of the class name of the application to be launched followed by the authorization mode used to check the source of the events followed an optional launch condition that MUST cause the application to be launched. For `boolean` and `String` type events the launch condition may consist of just the event name in which case the application will be launched on any change to the event.

Static registration of launch on event occurs during installation of the MIDlet suite or during a re-installation (update). If the MIDlet suite is being updated (re-installed), then any pre-existing registrations, both static and dynamic, for launching the MIDlet suite MUST be removed. Multiple `MIDlet-Event-Launch` attributes can used to register launch on several event requests. If the implementation does not
have the resources to register all of the requested launch requests the installation MUST fail with Status Code 901 (Insufficient Memory).

If the application does not have EventPermission with the name equal to the event name and action equal to "register", the installation MUST fail with Status Code 910 (Application authorization failure).

If an attempt is made to register an application for an event that is not supported by the device, or if an invalid value is requested, or if an inappropriate launch condition format is specified, then the installation MUST proceed without registering the application for the event.

The Jar Manifest attribute is defined as follows:

MIDlet-Event-Launch-<n>:<Classname>;<AuthorizationMode>;<Launch-Condition>

Where:

- **MIDlet-Event-Launch-<n>** = The application launch registration attribute name. The value of <n> starts at 1 and MUST use consecutive ordinal numbers for additional application launch registrations. The first missing entry terminates the list. Any additional entries are ignored.
- **Classname** = The fully-qualified name of the class to launch; must refer to a valid application for the current profile. For MIDP, the class must implement MIDlet found in the current MIDlet suite.
- **AuthorizationMode** = The Authorization mode specified using authmode = true|false. If authmode is true, only events from source applications that are authenticated using the access authorization mechanism are allowed to launch this application. If authmode is false, any source application that generates an event that satisfies the Launch-Condition is allowed to launch this application.
- **Launch-Condition** = The condition that should cause the application to be launched. Each MIDlet-Event-Launch-<n> attribute may include zero or one launch conditions. If one or more of the launch events is matched, then the application MUST be launched. Each launch condition MUST be of one of the following forms:

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Launch-Condition Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>&lt;Event-Name&gt;[=&quot;true&quot;</td>
</tr>
<tr>
<td>string</td>
<td>&lt;Event-Name&gt;[=&quot;&quot;&lt;String-Event-Value&gt;&quot;&quot; [, &quot;&quot;&lt;String-Event-Value&gt;&quot;&quot;&quot;]*</td>
</tr>
<tr>
<td>integer or long</td>
<td>&lt;Event-Name&gt;=&lt;Long-Event-Value&gt;,&lt;Long-Event-Value&gt;</td>
</tr>
<tr>
<td>float or double</td>
<td>&lt;Event-Name&gt;=&lt;Double-Event-Value&gt;,&lt;Double-Event-Value&gt;</td>
</tr>
</tbody>
</table>

Where:

- **Whitespace** is ignored before and after the delimiters quote ",", semi-colon ";" and comma ",".
- **Event-Name** = The name of the event on which to launch the application.
- **String-Event-Value** = The optional event value of interest. The application will be launched if the event value is equal to this value. The event value is quoted and can not contain a quote.
- **Long-Event-Value** = The application will be launched if the event value falls within the range between the low and high values (inclusive). The value must contain only characters valid for Long Literals as defined by the Java Language Specification [JLS] and are accepted by the java.lang.Long.valueOf method.
- **Double-Event-Value** = The application will be launched if the event value falls within the range between the low and high values (inclusive). The value must contain only characters valid for Floating Point Literals as defined by the Java Language Specification [JLS] and are accepted by the java.lang.Double.valueOf method.

Note: Care should be taken when registering applications to launch on events that change often as frequent attempts to launch the application may negatively impact performance. The implementation may limit the launch frequency to avoid degrading device performance or battery lifetime.
Example

Dynamic Registration of Launch on Event Handlers

In addition to statically registering an application to launch in response to events through appropriate entries in the Jar Manifest, an application can dynamically register applications using the `registerApplication` methods.

The implementation MUST support at least five (5) launch registrations whether registered via `MIDlet-Event-Launch-<n>` attributes or the API.

Code Examples

Querying a System Event Value

The application must first get an instance of `EventManager`. The `EventManager.getCurrent` method is then called to get an `EventData` object for the desired system event. The application must have the required permission to query the requested system event or a `SecurityException` will be thrown. The appropriate `getXXX` method may then be called to get the current event.

```
EventManager ssm = EventManager.getInstance(this);
try {
    battery = ssm.getCurrent(BATTERY_LEVEL);
} catch (SecurityException e) {
    ...
}
batteryLevel = battery.getInt();
```

Adding an Event Listener
The application must first get an instance of `EventManager`. The `EventManager.addListener` method is called to add the listener using system event or the application event name. The application must have the required permission to query the requested event or a `SecurityException` will be thrown.

```java
EventData battery;
boolean authmode = false;
EventManager ssm = EventManager.getInstance(this);

// Receive events at 10% and 20% battery level
ssm.addEventListener(BATTERY_LEVEL, this, authmode, 10, 20);

// Also receive events at 90% and 100% battery level
ssm.addEventListener(BATTERY_LEVEL, this, authmode, 90, 100);

// Listen for login/logout application events
ssm.addEventListener("com.foo.UserLogin", this, authmode);
ssm.addEventListener("com.foo.UserLogout, this, authmode);
```

### Handling an Event

The application had previously registered this object as a listener for multiple events and has implemented the `EventDataListener.handleEvent` method as follows.

```java
handleEvent(EventData event) {
    String eventName = event.getName();
    int batteryLevel;
    boolean bodyOpen;

    if (eventName.equals(BATTERY_LEVEL)) {
        batteryLevel = event.getInt();
    } else if (eventName.equals(BODY_OPEN)) {
        bodyOpen = event.getBoolean();
    } else if (eventName.equals("com.foo.Login")) {
        String user = event.getString();
        MIDletIdentity source = event.getSourceInfo();
        // check if the source is access authorized
        boolean authmode = source.isAuthorized();
        
        ...
    }
}
```

### Posting an Event

The application must first get an instance of `EventManager`. An `EventData` object must then be created using the appropriate constructor. The `EventManager.post` method is called to send the event. The application must have the appropriate permission to post the event or a `SecurityException` will be thrown.

```java
javax.microedition.event.EventManager
```
Registering an Application to Launch on Events

The application must first get an instance of `EventManager`. The appropriate `registerApplication` method is then called to register to be launched on the event.

```java
String username = "user1";
EventManager ssm = EventManager.getInstance(this);

EventData login = new EventData("com.foo.UserLogin", "User Login",
    username, null);
EventData appState = new EventData("APPLICATION_STATUS", "IDLE",
    "The app is idle", null);
try {
    ssm.post(login);
    ssm.post(appState);
} catch (SecurityException e) {
    ...
}
```

```java
EventData battery;
boolean authmode = false;
EventManager ssm = EventManager.getInstance(this);

try {
    // Launch the application if the battery level drops below 10%
    ssm.registerApplication(BATTERY_LEVEL, "com.fictionalco.PowerMgr",
        authmode, 0, 10);
} catch (SecurityException e) {
    ...
}
```

**Since:** MIDP 3.0

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addEventListener(String event, EventDataListener listener, boolean authmode)</code></td>
<td>Add an <code>EventListener</code> to be notified.</td>
</tr>
<tr>
<td><code>addEventListener(String event, EventDataListener listener, boolean authmode, boolean value)</code></td>
<td>Add an event listener to be notified of events with boolean values.</td>
</tr>
<tr>
<td><code>addEventListener(String event, EventDataListener listener, boolean authmode, double low, double high)</code></td>
<td>Add an event listener to be notified of an event represented by double values.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>addEventListener(String event, EventDataListener listener, boolean authmode, long low, long high)</code></td>
<td>Add an event listener to be notified of an event represented by <code>int</code> or <code>long</code> values.</td>
</tr>
<tr>
<td><code>addEventListener(String event, EventDataListener listener, boolean authmode, String[] values)</code></td>
<td>Add an event listener to be notified of an event represented by <code>String</code> values.</td>
</tr>
<tr>
<td><code>getCurrent(String event)</code></td>
<td>Returns an <code>EventData</code> object for the requested system event.</td>
</tr>
<tr>
<td><code>getInstance()</code></td>
<td>Returns the instance of <code>EventManager</code>.</td>
</tr>
<tr>
<td><code>getSystemEvents()</code></td>
<td>Returns a <code>String</code> array containing the names of all the system events available on the device.</td>
</tr>
<tr>
<td><code>post(EventData event, boolean authmode)</code></td>
<td>Post an event.</td>
</tr>
<tr>
<td><code>registerApplication(String event, String application, boolean authmode)</code></td>
<td>Register an application to be launched on an event.</td>
</tr>
<tr>
<td><code>registerApplication(String event, String application, boolean authmode, boolean value)</code></td>
<td>Register an application to be launched in response to an event represented by a <code>boolean</code> value.</td>
</tr>
<tr>
<td><code>registerApplication(String event, String application, boolean authmode, double low, double high)</code></td>
<td>Register an application to be launched in response to an event represented by a <code>double</code> value.</td>
</tr>
<tr>
<td><code>registerApplication(String event, String application, boolean authmode, long low, long high)</code></td>
<td>Register an application to be launched in response to an event represented by an <code>int</code> or <code>long</code> value.</td>
</tr>
<tr>
<td><code>registerApplication(String event, String application, boolean authmode, String[] values)</code></td>
<td>Register an application to be launched in response to an event represented by a <code>String</code> value.</td>
</tr>
<tr>
<td><code>removeEventListener(String event, EventDataListener listener)</code></td>
<td>Removes an event listener for the named event(s) including wildcards.</td>
</tr>
<tr>
<td><code>unregisterApplication(String event, String application)</code></td>
<td>Unregister an application from launching in response to an event.</td>
</tr>
</tbody>
</table>

**Methods inherited from class `Object`**

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

public void addEventListener(String event,
   EventDataListener listener,
   boolean authmode)

Add an EventListener to be notified. The listener will be notified for changes in every known event that matches the requested name and authorization mode.

If the listener is removed using removeEventListener then it will no longer be notified.

Parameters:
   event - The event(s) for which the listener will be added; if event contains a final "***", the listener will match any event name with identical prefix (without the "***").
   listener - The listener to add.
   authmode - If true, check if the source of event is authorized as specified by the MIDlet's access authorization attributes. Authorized sources are allowed to notify the listener. If false, events irrespective of source are allowed to notify the listener.

Throws:
   NullPointerException - If listener or event is null
   SecurityException - If the application does not have EventPermission with the name equal the event name and action equals "read".

See Also: EventData, removeEventListener(String, EventDataListener)

addEventListener

public void addEventListener(String event,
   EventDataListener listener,
   boolean authmode,
   boolean value)

Add an event listener to be notified of events with boolean values. The listener will be called when the value changes to the requested value. The listener will be called immediately for every known event that matches the requested name, value, and authorization mode.

If the listener is removed using removeEventListener then it will no longer be notified.

Parameters:
   event - The event(s) for which the listener will be added; if event contains a final "***", the listener will match any event name with identical prefix (without the "***").
   listener - The listener to add.
   authmode - If true, check if the source of event is authorized as specified by the MIDlet's access authorization attributes. Authorized sources are allowed to notify the listener. If false, events irrespective of source are allowed to notify the listener.
   value - The event value for which the listener should be notified.

Throws:
   NullPointerException - If listener or event is null
   IllegalArgumentException - If the named event is a system event and is known not to be a boolean event.
   SecurityException - If the application does not have EventPermission with the name equal the event name and action equals "read".

See Also: EventData, addEventListener(String, EventDataListener, boolean)

addEventListener

public void addEventListener(String event,
   EventDataListener listener,
   boolean authmode,
   double low,
   double high)
Add an event listener to be notified of an event represented by double values. The listener will be notified when the value falls within the range between the low and high values (inclusive). A listener may be added repeatedly using different ranges of values but will be notified only once for any single change. The listener will be called immediately for every known event that matches the requested name, value, and authorization mode.

If the listener is removed using removeEventListener then it will be removed for all ranges and will no longer be notified.

Parameters:
- **event** - The event(s) for which the listener will be added; if event contains a final "***", the listener will match any event name with identical prefix (without the "***").
- **listener** - The listener to add.
- **authmode** - If true, check if the source of event is authorized as specified by the MIDlet's access authorization attributes. Authorized sources are allowed to notify the listener. If false, events irrespective of source are allowed to notify the listener.
- **low** - The low end of the range of interest.
- **high** - The high end of the range of interest.

Throws:
- NullPointerException - If listener or event is null.
- IllegalArgumentException - If high is less than low or if either low or high is NaN or outside the valid range for the event or if the named event is a system event and is known not to be a double event.
- SecurityException - If the application does not have EventPermission with the name equal the event name and action equals "read".

See Also: EventData, removeEventListener(String,EventDataListener)

### addEventListener

```java
public void addEventListener(String event,
    EventDataListener listener,
    boolean authmode,
    long low,
    long high)
```

Add an event listener to be notified of an event represented by int or long values. The listener will be notified when the value falls within the range between the low and high values (inclusive). A listener may be added repeatedly using different ranges of values but will be notified only once for any single event change. The listener will be called immediately for every known event that matches the requested name, value, and authorization mode.

If the listener is removed using removeEventListener then it will be removed for all ranges and will no longer be notified.

Parameters:
- **event** - The event(s) for which the listener will be added; if event contains a final "***", the listener will match any event name with identical prefix (without the "***").
- **listener** - The listener to add.
- **authmode** - If true, check if the source of event is authorized as specified by the MIDlet's access authorization attributes. Authorized sources are allowed to notify the listener. If false, events irrespective of source are allowed to notify the listener.
- **low** - The low end of the range of interest.
- **high** - The high end of the range of interest.

Throws:
- NullPointerException - If listener or event is null.
- IllegalArgumentException - If high is less than low or if either low or high is NaN or outside the valid range of values for the event or if the named event is a system event and is known not to be an int event.
- SecurityException - If the application does not have EventPermission with the name equal the event name and action equals "read".

See Also: EventData, removeEventListener(String,EventDataListener)
**addEventFilterListener**

```java
public void addEventFilterListener(String event,
                                   EventDataListener listener,
                                   boolean authmode,
                                   String[] values)
```

Add an event listener to be notified of an event represented by `String` values. The listener will be notified if the event contains one of the requested values. The listener will be called immediately for every known value that matches the requested name, value, and authorization mode.

If the listener is removed using `removeEventListener` then it will be removed for all values and will no longer be notified.

**Parameters:**
- `event` - The event(s) for which the listener will be added; if `event` contains a final `"*"`, the listener will match any event name with identical prefix (without the `"*"`).
- `listener` - The listener to add.
- `authmode` - If `true`, check if the source of event is authorized as specified by the MIDlet's access authorization attributes. Authorized sources are allowed to notify the listener. If `false`, events irrespective of source are allowed to notify the listener.
- `values` - The event values for which the listener should be notified.

**Throws:**
- `NullPointerException` - If `listener` or `events` is `null`.
- `IllegalArgumentException` - If `values` contain strings that are not valid for the event or if the named event is a system event and is known not to be a `String` event.
- `SecurityException` - If the application does not have `EventPermission` with the name equal the event name and action equals "read".

**See Also:** `EventData`, `removeEventFilterListener(String, EventDataListener)`

**getCurrent**

```java
public javax.microedition.event.EventData getCurrent(String event)
```

Returns an `EventData` object for the requested system event.

**Parameters:**
- `event` - The name for the event.

**Returns:**
- An `EventData` object for the requested system event; or `null` if the requested name is not a known system event as returned from `getSystemEvents()`.

**Throws:**
- `SecurityException` - If the application does not have `EventPermission` with the name equal the event name and action equals "read".
- `NullPointerException` - If `event` is `null`.

**See Also:** `EventData`, `EventPermission`, `getSystemEvents()`

**getInstance**

```java
public static javax.microedition.event.EventManager getInstance()
```

Returns the instance of `EventManager`.

**Returns:**
- The instance of `EventManager`.

**getSystemEvents**

```java
public java.lang.String[] getSystemEvents()
```

Mobile Information Device Profile v3.0 - JSR 271
© Copyright Motorola, Inc. 2002-2009. All Rights Reserved.
Returns a `String` array containing the names of all the system events available on the device. Application specific event names are not listed.

**Returns:**
A `String` array containing the names of all the system events available.

**See Also:** `EventData, getCurrent(String)`

### post

```java
public void post(EventData event,
                  boolean authmode)
        throws java.io.IOException
```

Post an event. The event is posted and the method MUST NOT wait for the event to be delivered. The event MUST be delivered to any recipient that matches the event.

If the name of the event is recognized by the system the value provided MUST match the value type for the event. The time from `java.lang.System.currentTimeMillis` initializes the value of the `EventData.getTimestamp` method. The implementation sets the event source to be the calling MIDlet.

**Parameters:**
- `event`: The event to post. The contents of the `EventData` object are the value for the event.
- `authmode`: If `true`, the event MUST only be delivered to authorized recipients. If `false`, the events MUST be delivered to any matching recipient.

**Throws:**
- `NullPointerException`: If `event` is `null`.
- `IllegalArgumentException`: If `event` names an event for which the type is known to the system and the value provided does not match.
- `SecurityException`: If the application does not have the `EventPermission` with the name equal the event name and action equals "post"; or if the event name is one of those returned by `getSystemEvents()` and the application does not have the `EventPermission` with the name equal the event name and action equals "postsystem".
- `IOException`: If there are insufficient resources to retain the `EventData` until it can be delivered to all listeners. Refer to `EventData` for the minimum size the MUST be supported.

**See Also:** `EventData, EventPermission`

### registerApplication

```java
public void registerApplication(String event,
                                String application,
                                boolean authmode)
        throws java.lang.ClassNotFoundException,
                java.io.IOException
```

Register an application to be launched on an event. An application can register multiple times with different events and values with each registration accumulating; duplicate registrations with the same input parameters will have no effect. The application MUST NOT be launched if it is already running.

If the application is unregistered using `unregisterApplication` then it will no longer be launched in response to the event.

Note: Care should be taken when registering applications to launch on events that change often as frequent attempts to launch the application may negatively impact performance.

**Parameters:**
- `event`: The event(s) for which the application is to be registered; if `event` contains a final "+", the listener will match any event name with identical prefix (without the "+").
application - The fully qualified class name of the application being registered. For MIDP, the named application class MUST be registered in the JAD or JAR Manifest for the MIDlet suite of the calling MIDlet with a MIDlet-<n> application attribute, and MUST extend javax.microedition.midlet.MIDlet.

authmode - If true, check if the source of event is authorized as specified by the MIDlet's access authorization attributes. Authorized sources are allowed to launch this application. If false, events irrespective of source are allowed to launch this application.

Throws:
NullPointerException - If event or application is null.
IllegalArgumentException - If application does not refer to a valid application for the current profile. For MIDP, the named application class MUST be registered in the JAD or JAR Manifest for the MIDlet suite of the calling MIDlet with a MIDlet-<n> application attribute, and MUST extend javax.microedition.midlet.MIDlet.
SecurityException - If the application does not have EventPermission with the name equal the event name and action equals "register".
ClassNotFoundException - If the application class cannot be found
IOException - If there are insufficient resources to perform the registration.

See Also: EventPermission, unregisterApplication(String, String)

```
javax.microedition.event.EventManager

public void registerApplication(String event,
String application,
boolean authmode,
boolean value)
throws java.lang.ClassNotFoundException,
java.io.IOException

Register an application to be launched in response to an event represented by a boolean values. The application will be launched if the event matches the requested name, value, and authorization mode. An application can register multiple times with different events and values with each registration accumulating; duplicate registrations with the same input parameters will have no effect. The application MUST be launched only once for any single event change. The application MUST NOT be launched if it is already running.

If the application is unregistered using unregisterApplication then it will no longer be launched in response to the event.

Note: Care should be taken when registering applications to launch on events that change often as frequent attempts to launch the application may negatively impact performance.

Parameters:

event - The name of a boolean event for which the application is to be registered; if event contains a final "*", the listener will match any event name with identical prefix (without the "*").
application - The fully qualified class name of the application being registered. For MIDP, the named application class MUST be registered in the JAD or JAR Manifest for the MIDlet suite of the calling MIDlet with a MIDlet-<n> application attribute, and MUST extend javax.microedition.midlet.MIDlet.
authmode - If true, check if the source of event is authorized as specified by the MIDlet's access authorization attributes. Authorized sources are allowed to launch this application. If false, events irrespective of source are allowed to launch this application.
value - The event value for which the application should be launched.

Throws:
NullPointerException - If event or application is null.
IllegalArgumentException - If application does not refer to a valid application for the current profile. For MIDP, the named application class MUST be registered in the JAD or JAR Manifest for the MIDlet suite of the calling MIDlet with a MIDlet-<n> application attribute, and MUST extend javax.microedition.midlet.MIDlet.
IllegalArgumentException - If the named event is a system event and is known not to be a boolean event.
SecurityException - If the application does not have EventPermission with the name equal the event name and action equals "register".
ClassNotFoundException - If the application class cannot be found
IOException - If there are insufficient resources to perform the registration.
```
Register an application to be launched in response to an event represented by a double value. The application will be launched if the event value falls within the range between the low and high values (inclusive). An application can register multiple times with different events and values with each registration accumulating; duplicate registrations with the same input parameters will have no effect. The application MUST be launched only once for any single event change. The application MUST NOT be launched if it is already running.

If the application is unregistered using unregisterApplication it will be unregistered for all values and will no longer be launched in response to the event.

Note: Care should be taken when registering applications to launch on events that change often as frequent attempts to launch the application may negatively impact performance.

**Parameters:**
- event - The name of the event to which the application is to be registered; if event contains a final "*", the listener will match any event name with identical prefix (without the "*").
- application - The fully qualified class name of the application being registered. For MIDP, the named application class MUST be registered in the JAD or JAR Manifest for the MIDlet suite of the calling MIDlet with a MIDlet-<n> application attribute, and MUST extend javax.microedition.midlet.MIDlet.
- authmode - If true, check if the source of event is authorized as specified by the MIDlet's access authorization attributes. Authorized sources are allowed to launch this application. If false, events irrespective of source are allowed to launch this application.
- low - The low end of the range of interest.
- high - The high end of the range of interest.

**Throws:**
- NullPointerException - If event or application is null.
- IllegalArgumentException - If application does not refer to a valid application for the current profile. For MIDP, the named application class MUST be registered in the JAD or JAR Manifest for the MIDlet suite of the calling MIDlet with a MIDlet-<n> application attribute, and MUST extend javax.microedition.midlet.MIDlet.
- IllegalArgumentException - If the named event is a system event and is known not to be a double event.
- IllegalArgumentException - If high is less than low or if either low or high is NaN or is outside the valid range of values for the event.
- IllegalArgumentException - If application does not refer to a valid application for the current profile. For MIDP, the named application class MUST be registered in the descriptor file or the JAR Manifest with a MIDlet-<n> record and extends javax.microedition.midlet.MIDlet.
- SecurityException - If the application does not have EventPermission with the name equal the event name and action equals "register".
- ClassNotFoundException - If the application class cannot be found
- IOException - If there are insufficient resources to perform the registration.

**See Also:** EventPermission, unregisterApplication(String, String)
Register an application to be launched in response to an event represented by an `int` or `long` value. The application will be launched if the event value falls within the range between the low and high values (inclusive). An application can register multiple times with different events and values with each registration accumulating; duplicate registrations with the same input parameters will have no effect. The application MUST be launched only once for any single event change. The application MUST NOT be launched if it is already running.

If the application is unregistered using `unregisterApplication` it will be unregistered for all values and will no longer be launched in response to the event.

Note: Care should be taken when registering applications to launch on events that change often as frequent attempts to launch the application may negatively impact performance.

**Parameters:**
- `event` - The name of the event for which the application is to be registered; if `event` contains a final "*", the listener will match any event name with identical prefix (without the "*").
- `application` - The fully qualified class name of the application being registered. For MIDP, the named application class MUST be registered in the JAD or JAR Manifest for the MIDlet suite of the calling MIDlet with a MIDlet-<n> application attribute, and MUST extend `javax.microedition.midlet.MIDlet`.
- `authmode` - If true, check if the source of event is authorized as specified by the MIDlet's access authorization attributes. Authorized sources are allowed to launch this application. If false, events irrespective of source are allowed to launch this application.
- `low` - The low end of the range of interest.
- `high` - The high end of the range of interest.

**Throws:**
- `NullPointerException` - If `event` or `application` is null.
- `IllegalArgumentException` - If `high` is less than `low` or if either `low` or `high` is outside the valid range of values for the event.
- `IllegalStateException` - If application does not refer to a valid application for the current profile. For MIDP, the named application class MUST be registered in the JAD or JAR Manifest for the MIDlet suite of the calling MIDlet with a MIDlet-<n> application attribute, and MUST extend `javax.microedition.midlet.MIDlet`.
- `SecurityException` - If the application does not have `EventPermission` with the name equal the event name and action equals "register".
- `ClassNotFoundException` - If the application class cannot be found.
- `IOException` - If there are insufficient resources to perform the registration.

**See Also:** `EventPermission`, `unregisterApplication(String, String)`

```
public void registerApplication(String event,
   String application,
   boolean authmode,
   long low,
   long high)
throws java.lang.ClassNotFoundException,
   java.io.IOException
```

```
public void registerApplication(String event,
   String application,
   boolean authmode,
   String[] values)
throws java.lang.ClassNotFoundException,
   java.io.IOException
```
Register an application to be launched in response to a event represented by a `String` value. The application will be launched if the event value changes to one of the values specified in the `values` parameter. An application can register multiple times with different events and values with each registration accumulating; duplicate registrations with the same input parameters will have no effect. The application MUST be launched only once for any single event change. The application MUST NOT be launched if it is already running.

If the application is unregistered using `unregisterApplication` it will be unregistered for all values and will no longer be launched in response to the event.

**Note:** Care should be taken when registering applications to launch on events that change often as frequent attempts to launch the application may negatively impact performance.

**Parameters:**
- `event` - The name of the event for which the application is to be registered; if `event` contains a final "*", the listener will match any event name with identical prefix (without the "*").
- `application` - The fully qualified class name of the application being registered. For MIDP, the named `application` class MUST be registered in the JAD or JAR Manifest for the MIDlet suite of the calling MIDlet with a MIDlet-<n> application attribute, and MUST extend `javax.microedition.midlet.MIDlet`.
- `authmode` - If true, check if the source of event is authorized as specified by the MIDlet's access authorization attributes. Authorized sources are allowed to launch this application. If false, events irrespective of source are allowed to launch this application.
- `values` - The event values for which the application is to be launched.

**Throws:**
- `NullPointerException` - If `event`, `values` or `application` is null or if `values` contains values that are null.
- `IllegalArgumentException` - If the named event is a system event and is known not to be an event with a `String` value or if `values` contains values that are not valid for the event.
- `NullPointerException` - If `application` does not refer to a valid application for the current profile. For MIDP, the named `application` class MUST be registered in the JAD or JAR Manifest for the MIDlet suite of the calling MIDlet with a MIDlet-<n> application attribute, and MUST extend `javax.microedition.midlet.MIDlet`.
- `SecurityException` - If the application does not have `EventPermission` with the name equal the event name and action equals "register".
- `ClassNotFoundException` - If the application class cannot be found.
- `IOException` - If there are insufficient resources to perform the registration.

**See Also:** `EventData`, `addEventListener(String, EventDataListener, boolean)`

### removeEventListener

```java
public void removeEventListener(String event,
        EventDataListener listener)
```

Removes an event listener for the named event(s) including wildcards. For example, to remove notifications to the listener for all events call `removeEventListener("*", listener)`. The listener will no longer be notified of any matching event. If the listener was added for multiple conditions then all of the conditions will be removed for the event and listener. If `listener` had not previously been added as a listener to the event then the request is ignored.

**Parameters:**
- `event` - The event(s) for which the listener will be removed; if `event` contains a final "*", the listener will match any event name with identical prefix (without the "*").
- `listener` - The listener to remove.

**Throws:**
- `NullPointerException` - If `listener` or `event` is null.

**See Also:** `EventData`, `addEventListener(String, EventDataListener, boolean)`

### unregisterApplication

```java
public void unregisterApplication(String event,
        String application)
```


javax.microedition.event.EventManager

Unregister an application from launching in response to an event. All registrations for the event and application will be removed. Static registrations made during installation are also removed using this API. The request will be ignored if the application had not previously been registered for the event.

**Parameters:**
- event - The name of the event from which the application is to be unregistered; if `event` contains a final "*", the listener will match any event name with identical prefix (without the "*").
- application - The fully qualified class name of the application to be unregistered.

**Throws:**
- NullPointerException - If `application` or `event` is null.
- IllegalArgumentException - If `application` does not refer to a valid application for the current profile. (e.g. MIDlet, etc.)

**See Also:** EventPermission, registerApplication(String, String, boolean)
javax.microedition.event

EventPermission

Declaration

public final class EventPermission extends Permission

Object

---Permission

---javax.microedition.event.EventPermission

Description

EventPermission allows access to system events. An EventPermission objects contains a name (also referred to as a "target name") and actions.

The target name is the name of the event ("BATTERY_LEVEL", "com.MyCompany.MyEvent", etc). The naming convention follows the hierarchical property naming convention and are explained in the package description. An asterisk MAY appear at the end of the event name, following a ".", or by itself, to signify a wildcard match. For example: "com.MyCompany."" or "" is valid, but ""MyCompany" or ""a"b" is not valid.

The actions to be granted are passed to the constructor in a String containing a list of one or more comma-separated keywords. The possible keywords are "post", "postsystem", "read", and "register". Their meaning is defined as follows:

- **post**
  Permission to post an event. Allows the EventManager.post method to post an event.

- **postsystem**
  Permission to post a system event. Allows the EventManager.post method to post an event that is returned by EventManager.getSystemEvents.

- **read**
  Permission to read an event. Allows the EventManager.getCurrent method return the current event.

- **register**
  Permission to register applications to launch in response to events. Allows the EventManager.registerApplication() method to register for the named event.

The actions string is converted to lowercase before processing.

Care should be taken before granting application permission to access certain events. For example, granting permission to post events could allow a badly behaving application to interfere with the operation of the device.

Since: MIDP 3.0

Constructor Summary

| public EventPermission(String event, String actions) |
| Creates a new EventPermission object with the specified name and actions. |

Method Summary
**javax.microedition.event.EventPermission**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>boolean equals(Object object)</code></td>
<td>Checks if another object is &quot;equal&quot; to this one.</td>
</tr>
<tr>
<td><code>java.lang.String getActions()</code></td>
<td>Gets the canonical string representing the actions.</td>
</tr>
<tr>
<td><code>int hashCode()</code></td>
<td>Gets the hash code value for this object.</td>
</tr>
<tr>
<td><code>boolean implies(Permission p)</code></td>
<td>Checks if the specified permission is &quot;implied&quot; by this object.</td>
</tr>
</tbody>
</table>

**Methods inherited from class:** Permission

- `getName`, `newPermissionCollection`, `toString`

**Methods inherited from class:** Object

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

**Constructors**

**EventPermission**

```java
public EventPermission(String event,
                        String actions)
```

Creates a new EventPermission object with the specified name and actions.

**Parameters:**
- `event` - The name of the event for which to grant permission.
- `actions` - Valid actions are "post", "postsystem", "read", and "register".

**Throws:**
- `NullPointerException` - if `event` or `actions` is null.
- `IllegalArgumentException` - if `actions` is empty or includes any string other than those allowed.

**See Also:** EventManager

**Methods**

**equals**

```java
public boolean equals(Object object)
```

Checks if another object is "equal" to this one.

**Parameters:**
- `object` - an object to compare

**Returns:**
- `true` if the objects are of the same type and the event and action are equal.

**getActions**

```java
public java.lang.String getActions()
```
Gets the canonical string representing the actions. This method always returns present actions in the following order: "post", "postsystem", "read", and "register". For example, if this EventPermission object allows both post and read actions, a call to getActions will return the string "post,read".

**Returns:**
the canonical string of actions.

---

**hashCode**

```java
public int hashCode()
```

Gets the hash code value for this object.

**Returns:**
the hashcode

---

**implies**

```java
public boolean implies(Permission p)
```

Checks if the specified permission is "implied" by this object. Returns true if:

- p's class is the same as this Object's class, and
- p's name equals or (in the case of wildcards) is implied by this object's name.
- p's actions are a subset of this object's actions

**Parameters:**
p - the permission to check against.

**Returns:**
true if the passed permission is equal to or implied by this permission, false otherwise.
Appendix A - Application Attributes

This appendix specifies all Application Attributes and the allowed locations for each attribute (JAD, Manifest, or both), along with information about the specification version.

In Table A-1 below, <locale> represents a valid value of the microedition.locale system property.

Status:
● = MUST appear
○ = MAY appear
in manifest and/or JAD file as indicated in the respective columns.

Manifest and JAD:
● = can be placed in this location
– = not allowed in this location.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Status</th>
<th>Manifest</th>
<th>JAD</th>
<th>Since</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDlet-Name</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>1.0</td>
</tr>
<tr>
<td>MIDlet-Version</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>1.0</td>
</tr>
<tr>
<td>MIDlet-Vendor</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>1.0</td>
</tr>
<tr>
<td>MIDlet-&lt;n&gt;</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>1.0</td>
</tr>
<tr>
<td>MicroEdition-Profile</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>1.0</td>
</tr>
<tr>
<td>MicroEdition-Configuration</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>1.0</td>
</tr>
<tr>
<td>LIBlet-Name</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>LIBlet-Version</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>LIBlet-Vendor</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Jar-URL</td>
<td>●</td>
<td>–</td>
<td>●</td>
<td>1.0</td>
</tr>
<tr>
<td>MIDlet-Jar-Size</td>
<td>●</td>
<td>–</td>
<td>●</td>
<td>1.0</td>
</tr>
<tr>
<td>LIBlet-Jar-URL</td>
<td>●</td>
<td>–</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>LIBlet-Jar-Size</td>
<td>●</td>
<td>–</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Name-&lt;locale&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Description</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>1.0</td>
</tr>
<tr>
<td>MIDlet-Description-&lt;locale&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Info-URL</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>1.0</td>
</tr>
<tr>
<td>MIDlet-&lt;n&gt;-&lt;locale&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Update-URL</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Data-Size</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>1.0</td>
</tr>
<tr>
<td>MIDlet-Minimum-Canvas-Size</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
</tbody>
</table>
### Table A-1: Application Descriptor Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Status</th>
<th>Manifest</th>
<th>JAD</th>
<th>Since</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDlet-Maximum-Canvas-Size</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Push-&lt;n&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>2.0</td>
</tr>
<tr>
<td>MIDlet-Install-Notify</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>2.0</td>
</tr>
<tr>
<td>LIBlet-Install-Notify</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Required-IP-Version</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-&lt;n&gt;-Type</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-&lt;n&gt;-Category</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-&lt;n&gt;-UserDenied</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-UserDenied</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Operator-Allowed</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>LIBlet-Description</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>LIBlet-Description-&lt;locale&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>LIBlet-NonShared-Data-Size</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>LIBlet-Shared-Data-Size</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Dependency-&lt;n&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>LIBlet-Dependency-&lt;n&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Icon</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Icon-&lt;locale&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Scalable-Icon</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Scalable-Icon-&lt;locale&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-&lt;n&gt;-Scalable-Icon</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-&lt;n&gt;-Scalable-Icon-&lt;locale&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Font</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Splash-Screen-Image</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Event-Launch-&lt;n&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Delete-Notify</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>2.0</td>
</tr>
<tr>
<td>MIDlet-Delete-Confirm</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>2.0</td>
</tr>
<tr>
<td>MIDlet-Delete-Confirm-&lt;locale&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Persistent-Data-URL-&lt;n&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>LIBlet-Persistent-Data-URL-&lt;n&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>LIBlet-Icon</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>LIBlet-Delete-Notify</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Dependency-JAD-URL-&lt;n&gt;</td>
<td>○</td>
<td>–</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>LIBlet-Dependency-JAD-URL-&lt;n&gt;</td>
<td>○</td>
<td>–</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>LIBlet-Font</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Permission-&lt;n&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Permission-Opt-&lt;n&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Access-Auth-Type-&lt;n&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Access-Auth-Cert-&lt;n&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Permissions</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>2.0</td>
</tr>
<tr>
<td>MIDlet-Permissions-Opt</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>2.0</td>
</tr>
</tbody>
</table>
### Table A-1: Application Descriptor Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Status</th>
<th>Manifest</th>
<th>JAD</th>
<th>Since</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBlet-Permissions</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>LIBlet-Permissions-Opt</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>LIBlet-Permission-&lt;n&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>LIBlet-Permission-Opt-&lt;n&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Dependency-Jar-SHA1-&lt;n&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>LIBlet-Dependency-Jar-SHA1-&lt;n&gt;</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>LIBlet-Jar-SHA1</td>
<td>●</td>
<td>–</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Certificate-&lt;n&gt;-&lt;m&gt;</td>
<td>○</td>
<td>–</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Jar-RSA-SHA1-&lt;n&gt;</td>
<td>○</td>
<td>–</td>
<td>●</td>
<td>3.0</td>
</tr>
<tr>
<td>MIDlet-Jar-RSA-SHA1</td>
<td>○</td>
<td>–</td>
<td>●</td>
<td>2.0</td>
</tr>
<tr>
<td>MIDlet-Profile-Request</td>
<td>○</td>
<td>–</td>
<td>●</td>
<td>3.0</td>
</tr>
</tbody>
</table>
Appendix B - MIDP 3.0 Example Domain Policies

The Security Policy Chapter defines a security policy for APIs in this specification. For each domain the policy specifies the Permissions that are user and allowed and groups them into Function Groups to facilitate the user prompting. The examples below use the External Domain Policy Format to represent each of the Operator, Manufacturer, Identified, and Unidentified domains. These are examples only and are not a complete security policy.

Operator Domain Policy Example

The Operator policy does not restrict any behavior and therefore includes all permissions.

```java
domain Operator;
grant allowed "network" {
  permission javax.microedition.io.HttpProtocolPermission "http://*::*";
  permission javax.microedition.io.HttpsProtocolPermission "https://*::*";
  permission javax.microedition.io.DatagramProtocolPermission "datagram://*::*";
  permission javax.microedition.io.DatagramProtocolPermission "datagram://:*";
  permission javax.microedition.io.SocketProtocolPermission "socket://*::*";
  permission javax.microedition.io.SocketProtocolPermission "socket://:*";
  permission javax.microedition.io.SSLProtocolPermission "ssl://**.*";
}
grant allowed "misc" {
  permission javax.microedition.io.PushRegistryPermission "*" "static,dynamic,alarm";
  permission javax.microedition.io.CommProtocolPermission "comm::*";
  permission javax.microedition.media.PlayerPermission "*" "record,snapshot";
  permission java.util.PropertyPermission "*" "read";
  permission javax.microedition.event.EventPermission "*" "read,register,post,postsystem";
  permission javax.microedition.midlet.ActionsDeniedPermission;
  permission javax.microedition.midlet.AutoStartPermission;
}
```

Manufacturer Domain Policy Example

The Manufacturer policy does not restrict any behavior and therefore includes all permissions.

```java
domain Manufacturer;
grant allowed {
  permission javax.microedition.io.HttpProtocolPermission "http://*";
  permission javax.microedition.io.HttpsProtocolPermission "https://*::*";
  permission javax.microedition.io.DatagramProtocolPermission "datagram://*::*";
  permission javax.microedition.io.DatagramProtocolPermission "datagram://:*";
  permission javax.microedition.io.SocketProtocolPermission "socket://*::*";
  permission javax.microedition.io.SocketProtocolPermission "socket://:*";
  permission javax.microedition.io.SSLProtocolPermission "ssl://*::*";
}
```
Identified Third Party Domain Policy Example

This is an example of the Identified Third Party Domain function groups and permissions; it does not contain permissions for APIs outside of MIDP 3.0.

```
domain IdentifiedThirdParty;
grant user "NetAccess" session,blanket,oneshot,no {
    permission javax.microedition.io.HttpProtocolPermission "http://*";
    permission javax.microedition.io.HttpsProtocolPermission "https://*";
}
grant user "LowLevelNetAccess" session,blanket,oneshot,no {
    permission javax.microedition.io.DatagramProtocolPermission "datagram://*";
    permission javax.microedition.io.DatagramProtocolPermission "datagram://";
    permission javax.microedition.io.SocketProtocolPermission "socket://*";
    permission javax.microedition.io.SocketProtocolPermission "socket://";
    permission javax.microedition.io.SSLProtocolPermission "ssl://*";
}
grant user "ApplicationAutoInvocation" oneshot,blanket,session,no {
    permission javax.microedition.io.PushRegistryPermission "static,dynamic,alarms";
}
grant user "LocalConnectivity" session,blanket,oneshot,no {
    permission javax.microedition.io.CommProtocolPermission "comm:*";
}
grant user "MultimediaRecording" session,blanket,oneshot,no {
    permission javax.microedition.media.PlayerPermission "record,snapshot";
    permission javax.microedition.media.PlayerPermission "record";
    permission javax.microedition.media.PlayerPermission "snapshot";
}
grant allowed {
    permission java.util.PropertyPermission "microedition.*" "read";
    permission java.microedition.event.EventPermission "read,register,post";
}
```

Unidentified Third Party Domain Policy Example

This is an example of the Unidentified Third Party Domain function groups and permissions. It is similar to the Identified Third Party domain except that it does not allow the Blanket user interaction mode. This example does not contain permissions for APIs outside of MIDP 3.0.

```
domain UnidentifiedThirdParty;
grant user "NetAccess" oneshot,session,no {
    permission javax.microedition.io.HttpProtocolPermission "http://*";
    permission javax.microedition.io.HttpsProtocolPermission "https://*";
}
grant user "LowLevelNetAccess" oneshot,session,no {
    permission javax.microedition.io.DatagramProtocolPermission "datagram://*";
    permission javax.microedition.io.DatagramProtocolPermission "datagram://";
    permission javax.microedition.io.SocketProtocolPermission "socket://*";
    permission javax.microedition.io.SocketProtocolPermission "socket://";
}
```
Appendix B - MIDP 3.0 Example Domain Policies

```java
permission javax.microedition.io.SSLProtocolPermission "ssl://*";
grant user "ApplicationAutoInvocation" oneshot,session,no {
    permission javax.microedition.io.PushRegistryPermission "*" "static,dynamic,alarms";
}
grant user "LocalConnectivity" session,blanket,oneshot,no {
    permission javax.microedition.io.CommProtocolPermission "comm:*";
}
grant user "MultimediaRecording" session,oneshot,no {
    permission javax.microedition.media.PlayerPermission "*" "record";
    permission javax.microedition.media.PlayerPermission "*" "snapshot";
}
grant allowed {
    permission java.util.PropertyPermission "microedition.*" "read"
    permission javax.microedition.event.EventPermission "*" "read,register,post"
}
```
Appendix C - Application Level Access Authorization Examples

This appendix provides examples of how Application Level Access Authorization can be used.

C.1 Example 1 - Accessing Shared Record Store

There are three MIDlet Suites installed on a device: RMSOwner and RMSUser1 are installed in the Manufacturer Protection Domain whereas RMSUser2 is installed in the Operator Protection Domain. RMSOwner grants access to its record store to any MIDlet Suite that is installed in the same protection domain.

RMSOwner creates a shared record store using RecordStore.openRecordStore with authmode parameter set to RecordStore.AUTHMODE_APPLEVEL. In this example, RMSOwner grants RMSUser1 access to its shared resources as application access authorization is verified successfully for RMSUser1. When RMSUser2 tries to gain access to RMSOwner's shared resources, application access authorization verification fails and SecurityException is thrown.

The application attributes for the RMSOwner MIDlet are:

| MIDlet-1 : RMSOwner,,RMSOwner |
| MIDlet-Access-Auth-Type-1 : domain=SELF;vendor=ANY;signer=ANY |

Figure C-1 below shows how each of the MIDlets in this example would either be granted or denied access to the shared record store.

| Figure C-1 : Application Level Access Authorization Example 1 |
A LIBlet bound to either RMSUser1 or RMSUser2 would acquire the same rights to access RMSOwner shared record store as the MIDlet it is bound to.

Same behavior applies to accessing provisioned record stores that have authmode set to RecordStore.AUTHMODE_APPLEVEL. The application attributes for RMSOwner in this case would include the an application attribute pointing to the URL for the provisioned record store:

```
MIDlet-1 : RMSOwner,,RMSOwner
MIDlet-Access-Auth-Type-1 : domain=SELF;vendor=ANY;signer=ANY
MIDlet-Persistent-Data-URL-1 : http://coolcompanyinc.com/data/cooldata.rms
```

C.2 Example 2 - IMC

IMCServer1 is a MIDlet Suite that provides a Service using the IMC protocol. This service is available only to authorized MIDlet Suites. IMCServer1 sets that up by setting the authmode parameter to true while creating an IMCServerConnection. IMCServer1 grants access to any MIDletSuite that belongs to vendor "CoolCompany". Since a rogue MIDlet installed in the Unidentified Third Party Protection Domain can claim to belong to the "CoolCompany" vendor and gain access to the service, it is recommended that MIDlet Suites use the vendor attribute in conjunction with the domain and/or signer values (see Example 3).

IMCClient1A is a MIDlet Suite that has been published by vendor "CoolCompany" while IMCClient1B has been published by vendor "OtherCompany".

The application attributes for the IMCServer1 are:

```
MIDlet-1 : IMCServer1,,IMCServer1
MIDlet-Access-Auth-Type-1 : domain=ANY;vendor="CoolCompany";signer=ANY
```

The application attributes for the IMCClient1A are:

```
MIDlet-1 : IMCClient1A,,IMCClient1A
MIDlet-Vendor : CoolCompany
```
The application attributes for the IMCClient1B are:

```plaintext
MIDlet-1 : IMCClient1B, ,IMCClient1B
MIDlet-Vendor : OtherCompany
```

Figure C-2 below shows how IMCClient1A and IMCClient1B would either be granted or denied access to the server MIDlet IMCServer1.

![Diagram](image)

**C.3 Example 3 - IMC**

In this example, both the IMCServer2 and IMCClient2 request access authorization while using IMC based services.

IMCServer2 grants access to any MIDlet suite that belongs to vendor "CoolCompany" and is signed by signer "Good Certificate".

IMCClient2 grants access to any MIDlet that is installed in the same protection domain.

Both MIDlet Suites are installed in the Identified Third Party Protection Domain.

When IMCClient2 makes the `Connector.open` call to the IMCServer setup by IMCServer2, the MIDP Implementation checks if both suites are authorized to access each other and only then provides a valid IMCConnection. If either of the MIDlet Suites were not authorized, a `SecurityException` would have been thrown.

The application attributes for the IMCServer2, installed into the Identified Third Party protection domain, are:

```plaintext
MIDlet-1 : IMCServer2,,IMCServer2
MIDlet-Access-Auth-Type-1 : domain=ANY;vendor="CoolCompany";signer=goodcert
MIDlet-Access-Auth-Cert-1 : goodcert <base 64 encoding of Good Certificate>
```
Appendix C - Application Level Access Authorization Examples

The application attributes for the IMCClient2, also installed in the Identified Third Party security domain, are:

<table>
<thead>
<tr>
<th>MIDlet-1</th>
<th>IMCClient2,,IMCClient2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDlet-Vendor</td>
<td>CoolCompany</td>
</tr>
<tr>
<td>MIDlet-Certificate-1-1</td>
<td>&lt;base64 encoding of &quot;Good Certificate&quot;&gt;</td>
</tr>
<tr>
<td>MIDlet-JAR-RSA-SHA1-1</td>
<td>&lt;base64 encoding of the JAR signature&gt;</td>
</tr>
<tr>
<td>MIDlet-Access-Auth-Type-1</td>
<td>domain=SELF;vendor=ANY;signer=ANY</td>
</tr>
</tbody>
</table>

Figure C-3 below shows how IMCServer2 and IMCClient2 are able enabled to communicate with one another via IMC:

![Figure C-3: Application Level Access Authorization Example 3](image)

C.4 Example 4 - Events

In this example, EVTListnr1 and EVTListnr2 are interested in an application event posted by EVTSource. EVTListnr1 and EVTListnr2 register listeners to listen to it.

EVTSource grants access to any MIDlet suite that is either within the same domain, or belongs to vendor "CoolCompany" and is signed by "Good Certificate".

EVTListnr1 is not authorized to receive events from EVTSource because its credentials do not satisfy any of the requirements set by EVTSource in MIDlet-Access-Auth-Type-1 and MIDlet-Access-Auth-Type-2 attributes, so the EVTListnr1:handleEvent method is not called. Since both EVTSource and EVTListnr2 both fulfill each others application access authorization requirements, they are authorized to access each other and a callback is made to EVTListnr2:handleEvent method.

The application attributes for the EVTSource, installed into the Operator protection domain, are:

<table>
<thead>
<tr>
<th>MIDlet-1</th>
<th>EVTSource,,EVTSource</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDlet-Vendor</td>
<td>IndustryLeader</td>
</tr>
<tr>
<td>MIDlet-Access-Auth-Type-1</td>
<td>domain=SELF;vendor=ANY;signer=ANY</td>
</tr>
<tr>
<td>MIDlet-Access-Auth-Type-2</td>
<td>domain=ANY;vendor=&quot;CoolCompany&quot;;signer=goodcert</td>
</tr>
<tr>
<td>MIDlet-Access-Cert-1</td>
<td>goodcert &lt;base 64 encoding of &quot;Good Certificate&quot;&gt;</td>
</tr>
</tbody>
</table>

The application attributes for the EVTListnr1, installed into the Manufacturer protection domain, are:
Appendix C - Application Level Access Authorization Examples

The application attributes for the EVTListnr2, installed into the Identified Third Party protection domain, are:

```
MIDlet-1 : EVTListnr2,,EVTListnr2
MIDlet-Vendor : CoolCompany
MIDlet-Certificate-1-1 : <base64 encoding of "Good Certificate">
MIDlet-JAR-RSA-SHA1-1 : <base64 encoding of the JAR signature>
```

Access authorization for MIDlet suites that request application launch on events has a behavior similar to what is described in this example. Access authorization is requested by setting the authmode attribute to true irrespective of whether the application attribute MIDlet-Event-Launch-<n> or EventManager.registerApplication is used.
### 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.

<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>

7. Return type of a method or declared type of a field. Blank for constructors.
8. Name of the constructor, field or method. Nested classes are listed in 1, not here.
### Almanac

#### ActionsDeniedPermission

<table>
<thead>
<tr>
<th>Method</th>
<th>Declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 3.0</td>
<td>ActionsDeniedPermission()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>boolean implies(Permission permission)</td>
</tr>
</tbody>
</table>

#### Alert

<table>
<thead>
<tr>
<th>Method</th>
<th>Declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 1.0</td>
<td>void addCommand(Command cmd)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>Alert(String title)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>Alert(String title,String alertText,Image alertImage,AlertType alertType)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>Command DISMISS_COMMAND</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int FOREVER</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int getDefaultTimeout()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int getHeight()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>Image getImage()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>Gauge getIndicator()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>String getString()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int getTimeout()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>AlertType getType()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int getWidth()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void removeCommand(Command cmd)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void setCommandListener(CommandListener l)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void setImage(Image img)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>void setIndicator(Gauge indicator)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void setString(String str)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void setTimeout(int time)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void setType(AlertType type)</td>
</tr>
</tbody>
</table>
### AlertType

<table>
<thead>
<tr>
<th>MIDP 1.0</th>
<th>AlertType</th>
<th>ALARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 1.0</td>
<td>AlertType()</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>AlertType</td>
<td>CONFIRMATION</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>AlertType</td>
<td>ERROR</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>AlertType</td>
<td>INFO</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>boolean playSound(Display display)</td>
<td>WARNING</td>
</tr>
</tbody>
</table>

### AnimatedImage

<table>
<thead>
<tr>
<th>MIDP 3.0</th>
<th>Image</th>
<th>getFrame(int index)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
<td>getFrameCount()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
<td>getFrameDelay(int index)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
<td>getLoopCount()</td>
</tr>
</tbody>
</table>

### AutoStartPermission

<table>
<thead>
<tr>
<th>MIDP 3.0</th>
<th>AutoStartPermission()</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 3.0</td>
<td>boolean implies(Permission permission)</td>
</tr>
</tbody>
</table>

### Canvas

<table>
<thead>
<tr>
<th>MIDP 3.0</th>
<th>int ACTIONS_ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 3.0</td>
<td>int ACTIONS_NAVIGATION</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int ACTIONS_NONE</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>Canvas()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int DOWN</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int FIRE</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>int GAME_A</td>
<td></td>
</tr>
<tr>
<td>int GAME_B</td>
<td></td>
</tr>
<tr>
<td>int GAME_C</td>
<td></td>
</tr>
<tr>
<td>int GAME_D</td>
<td></td>
</tr>
<tr>
<td>int getGameAction(int keyCode)</td>
<td></td>
</tr>
<tr>
<td>int getHeight()</td>
<td></td>
</tr>
<tr>
<td>int getKeyCode(int action)</td>
<td></td>
</tr>
<tr>
<td>String getKeyName(int keyCode)</td>
<td></td>
</tr>
<tr>
<td>int[] getSoftkeyLabelCoordinates(int placement)</td>
<td></td>
</tr>
<tr>
<td>int getWidth()</td>
<td></td>
</tr>
<tr>
<td>boolean hasPointerEvents()</td>
<td></td>
</tr>
<tr>
<td>boolean hasPointerMotionEvents()</td>
<td></td>
</tr>
<tr>
<td>boolean hasRepeatEvents()</td>
<td></td>
</tr>
<tr>
<td>void hideNotify()</td>
<td></td>
</tr>
<tr>
<td>boolean isDoubleBuffered()</td>
<td></td>
</tr>
<tr>
<td>int KEY_BACKSPACE</td>
<td></td>
</tr>
<tr>
<td>int KEY_DELETE</td>
<td></td>
</tr>
<tr>
<td>int KEY_DOWN</td>
<td></td>
</tr>
<tr>
<td>int KEY_ENTER</td>
<td></td>
</tr>
<tr>
<td>int KEY_ESCAPE</td>
<td></td>
</tr>
<tr>
<td>int KEY_LEFT</td>
<td></td>
</tr>
<tr>
<td>int KEY_NUM0</td>
<td></td>
</tr>
<tr>
<td>int KEY_NUM1</td>
<td></td>
</tr>
<tr>
<td>int KEY_NUM2</td>
<td></td>
</tr>
<tr>
<td>int KEY_NUM3</td>
<td></td>
</tr>
<tr>
<td>int KEY_NUM4</td>
<td></td>
</tr>
<tr>
<td>int KEY_NUM5</td>
<td></td>
</tr>
<tr>
<td>int KEY_NUM6</td>
<td></td>
</tr>
<tr>
<td>int KEY_NUM7</td>
<td></td>
</tr>
<tr>
<td>int KEY_NUM8</td>
<td></td>
</tr>
<tr>
<td>int KEY_NUM9</td>
<td></td>
</tr>
<tr>
<td>int KEY_POUND</td>
<td></td>
</tr>
</tbody>
</table>
MIDP 1.0  
int KEY_RIGHT
MIDP 1.0  
int KEY_SELECT
MIDP 1.0  
int KEY_SPACE
MIDP 1.0  
int KEY_STAR
MIDP 1.0  
int KEY_TAB
MIDP 1.0  
int KEY_UP
MIDP 1.0  
void keyPressed(int keyCode)
MIDP 1.0  
void keyReleased(int keyCode)
MIDP 1.0  
void keyRepeated(int keyCode)
MIDP 1.0  
int LEFT
MIDP 1.0  
void paint(Graphics g)
MIDP 1.0  
void pointerDragged(int x,int y)
MIDP 1.0  
void pointerPressed(int x,int y)
MIDP 1.0  
void pointerReleased(int x,int y)
MIDP 1.0  
void repaint()
MIDP 1.0  
void repaint(int x,int y,int width,int height)
MIDP 1.0  
void serviceRepaints()
MIDP 2.0  
void setFullScreenMode(boolean mode)
MIDP 3.0  
void setKeyListener(KeyListener listener)
MIDP 3.0  
void setPaintMode(boolean opaque)
MIDP 3.0  
void setRequiredActions(int actionSet)
MIDP 1.0  
void showNotify()
MIDP 2.0  
void sizeChanged(int w,int h)
MIDP 1.0  
int UP

CanvasItem  
javax.microedition.lcdui

Object  
CanvasItem

3.0  *
CanvasItem()
3.0  
int getHeight()
3.0  
Object getParent()
3.0  
int getPositionX()
### Certificate

<table>
<thead>
<tr>
<th>MIDP 2.0</th>
<th>String</th>
<th>getIssuer()</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 2.0</td>
<td>long</td>
<td>getNotAfter()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>long</td>
<td>getNotBefore()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>String</td>
<td>getSerialNumber()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>String</td>
<td>getSigAlgName()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>String</td>
<td>getSubject()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>String</td>
<td>getType()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>String</td>
<td>getVersion()</td>
</tr>
</tbody>
</table>

### CertificateException

<table>
<thead>
<tr>
<th>MIDP 2.0</th>
<th>byte</th>
<th>BAD_EXTENSIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 2.0</td>
<td>byte</td>
<td>BROKEN_CHAIN</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>byte</td>
<td>CERTIFICATE_CHAIN_TOO_LONG</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>CertificateException(Certificate certificate,byte status)</td>
<td></td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>CertificateException(String message,Certificate certificate,byte status)</td>
<td></td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>byte</td>
<td>EXPIRED</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>Certificate</td>
<td>getCertificate()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>byte</td>
<td>getReason()</td>
</tr>
<tr>
<td></td>
<td>byte</td>
<td>INAPPROPRIATE_KEY_USAGE</td>
</tr>
<tr>
<td></td>
<td>byte</td>
<td>MISSING_SIGNATURE</td>
</tr>
<tr>
<td></td>
<td>byte</td>
<td>NOT_YET_VALID</td>
</tr>
<tr>
<td></td>
<td>byte</td>
<td>ROOT_CA_EXPIRED</td>
</tr>
<tr>
<td></td>
<td>byte</td>
<td>SITENAME_MISMATCH</td>
</tr>
<tr>
<td></td>
<td>byte</td>
<td>UNAUTHORIZED_INTERMEDIATE_CA</td>
</tr>
<tr>
<td></td>
<td>byte</td>
<td>UNRECOGNIZED_ISSUER</td>
</tr>
<tr>
<td></td>
<td>byte</td>
<td>UNSUPPORTED_PUBLIC_KEY_TYPE</td>
</tr>
<tr>
<td></td>
<td>byte</td>
<td>UNSUPPORTED_SIGALG</td>
</tr>
<tr>
<td></td>
<td>byte</td>
<td>VERIFICATION_FAILED</td>
</tr>
</tbody>
</table>

### Choice

<table>
<thead>
<tr>
<th>javax.microedition.lcdui</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Choice</td>
<td></td>
</tr>
</tbody>
</table>

| MIDP 1.0 | int  | append(String stringPart,Image imagePart) |
| MIDP 1.0 | void | delete(int elementNum) |
| MIDP 2.0 | void | deleteAll() |
| MIDP 1.0 | int  | EXCLUSIVE |
| MIDP 2.0 | int  | getFitPolicy() |
| MIDP 2.0 | Font | getFont(int elementNum) |
| MIDP 1.0 | Image | getImage(int elementNum) |
| MIDP 1.0 | int  | getSelectedFlags(boolean[] selectedArray_return) |
| MIDP 1.0 | int  | getSelectedIndex() |
| MIDP 1.0 | String | getString(int elementNum) |
| MIDP 1.0 | int  | IMPLICIT |
| MIDP 1.0 | void | insert(int elementNum,String stringPart,Image imagePart) |
| MIDP 3.0 | boolean | isEnabled(int elementNum) |
| MIDP 1.0 | boolean | isSelected(int elementNum) |
| MIDP 1.0 | int  | MULTIPLE |
| MIDP 2.0 | int  | POPUP |
| MIDP 1.0 | void | set(int elementNum,String stringPart,Image imagePart) |
| MIDP 3.0 | void | setEnabled(int elementNum,boolean isEnabled) |
| MIDP 2.0 | void setFitPolicy(int fitPolicy) |
| MIDP 2.0 | void setFont(int elementNum, Font font) |
| MIDP 1.0 | void setSelectedFlags(boolean[] selectedArray) |
| MIDP 1.0 | void setSelectedIndex(int elementNum, boolean selected) |
| MIDP 1.0 | int size() |
| MIDP 2.0 | int TEXT_WRAP_DEFAULT |
| MIDP 2.0 | int TEXT_WRAP_OFF |
| MIDP 2.0 | int TEXT_WRAP_ON |

### ChoiceGroup

<table>
<thead>
<tr>
<th>Object</th>
<th>javax.microedition.lcdui</th>
</tr>
</thead>
<tbody>
<tr>
<td>➔ Item</td>
<td>➔ ChoiceGroup</td>
</tr>
</tbody>
</table>

| MIDP 1.0 | int append(String stringPart, Image imagePart) |
| MIDP 1.0 | ChoiceGroup(String label, int choiceType) |
| MIDP 1.0 | ChoiceGroup(String label, int choiceType, java.lang.String[] stringElements, javax.microedition.lcdui.Image[] imageElements) |
| MIDP 1.0 | void delete(int elementNum) |
| MIDP 1.0 | void deleteAll() |
| MIDP 2.0 | int getFitPolicy() |
| MIDP 2.0 | Font getFont(int elementNum) |
| MIDP 1.0 | Image getImage(int elementNum) |
| MIDP 1.0 | int setSelectedFlags(boolean[] selectedArray_return) |
| MIDP 1.0 | int setSelectedIndex() |
| MIDP 1.0 | String getString(int elementNum) |
| MIDP 1.0 | void insert(int elementNum, String stringPart, Image imagePart) |
| MIDP 3.0 | boolean isEnabled(int elementNum) |
| MIDP 1.0 | boolean isSelected(int elementNum) |
| MIDP 1.0 | void set(int elementNum, String stringPart, Image imagePart) |
| MIDP 3.0 | void setEnabled(int elementNum, boolean isEnabled) |
| MIDP 2.0 | void setFitPolicy(int fitPolicy) |
| MIDP 2.0 | void setFont(int elementNum, Font font) |
| MIDP 1.0 | void setSelectedFlags(boolean[] selectedArray) |
| MIDP 1.0 | void setSelectedIndex(int elementNum, boolean selected) |
### Command

```java
javax.microedition.lcdui
```

#### Object

- `Command`

#### Methods

- **MIDP 1.0**
  - `int size()`

- **MIDP 3.0**
  - `int BACK`
  - `int CANCEL`
  - `int EXIT`
  - `int HELP`
  - `int ITEM`
  - `int OK`
  - `int SCREEN`
  - `int STOP`

- **MIDP 2.0**
  - `int getCommandType()`
  - `boolean getEnabled()`
  - `Font getFont()`
  - `Image getImage()`
  - `String getLabel()`
  - `String getLongLabel()`
  - `int getPriority()`

- **MIDP 3.0**
  - `Command(String label, int commandType, int priority)`
  - `Command(String shortLabel, String longLabel, Image image, int commandType, int priority)`
  - `Command(String shortLabel, String longLabel, int commandType, int priority)`

- **MIDP 3.0**
  - `void onParentEnabled(boolean enabled)`
  - `void setEnabled(boolean enabled)`
  - `void setFont(Font font)`
  - `void setImage(Image image)`
  - `void setLabel(String label)`
  - `void setLongLabel(String longLabel)`

#### CommandLayoutPolicy

- `CommandLayoutPolicy`

```java
javax.microedition.lcdui
```
<table>
<thead>
<tr>
<th>MIDP 3.0</th>
<th>void</th>
<th>onCommandLayout(Displayable displayable)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CommandListener</strong></td>
<td>javax.microedition.lcdui</td>
<td>CommandListener</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void</td>
<td>commandAction(Command c,Displayable d)</td>
</tr>
<tr>
<td><strong>CommConnection</strong></td>
<td>javax.microedition.io</td>
<td>CommConnection StreamConnection</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int</td>
<td>getBaudRate()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int</td>
<td>setBaudRate(int baudrate)</td>
</tr>
<tr>
<td><strong>CustomItem</strong></td>
<td>javax.microedition.lcdui</td>
<td>Object Item CustomItem</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>✽✦</td>
<td>CustomItem(String label)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>✽✦</td>
<td>int getGameAction(int keyCode)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>✽✦</td>
<td>int getInteractionModes()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>✽✦</td>
<td>int getKeyCode(int action)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>✽✦</td>
<td>int getMinContentHeight()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>✽✦</td>
<td>int getMinContentWidth()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>✽✦</td>
<td>int getPrefContentHeight(int width)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>✽✦</td>
<td>int getPrefContentWidth(int height)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>✽</td>
<td>void hideNotify()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>✽✦</td>
<td>void invalidate()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>✽✦</td>
<td>int KEY_PRESS</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>✽✦</td>
<td>int KEY_RELEASE</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>✽✦</td>
<td>int KEY_REPEAT</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>✽</td>
<td>void keyPressed(int keyCode)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>✽</td>
<td>void keyReleased(int keyCode)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>✽</td>
<td>void keyRepeated(int keyCode)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>✽✦</td>
<td>int NONE</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>✽✦</td>
<td>void paint(Graphics g,int w,int h)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>✽✦</td>
<td>int POINTER_DRAG</td>
</tr>
</tbody>
</table>
### MIDP 2.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int</code></td>
<td><code>POINTER_PRESS</code></td>
</tr>
<tr>
<td><code>int</code></td>
<td><code>POINTER_RELEASE</code></td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>pointerDragged(int x, int y)</code></td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>pointerPressed(int x, int y)</code></td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>pointerReleased(int x, int y)</code></td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>repaint()</code></td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>repaint(int x, int y, int w, int h)</code></td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>setKeyListener(KeyListener listener)</code></td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>setPaintMode(boolean opaque)</code></td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>showNotify()</code></td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>sizeChanged(int w, int h)</code></td>
</tr>
<tr>
<td><code>int</code></td>
<td><code>TRAVERSE_HORIZONTAL</code></td>
</tr>
<tr>
<td><code>int</code></td>
<td><code>TRAVERSE_VERTICAL</code></td>
</tr>
<tr>
<td><code>boolean</code></td>
<td><code>traverse(int dir, int viewportWidth, int viewportHeight, int[] visRect_inout)</code></td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>traverseOut()</code></td>
</tr>
</tbody>
</table>

### MIDP 3.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void</code></td>
<td><code>addDisplayListener(DisplayListener l)</code></td>
</tr>
</tbody>
</table>

---

### DateField

- **Object**: `Item`
- **Item**: `DateField`

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int</code></td>
<td><code>DATE</code></td>
</tr>
<tr>
<td><code>int</code></td>
<td><code>DATE_TIME</code></td>
</tr>
<tr>
<td><code>DateField(String label, int mode)</code></td>
<td><code>DateField(String label, int mode, TimeZone timeZone)</code></td>
</tr>
<tr>
<td><code>Date</code></td>
<td><code>getDateTime()</code></td>
</tr>
<tr>
<td><code>int</code></td>
<td><code>getInputMode()</code></td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>setDate(Date date)</code></td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>setInputMode(int mode)</code></td>
</tr>
<tr>
<td><code>int</code></td>
<td><code>TIME</code></td>
</tr>
</tbody>
</table>

---

### Display

- **Object**: `Display`

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void</code></td>
<td><code>addDisplayListener(DisplayListener l)</code></td>
</tr>
</tbody>
</table>
MIDP 2.0  ⚫ int ALERT
MIDP 1.0 void callSerially(Runnable r)
MIDP 2.0  ⚫ int CHOICE_GROUP_ELEMENT
MIDP 2.0  ⚫ int COLOR_BACKGROUND
MIDP 2.0  ⚫ int COLOR_BORDER
MIDP 2.0  ⚫ int COLOR_FOREGROUND
MIDP 2.0  ⚫ int COLOR_HIGHLIGHTED_BACKGROUND
MIDP 2.0  ⚫ int COLOR_HIGHLIGHTED_BORDER
MIDP 2.0  ⚫ int COLOR_HIGHLIGHTED_FOREGROUND
MIDP 3.0  ⚫ int COLOR_IDLE_BACKGROUND
MIDP 3.0  ⚫ int COLOR_IDLE_FOREGROUND
MIDP 3.0  ⚫ int COLOR_IDLE_HIGHLIGHTED_BACKGROUND
MIDP 3.0  ⚫ int COLOR_IDLE_HIGHLIGHTED_FOREGROUND
MIDP 3.0  ⚫ int COMMAND
MIDP 3.0  ⚫ int DISPLAY_HARDWARE_ABSENT
MIDP 3.0  ⚫ int DISPLAY_HARDWARE_DISABLED
MIDP 3.0  ⚫ int DISPLAY_HARDWARE_ENABLED
MIDP 2.0  boolean flashBacklight(int duration)
MIDP 3.0  int get.ActivityMode()
MIDP 2.0  int getBestImageHeight(int imageType)
MIDP 2.0  int getBestImageWidth(int imageType)
MIDP 2.0  int getBorderStyle(boolean highlighted)
MIDP 3.0  int getCapabilities()
MIDP 2.0  int getColor(int colorSpecifier)
MIDP 3.0  CommandLayoutPolicy getCommandLayoutPolicy()
MIDP 3.0  int[] getCommandPreferredPlacements(int commandType)
MIDP 1.0  Displayable getCurrent()
MIDP 1.0  Display getDisplay(MIDlet m)
MIDP 3.0  Display[] getDisplays(int capabilities)
MIDP 3.0  int getDisplayState()
MIDP 3.0  int getDotPitch()
MIDP 3.0  int[] getExactPlacementPositions(int border)
<table>
<thead>
<tr>
<th>MIDP 3.0</th>
<th>int getHardwareState()</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 3.0</td>
<td>int getHeight()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>IdleItem getIdleItem()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int[] getMenuPreferredPlacements()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int[] getMenuSupportedPlacements()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int getOrientation()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int getWidth()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>boolean hasPointerEvents()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>boolean hasPointerMotionEvents()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>boolean isBuiltIn()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>boolean isColor()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int LIST_ELEMENT</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int MENU</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int MODE_ACTIVE</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int MODE_NORMAL</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int NOTIFICATION</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int numAlphaLevels()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int numColors()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int ORIENTATION_LANDSCAPE</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int ORIENTATION_LANDSCAPE_180</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int ORIENTATION_PORTRAIT</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int ORIENTATION_PORTRAIT_180</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void removeCurrent()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void removeDisplayListener(DisplayListener l)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void setActivityMode(int mode)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void setCommandLayoutPolicy(CommandLayoutPolicy policy)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void setCurrent(Alert alert, Displayable nextDisplayable)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void setCurrent(Displayable nextDisplayable)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>void setCurrentItem(Item item)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void setIdleItem(IdleItem idleItem)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void setPreferredOrientation(int orientation)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int SOFTKEY_BOTTOM</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>boolean</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Displayable</th>
<th>javax.microedition.lcdui</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>➡Displayable</td>
</tr>
</tbody>
</table>

<p>| MIDP 3.0 | void | addCommand(Command cmd) |
| MIDP 3.0 | Command | getCommand(int placement) |
| MIDP 3.0 | CommandLayoutPolicy | getCommandLayoutPolicy() |</p>
<table>
<thead>
<tr>
<th>MIDP 3.0</th>
<th>Command[] getCommands()</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 3.0</td>
<td>Display getCurrentDisplay()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int getHeight()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>Menu getMenu(int placement)</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>MIDP 3.0</td>
<td>void invalidateCommandLayout()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>boolean isShown()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void removeCommand(Command cmd)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void removeCommandOrMenu(int placement)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void setCommand(Command cmd,int placement)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void setCommandLayoutPolicy(CommandLayoutPolicy policy)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void setCommandListener(CommandListener l)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void setMenu(Menu menu,int placement)</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>

### DisplayCapabilityException

<table>
<thead>
<tr>
<th>javax.microedition.lcdui.displayCapabilityException</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throwable</td>
</tr>
<tr>
<td>Exception</td>
</tr>
<tr>
<td>RuntimeException</td>
</tr>
<tr>
<td>DisplayCapabilityException</td>
</tr>
</tbody>
</table>

| MIDP 3.0 | * | DisplayCapabilityException() |
| MIDP 3.0 | * | DisplayCapabilityException(String s) |

### DisplayListener

<table>
<thead>
<tr>
<th>javax.microedition.lcdui.displayListener</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisplayListener</td>
</tr>
</tbody>
</table>

<p>| MIDP 3.0 | void displayAdded(Display d) |
| MIDP 3.0 | void displayStateChanged(Display d,int newState) |
| MIDP 3.0 | void hardwareStateChanged(Display d,int newState) |
| MIDP 3.0 | void orientationChanged(Display d,int newOrientation) |
| MIDP 3.0 | void sizeChanged(Display d,int w,int h) |</p>
<table>
<thead>
<tr>
<th>MIDP 3.0</th>
<th>String</th>
<th>APPLICATION_RELAUNCH_PREFIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 3.0</td>
<td>String</td>
<td>AUDIO_MUTE</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String</td>
<td>BACKLIGHT</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
<td>BACKLIGHT_DIM</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
<td>BACKLIGHT_OFF</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
<td>BACKLIGHT_ON</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String</td>
<td>BATTERY_CHARGING</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String</td>
<td>BATTERY_LEVEL</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String</td>
<td>BATTERY_LOW</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String</td>
<td>DATA_NETWORK</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>boolean</td>
<td>equals(Object object)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td></td>
<td>EventData(String event,boolean value,String message,Object info)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td></td>
<td>EventData(String event,double value,String message,Object info)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td></td>
<td>EventData(String event,long value,String message,Object info)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td></td>
<td>EventData(String event,String value,String message,Object info)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String</td>
<td>EXTERNAL_POWER</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String</td>
<td>FLIGHT_MODE</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>boolean</td>
<td>getBoolean()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>double</td>
<td>getDouble()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>float</td>
<td>getFloat()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>Object</td>
<td>getInfo()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
<td>getInt()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>long</td>
<td>getLong()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String</td>
<td>getMessage()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String</td>
<td>getName()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>MIDletIdentity</td>
<td>getSourceInfo()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String</td>
<td>getString()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>long</td>
<td>getTimestamp()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>Object</td>
<td>getValue()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
<td>hashCode()</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String NETWORK_3GPP_CSD</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String NETWORK_3GPP_PD</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String NETWORK_3GPP_PD_3G</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String NETWORK_3GPP_PD_EDGE</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String NETWORK_3GPP_PD_HSDPA</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String NETWORK_802DOT11</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String NETWORK_802DOT16</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String NETWORK_CDMA</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String PROFILE_ACTIVATED</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String PROFILE_GENERAL</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String PROFILE_MEETING</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String PROFILE_OFFLINE</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String PROFILE_OUTDOOR</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String PROFILE_PAGER</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String PROFILE_SILENT</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String PROFILE_SYSTEM1</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String PROFILE_SYSTEM2</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String PROFILE_SYSTEM3</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String PROFILE_SYSTEM4</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String PROFILE_USER1</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String PROFILE_USER2</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String PROFILE_USER3</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String PROFILE_USER4</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String SCREENSAVER_MODE</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String SCREENSAVER_MODE_ACTIVATED</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String SCREENSAVER_MODE_DEACTIVATED</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String SYSTEM_STATE</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String SYSTEM_STATE_NORMAL</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String SYSTEM_STATE_SHUTDOWN</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String SYSTEM_STATE_STANDBY</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String SYSTEM_STATE_STARTUP</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String toString()</td>
<td></td>
</tr>
</tbody>
</table>
MIDP 3.0  String VOICE_CALL

<table>
<thead>
<tr>
<th><strong>EventDataListener</strong></th>
<th>javax.microedition.event</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventDataListener</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void handleEvent(EventData value)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>EventManager</strong></th>
<th>javax.microedition.event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td></td>
</tr>
<tr>
<td>➡ EventManager</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void addEventListener(String event,EventDataListener listener,boolean authmode)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void addEventListener(String event,EventDataListener listener,boolean value)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void addEventListener(String event,EventDataListener listener,boolean authmode,double low,double high)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void addEventListener(String event,EventDataListener listener,boolean authmode,long low,long high)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void addEventListener(String event,EventDataListener listener,boolean authmode,java.lang.String[] values)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>EventData getCurrent(String event)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>EventManager getInstance()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String[] getSystemEvents()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void post(EventData event,boolean authmode)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void registerApplication(String event,String application,boolean authmode)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void registerApplication(String event,String application,boolean value)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void registerApplication(String event,String application,boolean authmode,double low,double high)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void registerApplication(String event,String application,boolean authmode,long low,long high)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void registerApplication(String event,String application,boolean authmode,java.lang.String[] values)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void removeEventListener(String event,EventDataListener listener)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void unregisterApplication(String event,String application)</td>
</tr>
</tbody>
</table>
### EventPermission

<table>
<thead>
<tr>
<th>MIDP 3.0</th>
<th>boolean</th>
<th>equals(Object object)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 3.0</td>
<td>*</td>
<td>EventPermission(String event, String actions)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String</td>
<td>getActions()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
<td>hashCode()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>boolean</td>
<td>implies(Permission p)</td>
</tr>
</tbody>
</table>

### FileSelector

<table>
<thead>
<tr>
<th>MIDP 3.0</th>
<th>void</th>
<th>addCommand(Command cmd)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 3.0</td>
<td>Command</td>
<td>CANCEL_COMMAND</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
<td>DIRECTORY_CREATE</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
<td>DIRECTORY_SELECT</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>FileSelector(String title, int mode)</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String[]</td>
<td>getFilterExtensions()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
<td>getHeight()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
<td>getMode()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String</td>
<td>getURL()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
<td>getWidth()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
<td>LOAD</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>Command</td>
<td>OK_COMMAND</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>StreamConnection</td>
<td>open(int mode, boolean timeouts)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
<td>removeCommand(Command cmd)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
<td>removeCommandOrMenu(int placement)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
<td>SAVE</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
<td>setCommand(Command cmd, int placement)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
<td>setCommandLayoutPolicy(CommandLayoutPolicy policy)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
<td>setFilterExtensions(java.lang.String[] extensions)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
<td>setMenu(Menu menu, int placement)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void setMode(int mode)</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void setURL(String URL)</td>
<td></td>
</tr>
</tbody>
</table>

## Font

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int charsWidth(char[] ch, int offset, int length)</td>
<td>MIDP 1.0</td>
</tr>
<tr>
<td>int charWidth(char ch)</td>
<td>MIDP 1.0</td>
</tr>
<tr>
<td>Font createFont(InputStream fontData)</td>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>Font deriveFont(int pixelSize)</td>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>Font deriveFont(int style, int pixelSize)</td>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>boolean equals(Object obj)</td>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>int FACE_MONOSPACE</td>
<td>MIDP 1.0</td>
</tr>
<tr>
<td>int FACE_PROPORTIONAL</td>
<td>MIDP 1.0</td>
</tr>
<tr>
<td>int FACE_SYSTEM</td>
<td>MIDP 1.0</td>
</tr>
<tr>
<td>int FONT_IDLE_HIGHLIGHTED_TEXT</td>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>int FONT_IDLE_TEXT</td>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>int FONT_INPUT_TEXT</td>
<td>MIDP 2.0</td>
</tr>
<tr>
<td>int FONT_STATIC_TEXT</td>
<td>MIDP 2.0</td>
</tr>
<tr>
<td>intgetAscent()</td>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>Font[] getAvailableFonts()</td>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>Font[] getAvailableFonts(int style)</td>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>Font[] getAvailableFonts(int face, int style, int pixelSize)</td>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>int getBaselinePosition()</td>
<td>MIDP 1.0</td>
</tr>
<tr>
<td>Font getDefaultFont()</td>
<td>MIDP 1.0</td>
</tr>
<tr>
<td>int getDescent()</td>
<td>MIDP 1.0</td>
</tr>
<tr>
<td>int getFace()</td>
<td>MIDP 1.0</td>
</tr>
<tr>
<td>String getFamily()</td>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>Font getFont(int fontSpecifier)</td>
<td>MIDP 2.0</td>
</tr>
<tr>
<td>Font getFont(int face, int style, int size)</td>
<td>MIDP 1.0</td>
</tr>
<tr>
<td>Font getFont(String name, int style, int pixelSize)</td>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>String getFontName()</td>
<td>MIDP 1.0</td>
</tr>
<tr>
<td>int getHeight()</td>
<td>MIDP 1.0</td>
</tr>
</tbody>
</table>
MIDP 3.0

int getLeading()

MIDP 3.0

int getMaxAscent()

MIDP 3.0

int getMaxDescent()

MIDP 1.0

String getName()

MIDP 3.0

int getPixelSize()

MIDP 3.0

int getPixelSize(String name)

MIDP 1.0

int getSize()

MIDP 1.0

int getStyle()

MIDP 3.0

int getStyle(String name)

MIDP 1.0

int hashCode()

MIDP 1.0

boolean isBold()

MIDP 1.0

boolean isItalic()

MIDP 1.0

boolean isPlain()

MIDP 1.0

boolean isUnderlined()

MIDP 1.0

int SIZE_LARGE

MIDP 1.0

int SIZE_MEDIUM

MIDP 1.0

int SIZE_SMALL

MIDP 1.0

int stringWidth(String str)

MIDP 1.0

int STYLE_BOLD

MIDP 1.0

int STYLE_ITALIC

MIDP 1.0

int STYLE_PLAIN

MIDP 1.0

int STYLE_UNDERLINED

MIDP 1.0

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

FontFormatException

javax.microedition.midlet

Object

↳ Throwable

↳ Exception

↳ RuntimeException

↳ FontFormatException

MIDP 3.0

FontFormatException()

MIDP 3.0

FontFormatException(String s)

Form

javax.microedition.midlet
<table>
<thead>
<tr>
<th>Object</th>
<th>Displayable</th>
<th>Screen</th>
<th>Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 1.0</td>
<td>int</td>
<td>append(Image img)</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int</td>
<td>append(Item item)</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int</td>
<td>append(String str)</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void</td>
<td>delete(int itemNum)</td>
<td></td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>void</td>
<td>deleteAll()</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>Form(String title)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>Form(String title, javax.microedition.lcdui.Item[] items)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>Item</td>
<td>get(int itemNum)</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>Item</td>
<td>getCurrent()</td>
<td></td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int</td>
<td>getHeight()</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>FormLayoutPolicy</td>
<td>getLayoutPolicy()</td>
<td></td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int</td>
<td>getWidth()</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void</td>
<td>insert(int itemNum, Item item)</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void</td>
<td>set(int itemNum, Item item)</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void</td>
<td>setStateListener(ItemStateListener iListener)</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
<td>setTraversalListener(ItemTraversalListener iListener)</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
<td>setLayoutPolicy(FormLayoutPolicy layoutPolicy)</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int</td>
<td>size()</td>
<td></td>
</tr>
</tbody>
</table>

**FormLayoutPolicy**

```
javax.microedition.lcdui
```

<table>
<thead>
<tr>
<th>Object</th>
<th>FormLayoutPolicy</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>Form</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
</tbody>
</table>
### MIDP 3.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getTraverse()</code></td>
<td><code>Item getTraverse(Item item, int dir)</code></td>
</tr>
<tr>
<td><code>getWidth()</code></td>
<td><code>int getWidth(Item item)</code></td>
</tr>
<tr>
<td><code>getX()</code></td>
<td><code>int getX(Item item)</code></td>
</tr>
<tr>
<td><code>getY()</code></td>
<td><code>int getY(Item item)</code></td>
</tr>
<tr>
<td><code>isValid()</code></td>
<td><code>boolean isValid(Item item)</code></td>
</tr>
<tr>
<td><code>setPosition()</code></td>
<td><code>void setPosition(Item item, int x, int y)</code></td>
</tr>
<tr>
<td><code>setSize()</code></td>
<td><code>void setSize(Item item, int width, int height)</code></td>
</tr>
<tr>
<td><code>setValid()</code></td>
<td><code>void setValid(Item item, boolean valid)</code></td>
</tr>
</tbody>
</table>

### GameCanvas

Object
- `Displayable`
- `Canvas`
- `GameCanvas`

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DOWN_PRESSED</code></td>
<td><code>int DOWN_PRESSED</code></td>
</tr>
<tr>
<td><code>FIRE_PRESSED</code></td>
<td><code>int FIRE_PRESSED</code></td>
</tr>
<tr>
<td><code>flushGraphics()</code></td>
<td><code>void flushGraphics()</code></td>
</tr>
<tr>
<td><code>flushGraphics(int x, int y, int width, int height)</code></td>
<td><code>void flushGraphics(int x, int y, int width, int height)</code></td>
</tr>
<tr>
<td><code>GAME_A_PRESSED</code></td>
<td><code>int GAME_A_PRESSED</code></td>
</tr>
<tr>
<td><code>GAME_B_PRESSED</code></td>
<td><code>int GAME_B_PRESSED</code></td>
</tr>
<tr>
<td><code>GAME_C_PRESSED</code></td>
<td><code>int GAME_C_PRESSED</code></td>
</tr>
<tr>
<td><code>GAME_D_PRESSED</code></td>
<td><code>int GAME_D_PRESSED</code></td>
</tr>
<tr>
<td><code>GameCanvas(boolean suppressKeyEvents)</code></td>
<td><code>GameCanvas(boolean suppressKeyEvents)</code></td>
</tr>
<tr>
<td><code>GameCanvas(boolean suppressKeyEvents, boolean preserveBuffer)</code></td>
<td><code>GameCanvas(boolean suppressKeyEvents, boolean preserveBuffer)</code></td>
</tr>
<tr>
<td><code>getGraphics()</code></td>
<td><code>Graphics getGraphics()</code></td>
</tr>
<tr>
<td><code>getKeyStates()</code></td>
<td><code>int getKeyStates()</code></td>
</tr>
<tr>
<td><code>LEFT_PRESSED</code></td>
<td><code>int LEFT_PRESSED</code></td>
</tr>
<tr>
<td><code>paint(Graphics g)</code></td>
<td><code>void paint(Graphics g)</code></td>
</tr>
<tr>
<td><code>RIGHT_PRESSED</code></td>
<td><code>int RIGHT_PRESSED</code></td>
</tr>
<tr>
<td><code>UP_PRESSED</code></td>
<td><code>int UP_PRESSED</code></td>
</tr>
</tbody>
</table>

### Gauge

Object
- `Item`
- `Gauge`

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
</table>

```java
javax.microedition.lcdui.game
javax.microedition.lcdui
```
<table>
<thead>
<tr>
<th>MIDP 2.0</th>
<th>int CONTINUOUS_IDLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 2.0</td>
<td>int CONTINUOUS_RUNNING</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>Gauge(String label,boolean interactive,int maxVal,ute int initialVal)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int getIncrementValue()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int getMaxValue()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int getMinValue()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int getValue()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int INCREMENTAL_IDLE</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int INCREMENTAL_UPDATING</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int INDEFINITE</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>boolean isInteractive()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void setIncrementValue(int incrementValue)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void setMaxValue(int maxVal)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void setMinValue(int minVal)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void setValue(int value)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Graphics</th>
<th>javax.microedition.lcdui</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>Graphics</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int BASELINE</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int BOTTOM</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void clipRect(int x,int y,int width,int height)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>void copyArea(int x_src,int y_src,int width,int height,int x_dest, int y_dest, int anchor)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void drawArc(int x,int y,int width,int height,int startAngle,int arcAngle)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void drawARGB16(short[] argbData,int offset,int scanlen,nt x,int y,int width,int height)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void drawChar(char character,int x,int y,int anchor)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void drawChars(char[] data,int offset,int length,int x,int y,int anchor)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void drawImage(Image img,int x,int y,int anchor)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void drawLine(int x1,int y1,int x2,int y2)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void drawRect(int x,int y,int width,int height)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>drawRegion(Image src, int x_src, int y_src, int width, int height, int transform, int x_dest, int y_dest, int anchor)</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>drawRegion(Image src, int x_src, int y_src, int width, int height, int transform, int x_dest, int y_dest, int anchor, int width_dest, int height_dest)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>void drawRegion(int x_src, int y_src, int width, int height, int transform, int x_dest, int y_dest, int anchor)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void height_dest)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>void drawRGB(int[] rgbData, int offset, int scanlength, int x, int y, int width, int height, boolean processAlpha)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void drawRGB16(short[] rgbData, int offset, int scanlength, int x, int y, int width, int height)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void drawString(String str, int x, int y, int anchor)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void drawSubstring(String str, int offset, int len, int x, int y, int anchor)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void drawText(Text text, int x, int y)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void fillRect(int x, int y, int width, int height)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void arcHeight)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>void fillTriangle(int x1, int y1, int x2, int y2, int x3, int y3)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int getAlpha()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int getAlphaColor()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int getBlendingMode()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int getBlueComponent()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int getClipHeight()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int getClipWidth()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int getClipX()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int getClipY()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int getColor()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int getDisplayColor(int color)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int getFont()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int getGrayScale()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int getGreenComponent()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int getRedComponent()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int getStrokeStyle()</td>
</tr>
</tbody>
</table>
MIDP 1.0 | int getTranslateX()
MIDP 1.0 | int getTranslateY()
MIDP 1.0 | int HCENTER
MIDP 1.0 | int LEFT
MIDP 1.0 | int RIGHT
MIDP 3.0 | void setAlpha(int alpha)
MIDP 3.0 | void setAlphaColor(int ARGB)
MIDP 3.0 | void setAlphaColor(int alpha, int red, int green, int blue)
MIDP 3.0 | void setBlendingMode(int mode)
MIDP 1.0 | void setClip(int x, int y, int width, int height)
MIDP 1.0 | void setColor(int RGB)
MIDP 1.0 | void setColor(int red, int green, int blue)
MIDP 1.0 | void setFont(Font font)
MIDP 1.0 | void setGrayScale(int value)
MIDP 1.0 | void setStrokeStyle(int style)
MIDP 1.0 | int SOLID
MIDP 3.0 | int SRC
MIDP 3.0 | int SRC_OVER
MIDP 1.0 | int TOP
MIDP 1.0 | void translate(int x, int y)
MIDP 1.0 | int VCENTER

**HttpConnection**

<table>
<thead>
<tr>
<th>javax.microedition.io</th>
</tr>
</thead>
<tbody>
<tr>
<td>HttpConnection</td>
</tr>
<tr>
<td>ContentConnection</td>
</tr>
</tbody>
</table>

MIDP 3.0 | String DELETE
MIDP 1.0 | String GET
MIDP 1.0 | long getDate()
MIDP 1.0 | long getExpiration()
MIDP 1.0 | String getFile()
MIDP 1.0 | String getHeaderField(int n)
MIDP 1.0 | String getHeaderField(String name)
MIDP 1.0 | long getHeaderFieldDate(String name, long def)
<table>
<thead>
<tr>
<th>MIDP 1.0</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>int getHeaderFieldInt(String name,int def)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>String getHeaderFieldKey(int n)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>String getHost()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>long getLastModified()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int getPort()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>String getProtocol()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>String getQuery()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>String getRef()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>String getRequestMethod()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>String getRequestProperty(String key)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int getResponseCode()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>String getResponseMessage()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>String getURL()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>String HEAD</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_ACCEPTED</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_BAD_GATEWAY</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_BAD_METHOD</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_BAD_REQUEST</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_CLIENT_TIMEOUT</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_CONFLICT</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_CREATED</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_ENTITY_TOO_LARGE</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_EXPECT_FAILED</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_FORBIDDEN</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_GATEWAY_TIMEOUT</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_GONE</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_INTERNAL_ERROR</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_LENGTH_REQUIRED</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_MOVED_PERM</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_MOVED_TEMP</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_MULT_CHOICE</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_NO_CONTENT</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_NOT_ACCEPTABLE</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_NOT_AUTHORITATIVE</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_NOT_FOUND</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_NOT_IMPLEMENTED</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_NOT_MODIFIED</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_OK</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_PARTIAL</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_PAYMENT_REQUIRED</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_PRECON_FAILED</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_PROXY_AUTH</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_REQ_TOO_LONG</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_RESET</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTPSEE_OTHER</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_TEMP_REDIRECT</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_UNAUTHORIZED</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_UNAVAILABLE</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_UNSUPPORTED_RANGE</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_UNSUPPORTED_TYPE</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_USE_PROXY</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int HTTP_VERSION</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>String POST</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>String PUT</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void setRequestMethod(String method)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void setRequestProperty(String key, String value)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>javax.microedition.io.HttpsConnection</th>
<th>javax.microedition.io.HttpsConnection</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 2.0</td>
<td>int getPort()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>SecurityInfo getSecurityInfo()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>javax.microedition.lcdui.IdleItem</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.microedition.lcdui.IdleItem</td>
</tr>
</tbody>
</table>
### MIDP 3.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void addedToDisplay(Display display)</code></td>
<td></td>
</tr>
<tr>
<td><code>IdleItem(String label)</code></td>
<td></td>
</tr>
<tr>
<td><code>void removedFromDisplay(Display display)</code></td>
<td></td>
</tr>
</tbody>
</table>

### Image

![Image class structure](image.png)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Image creatImage(byte[] imageData,int imageOffset,int imageLength)</code></td>
<td></td>
</tr>
<tr>
<td><code>Image creatImage(Image source)</code></td>
<td></td>
</tr>
<tr>
<td><code>Image creatImage(Image image,int x,int y,int width,int height,int transform)</code></td>
<td></td>
</tr>
<tr>
<td><code>Image creatImage(Image image,int x,int y,int width,int height,int transform,int img_width,int img_height)</code></td>
<td></td>
</tr>
<tr>
<td><code>Image creatImage(InputStream stream)</code></td>
<td></td>
</tr>
<tr>
<td><code>Image creatImage(int width,int height)</code></td>
<td></td>
</tr>
<tr>
<td><code>Image creatImage(int width,int height,boolean withAlpha,int fillColor)</code></td>
<td></td>
</tr>
<tr>
<td><code>Image creatImage(String name)</code></td>
<td></td>
</tr>
<tr>
<td><code>Image createRGBImage(int[] rgb,int width,int height,boolean processAlpha)</code></td>
<td></td>
</tr>
<tr>
<td><code>void getARGB16(short[] argbData,int offset,int scanlength,int x,int y,int width,int height)</code></td>
<td></td>
</tr>
<tr>
<td><code>Graphics getGraphics()</code></td>
<td></td>
</tr>
<tr>
<td><code>int getHeight()</code></td>
<td></td>
</tr>
<tr>
<td><code>getRGB(int[] rgbData,int offset,int scanlength,int x,int y,int width,int height)</code></td>
<td></td>
</tr>
<tr>
<td><code>void getRGB16(short[] rgbData,int offset,int scanlength,int x,int y,int width,int height)</code></td>
<td></td>
</tr>
<tr>
<td><code>int getWidth()</code></td>
<td></td>
</tr>
<tr>
<td><code>boolean hasAlpha()</code></td>
<td></td>
</tr>
<tr>
<td><code>boolean isAnimated()</code></td>
<td></td>
</tr>
<tr>
<td><code>boolean isMutable()</code></td>
<td></td>
</tr>
<tr>
<td><code>boolean isScalable()</code></td>
<td></td>
</tr>
</tbody>
</table>
### ImageItem

<table>
<thead>
<tr>
<th>Class</th>
<th>String getAltText()</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 1.0</td>
<td>String getImage()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int getLayout()</td>
</tr>
<tr>
<td>MIDP 1.0</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>
<tr>
<td>MIDP 1.0</td>
<td>int LAYOUT_CENTER</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int LAYOUT_DEFAULT</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int LAYOUT_LEFT</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int LAYOUT_NEWLINE_AFTER</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int LAYOUT_NEWLINE_BEFORE</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int LAYOUT_RIGHT</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void setAltText(String text)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void setImage(Image img)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void setLayout(int layout)</td>
</tr>
</tbody>
</table>

### IMCConnection

<table>
<thead>
<tr>
<th>Class</th>
<th>javax.microedition.io.IMCConnection</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 3.0</td>
<td>MIDletIdentity getRemotefIdentity()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String getRequestedServerVersion()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String getServerName()</td>
</tr>
</tbody>
</table>

### IMCServerConnection

<table>
<thead>
<tr>
<th>Class</th>
<th>javax.microedition.io.IMCServerConnection</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 3.0</td>
<td>String getName()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String getVersion()</td>
</tr>
</tbody>
</table>

### InvalidRecordIDException

| Class | javax.microedition.rms.InvalidRecordIDException |

Mobile Information Device Profile v3.0 - JSR 271
© Copyright Motorola, Inc. 2002-2009. All Rights Reserved.
Java Community Process - Final Release
Page 834 of 891
<table>
<thead>
<tr>
<th>Object</th>
<th>Throwable</th>
<th>Exception</th>
<th>RecordStoreException</th>
<th>InvalidRecordIDException</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 1.0</td>
<td>*</td>
<td>InvalidRecordIDException()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>*</td>
<td>InvalidRecordIDException(String message)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Item

```java
javax.microedition.lcdui
```

<table>
<thead>
<tr>
<th>Object</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 2.0</td>
<td>void addCommand(Command cmd)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int BUTTON</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>Command[] getCommands()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>String getLabel()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int getLayout()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>ItemLayoutHint getLayoutHint()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int getMinimumHeight()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int getMinimumWidth()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int getPreferredHeight()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int getPreferredWidth()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int HYPERLINK</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int LAYOUT_2</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int LAYOUT_BOTTOM</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int LAYOUT_CENTER</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int LAYOUT_DEFAULT</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int LAYOUT_EXPAND</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int LAYOUT_LEFT</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int LAYOUT_NEWLINE_AFTER</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int LAYOUT_NEWLINE_BEFORE</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int LAYOUT_RIGHT</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int LAYOUT_SHRINK</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int LAYOUT_TOP</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int LAYOUT_VCENTER</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>void</td>
</tr>
</tbody>
</table>

### ItemCommandListener

**javax.microedition.lcdui**

<table>
<thead>
<tr>
<th>ItemCommandListener</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemCommandListener</td>
</tr>
</tbody>
</table>

| MIDP 2.0 | void | commandAction(Command c, Item item) |

### ItemLayoutHint

**javax.microedition.lcdui**

<table>
<thead>
<tr>
<th>ItemLayoutHint</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemLayoutHint</td>
</tr>
</tbody>
</table>

### ItemStateListener

**javax.microedition.lcdui**

<table>
<thead>
<tr>
<th>ItemStateListener</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemStateListener</td>
</tr>
</tbody>
</table>

| MIDP 1.0 | void | itemStateChanged(Item item) |

### ItemTraversalListener

**javax.microedition.lcdui**

<table>
<thead>
<tr>
<th>ItemTraversalListener</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemTraversalListener</td>
</tr>
</tbody>
</table>

| MIDP 3.0 | void | itemTraversedIn(Item item) |
| MIDP 3.0 | void | itemTraversedOut(Item item) |

### KeyListener

**javax.microedition.lcdui**

<table>
<thead>
<tr>
<th>KeyListener</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyListener</td>
</tr>
</tbody>
</table>

| MIDP 3.0 | void | keyPressed(int keyCode, int keyModifier) |
| MIDP 3.0 | void | keyReleased(int keyCode, int keyModifier) |
| MIDP 3.0 | void | keyRepeated(int keyCode, int keyModifier) |
| MIDP 3.0 | int | MODIFIER_ALT |
### Layer

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getHeight()</td>
<td>Returns the height of the layer.</td>
</tr>
<tr>
<td>getWidth()</td>
<td>Returns the width of the layer.</td>
</tr>
<tr>
<td>getX()</td>
<td>Returns the x-coordinate.</td>
</tr>
<tr>
<td>getY()</td>
<td>Returns the y-coordinate.</td>
</tr>
<tr>
<td>isVisible()</td>
<td>Returns true if the layer is visible.</td>
</tr>
<tr>
<td>move()</td>
<td>Moves the layer by dx and dy.</td>
</tr>
<tr>
<td>paint()</td>
<td>Paints the layer.</td>
</tr>
<tr>
<td>setPosition()</td>
<td>Sets the position of the layer.</td>
</tr>
<tr>
<td>setVisible()</td>
<td>Sets the visibility of the layer.</td>
</tr>
</tbody>
</table>

### LayerManager

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>append()</td>
<td>Appends a layer to the layer manager.</td>
</tr>
<tr>
<td>getLayerAt()</td>
<td>Returns the layer at the specified index.</td>
</tr>
<tr>
<td>getSize()</td>
<td>Returns the size of the layer manager.</td>
</tr>
<tr>
<td>insert()</td>
<td>Inserts a layer at the specified index.</td>
</tr>
<tr>
<td>LayerManager()</td>
<td>Constructs a new layer manager.</td>
</tr>
<tr>
<td>paint()</td>
<td>Paints the layer manager.</td>
</tr>
<tr>
<td>remove()</td>
<td>Removes a layer from the layer manager.</td>
</tr>
<tr>
<td>setViewWindow()</td>
<td>Sets the view window of the layer manager.</td>
</tr>
</tbody>
</table>

### List

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choice</td>
<td>Represents a choice within a list.</td>
</tr>
</tbody>
</table>
MIDP 1.0
int append(String stringPart, Image imagePart)

MIDP 1.0
void delete(int elementNum)

MIDP 1.0
void deleteAll()

MIDP 2.0
int getFitPolicy()

MIDP 2.0
Font getFont(int elementNum)

MIDP 2.0
int getHeight()

MIDP 1.0
Image getImage(int elementNum)

MIDP 1.0
int getSelectedFlags(boolean[] selectedArray_return)

MIDP 1.0
int getSelectedIndex()

MIDP 1.0
String getString(int elementNum)

MIDP 2.0
int getWidth()

MIDP 1.0
void insert(int elementNum, String stringPart, Image imagePart)

MIDP 3.0
boolean isEnabled(int elementNum)

MIDP 1.0
boolean isSelected(int elementNum)

MIDP 1.0
List(String title, int listType)

MIDP 1.0
List(String title, int listType, java.lang.String[] stringElements, javax.microedition.lcdui.Image[] imageElements)

MIDP 2.0
void removeCommand(Command cmd)

MIDP 1.0
Command SELECT_COMMAND

MIDP 1.0
void set(int elementNum, String stringPart, Image imagePart)

MIDP 3.0
void setEnabled(int elementNum, boolean isEnabled)

MIDP 2.0
void setFitPolicy(int fitPolicy)

MIDP 2.0
void setFont(int elementNum, Font font)

MIDP 2.0
void setSelectCommand(Command command)

MIDP 1.0
void setSelectedFlags(boolean[] selectedArray)

MIDP 1.0
void setSelectedIndex(int elementNum, boolean selected)

MIDP 1.0
int size()

---

**Menu**

<table>
<thead>
<tr>
<th>javax.microedition.lcdui</th>
</tr>
</thead>
</table>

Object ➡️ Menu

MIDP 3.0
int append(Command cmd)

MIDP 3.0
int append(Menu menu)
<table>
<thead>
<tr>
<th>MIDP 3.0</th>
<th>●</th>
<th>long  getSplashScreenTime()</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 3.0</td>
<td>●</td>
<td>boolean  isSelectedScreenSaver()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>✩</td>
<td>MIDlet()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>●</td>
<td>void  notifyDestroyed()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>●</td>
<td>void  notifyPaused()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>♦</td>
<td>void  pauseApp()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>●</td>
<td>boolean  platformRequest(String URL)</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>●</td>
<td>void  resumeRequest()</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>○♦</td>
<td>void  startApp()</td>
</tr>
</tbody>
</table>

**MIDletIdentity**

<table>
<thead>
<tr>
<th>javax.microedition.midlet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object  ➩ MIDletIdentity</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
</tbody>
</table>

**MIDletStateChangeException**

<table>
<thead>
<tr>
<th>javax.microedition.midlet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object  ➩ Throwable  ➩ Exception  ➩ MIDletStateChangeException</td>
</tr>
<tr>
<td>MIDP 1.0</td>
</tr>
<tr>
<td>MIDP 1.0</td>
</tr>
</tbody>
</table>

**Notification**

<table>
<thead>
<tr>
<th>javax.microedition.lcdui</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object  ➩ Notification</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>MIDP 3.0</td>
</tr>
</tbody>
</table>

**NotificationException**

javax.microedition.lcdui

| MIDP 3.0 | NotificationException() | |
| MIDP 3.0 | NotificationException(String s) | |

**NotificationListener**

javax.microedition.lcdui

| MIDP 3.0 | void notificationDismissed(Notification notification) | |
| MIDP 3.0 | void notificationSelected(Notification notification) | |
| MIDP 3.0 | void notificationTimeout(Notification notification) | |

**NotificationType**

javax.microedition.lcdui

| MIDP 3.0 | NotificationType | CALL |
| MIDP 3.0 | NotificationType | EMAIL |
| MIDP 3.0 | NotificationType | IM |
| MIDP 3.0 | NotificationType | MMS |
| MIDP 3.0 | NotificationType(String defaultLabel, Image defaultImage) | |
| MIDP 3.0 | NotificationType | REMINDER |
| MIDP 3.0 | NotificationType | SMS |
### PushRegistry

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>String getFilter(String connection)</code></td>
<td></td>
<td>MIDP 2.0: Returns the filter for a connection.</td>
</tr>
<tr>
<td><code>String getMIDlet(String connection)</code></td>
<td></td>
<td>MIDP 2.0: Returns the MIDlet for a connection.</td>
</tr>
<tr>
<td><code>String[] listConnections(boolean available)</code></td>
<td></td>
<td>MIDP 2.0: Returns a list of connections.</td>
</tr>
<tr>
<td><code>long registerAlarm(String midlet, long time)</code></td>
<td></td>
<td>MIDP 2.0: Registers an alarm for a midlet.</td>
</tr>
<tr>
<td><code>void registerConnection(String connection, String midlet, String filter)</code></td>
<td></td>
<td>MIDP 2.0: Registers a connection.</td>
</tr>
<tr>
<td><code>boolean unregisterConnection(String connection)</code></td>
<td></td>
<td>MIDP 2.0: Unregisters a connection.</td>
</tr>
</tbody>
</table>

### PushRegistryPermission

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>boolean equals(Object object)</code></td>
<td></td>
<td>MIDP 3.0: Determines if two permissions are equal.</td>
</tr>
<tr>
<td><code>String getActions()</code></td>
<td></td>
<td>MIDP 3.0: Returns the actions of the permission.</td>
</tr>
<tr>
<td><code>int hashCode()</code></td>
<td></td>
<td>MIDP 3.0: Returns the hash code of the permission.</td>
</tr>
<tr>
<td><code>boolean implies(Permission p)</code></td>
<td></td>
<td>MIDP 3.0: Determines if one permission implies another.</td>
</tr>
<tr>
<td><code>PushRegistryPermission(String uri, String actions)</code></td>
<td></td>
<td>MIDP 3.0: Constructor for a permission.</td>
</tr>
</tbody>
</table>

### RecordComparator

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int compare(byte[] rec1, byte[] rec2)</code></td>
<td></td>
<td>MIDP 1.0: Compares two records.</td>
</tr>
<tr>
<td><code>int EQUIVALENT</code></td>
<td></td>
<td>MIDP 1.0: Indicates equivalence.</td>
</tr>
<tr>
<td><code>int FOLLOWS</code></td>
<td></td>
<td>MIDP 1.0: Indicates follows.</td>
</tr>
<tr>
<td><code>int PRECEDES</code></td>
<td></td>
<td>MIDP 1.0: Indicates precedes.</td>
</tr>
</tbody>
</table>

### RecordEnumeration

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void destroy()</code></td>
<td></td>
<td>MIDP 1.0: Destroys the enumeration.</td>
</tr>
<tr>
<td><code>int getRecordId(int index)</code></td>
<td></td>
<td>MIDP 3.0: Returns the record ID of an element.</td>
</tr>
<tr>
<td><code>boolean hasNextElement()</code></td>
<td></td>
<td>MIDP 1.0: Determines if there is a next element.</td>
</tr>
<tr>
<td><code>boolean hasPreviousElement()</code></td>
<td></td>
<td>MIDP 1.0: Determines if there is a previous element.</td>
</tr>
<tr>
<td><code>boolean isKeptUpdated()</code></td>
<td></td>
<td>MIDP 1.0: Determines if the enumeration is kept updated.</td>
</tr>
<tr>
<td><code>void keepUpdated(boolean keepUpdated)</code></td>
<td></td>
<td>MIDP 1.0: Keeps the enumeration updated.</td>
</tr>
<tr>
<td><code>byte[] nextRecord()</code></td>
<td></td>
<td>MIDP 1.0: Returns the next record.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0 int nextRecordId()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0 int numRecords()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0 byte[] previousRecord()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0 int previousRecordId()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0 void rebuild()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0 void reset()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### RecordFilter

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 1.0 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>MIDP 1.0 void recordAdded(RecordStore, int recordId)</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0 void recordChanged(RecordStore, int recordId)</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0 void recordDeleted(RecordStore, int recordId)</td>
<td></td>
</tr>
</tbody>
</table>

### RecordStore

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 1.0 int addRecord(byte[] data, int offset, int numBytes)</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0 int addRecord(byte[] data, int offset, int numBytes, int tag)</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0 void addRecordListener(RecordListener listener)</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0 int AUTHMODE_ANY</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0 int AUTHMODE_APPLEVEL</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0 int AUTHMODE_PRIVATE</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0 void closeRecordStore()</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0 void deleteRecord(int recordId)</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0 void deleteRecordStore(String recordStoreName)</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0 RecordEnumeration enumerateRecords(RecordFilter, RecordComparator)</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0 RecordEnumeration enumerateRecords(RecordFilter, RecordComparator)</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0 RecordEnumeration enumerateRecords(RecordFilter, RecordComparator)</td>
<td></td>
</tr>
<tr>
<td>MIDP 3.0 void exportRecordStore(OutputStream, String recordStoreName, String internalPassword, String exportPassword)</td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>long</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>String</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>byte[]</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>RecordStoreInfo</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>RecordStore</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>String[]</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>RecordStore</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>RecordStore</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>RecordStore</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>RecordStore</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>RecordStore</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
</tbody>
</table>

**RecordStoreException**

`javax.microedition.rms`
### RecordStoreException

**MIDP 1.0**

- `RecordStoreException()`
- `RecordStoreException(String message)`

### RecordStoreFullException

**MIDP 1.0**

- `RecordStoreFullException()`
- `RecordStoreFullException(String message)`

### RecordStoreInfo

**MIDP 3.0**

- `int getAuthMode()`
- `long getSize()`
- `long getSizeAvailable()`
- `boolean isEncrypted()`
- `boolean isWriteable()`

### RecordStoreNotFoundException

**MIDP 1.0**

- `RecordStoreNotFoundException()`
- `RecordStoreNotFoundException(String message)`

### RecordStoreNotOpenException

**MIDP 1.0**

- `RecordStoreNotOpenException()`
### ScalableImage

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bind(Object extScalableImage)</td>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>getViewportHeight()</td>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>getViewportWidth()</td>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>setHeight(int height, boolean matchAspectRatio)</td>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>setWidth(int width, boolean matchAspectRatio)</td>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>unbind()</td>
<td>MIDP 3.0</td>
</tr>
</tbody>
</table>

### Screen

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getSecurityInfo()</td>
<td>MIDP 2.0</td>
</tr>
</tbody>
</table>

### SecureConnection

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getSecurityInfo()</td>
<td>MIDP 2.0</td>
</tr>
</tbody>
</table>

### SecureRecordStoreException

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SecureRecordStoreException()</td>
<td>MIDP 3.0</td>
</tr>
<tr>
<td>SecureRecordStoreException(String message)</td>
<td>MIDP 3.0</td>
</tr>
</tbody>
</table>

### SecurityInfo

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCipherSuite()</td>
<td>MIDP 2.0</td>
</tr>
<tr>
<td>getProtocolName()</td>
<td>MIDP 2.0</td>
</tr>
<tr>
<td>getProtocolVersion()</td>
<td>MIDP 2.0</td>
</tr>
<tr>
<td>getServerCertificate()</td>
<td>MIDP 2.0</td>
</tr>
</tbody>
</table>

### ServerSocketConnection

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### ServerSocketConnection

<table>
<thead>
<tr>
<th>MIDP 2.0</th>
<th>String</th>
<th>getLocalAddress()</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 2.0</td>
<td>int</td>
<td>getLocalPort()</td>
</tr>
</tbody>
</table>

### SocketConnection

<table>
<thead>
<tr>
<th>MIDP 2.0</th>
<th>byte</th>
<th>DELAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 2.0</td>
<td>String</td>
<td>getAddress()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>String</td>
<td>getLocalAddress()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int</td>
<td>getLocalPort()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int</td>
<td>getPort()</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>int</td>
<td>getSocketOption(byte option)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>byte</td>
<td>KEEPALIVE</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>byte</td>
<td>LINGER</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>byte</td>
<td>RCVBUF</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>void</td>
<td>setSocketOption(byte option, int value)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>byte</td>
<td>SNDBUF</td>
</tr>
</tbody>
</table>

### Spacer

<table>
<thead>
<tr>
<th>MIDP 2.0</th>
<th>void</th>
<th>addCommand(Command cmd)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 2.0</td>
<td>void</td>
<td>setDefaultCommand(Command cmd)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>void</td>
<td>setLabel(String label)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>void</td>
<td>setMinimumSize(int minWidth, int minHeight)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>Spacer(int minWidth, int minHeight)</td>
<td></td>
</tr>
</tbody>
</table>

### Sprite

<table>
<thead>
<tr>
<th>MIDP 2.0</th>
<th>boolean</th>
<th>collidesWith(Image image, int x, int y, boolean pixelLevel)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 2.0</td>
<td>boolean</td>
<td>collidesWith(Sprite s, boolean pixelLevel)</td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td>boolean</td>
<td>collidesWith(TiledLayer t, boolean pixelLevel)</td>
</tr>
<tr>
<td>Method</td>
<td>Signature</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>defineCollisionRectangle</td>
<td>void defineCollisionRectangle(int x, int y, int width, int height)</td>
<td></td>
</tr>
<tr>
<td>defineReferencePixel</td>
<td>void defineReferencePixel(int x, int y)</td>
<td></td>
</tr>
<tr>
<td>getFrame</td>
<td>int getFrame()</td>
<td></td>
</tr>
<tr>
<td>getFrameSequenceLength</td>
<td>int getFrameSequenceLength()</td>
<td></td>
</tr>
<tr>
<td>getRawFrameCount</td>
<td>int getRawFrameCount()</td>
<td></td>
</tr>
<tr>
<td>getRefPixelX</td>
<td>int getRefPixelX()</td>
<td></td>
</tr>
<tr>
<td>getRefPixelY</td>
<td>int getRefPixelY()</td>
<td></td>
</tr>
<tr>
<td>nextFrame</td>
<td>void nextFrame()</td>
<td></td>
</tr>
<tr>
<td>paint</td>
<td>void paint(Graphics g)</td>
<td></td>
</tr>
<tr>
<td>prevFrame</td>
<td>void prevFrame()</td>
<td></td>
</tr>
<tr>
<td>setFrame</td>
<td>void setFrame(int sequenceIndex)</td>
<td></td>
</tr>
<tr>
<td>setFrameSequence</td>
<td>void setFrameSequence(int[] sequence)</td>
<td></td>
</tr>
<tr>
<td>setImage</td>
<td>void setImage(Image img, int frameWidth, int frameHeight)</td>
<td></td>
</tr>
<tr>
<td>setRefPixelPosition</td>
<td>void setRefPixelPosition(int x, int y)</td>
<td></td>
</tr>
<tr>
<td>setTransform</td>
<td>void setTransform(int transform)</td>
<td></td>
</tr>
<tr>
<td>Sprite(Image image)</td>
<td>Sprite(Image image)</td>
<td></td>
</tr>
<tr>
<td>Sprite(Image image, int frameWidth, int frameHeight)</td>
<td>Sprite(Image image, int frameWidth, int frameHeight)</td>
<td></td>
</tr>
<tr>
<td>Sprite(Sprite s)</td>
<td>Sprite(Sprite s)</td>
<td></td>
</tr>
<tr>
<td>TRANS_MIRROR</td>
<td>int TRANS_MIRROR</td>
<td></td>
</tr>
<tr>
<td>TRANS_MIRROR_ROT180</td>
<td>int TRANS_MIRROR_ROT180</td>
<td></td>
</tr>
<tr>
<td>TRANS_MIRROR_ROT270</td>
<td>int TRANS_MIRROR_ROT270</td>
<td></td>
</tr>
<tr>
<td>TRANS_MIRROR_ROT90</td>
<td>int TRANS_MIRROR_ROT90</td>
<td></td>
</tr>
<tr>
<td>TRANS_NONE</td>
<td>int TRANS_NONE</td>
<td></td>
</tr>
<tr>
<td>TRANS_ROT180</td>
<td>int TRANS_ROT180</td>
<td></td>
</tr>
<tr>
<td>TRANS_ROT270</td>
<td>int TRANS_ROT270</td>
<td></td>
</tr>
<tr>
<td>TRANS_ROT90</td>
<td>int TRANS_ROT90</td>
<td></td>
</tr>
<tr>
<td>getAppearanceMode</td>
<td>int getAppearanceMode()</td>
<td></td>
</tr>
</tbody>
</table>
### MIDP 2.0

- **Font**
  - `setFont()`

- **String**
  - `getText()`

### MIDP 1.0

- **void**
  - `setFont(Font font)`
  - `setText(String text)`

### MIDP 1.0.

- **StringItem**
  - `StringItem(String label,String text)`

### MIDP 2.0

- **StringItem**
  - `StringItem(String label,String text,int appearanceMode)`

---

#### TabbedPane

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void</code></td>
<td><code>addTab(Screen tab,Image icon)</code></td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>addTabListener(TabListener tabListener)</code></td>
</tr>
<tr>
<td><code>int</code></td>
<td><code>getCount()</code></td>
</tr>
<tr>
<td><code>int</code></td>
<td><code>getHeight()</code></td>
</tr>
<tr>
<td><code>Screen</code></td>
<td><code>getScreen(int index)</code></td>
</tr>
<tr>
<td><code>int</code></td>
<td><code>getSelectedIndex()</code></td>
</tr>
<tr>
<td><code>Image</code></td>
<td><code>getTabIcon(int index)</code></td>
</tr>
<tr>
<td><code>int</code></td>
<td><code>getWidth()</code></td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>insertTab(int index,Screen tab,Image icon)</code></td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>removeTab(int index)</code></td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>setFocus(int index)</code></td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>setTabIcon(int index,Image icon)</code></td>
</tr>
<tr>
<td><code>❉</code></td>
<td><code>TabbedPane(String title,boolean stringTab,boolean suppressTitle)</code></td>
</tr>
</tbody>
</table>

---

#### TableLayoutPolicy

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>❉</code></td>
<td><code>doLayout(int viewportX,int viewportY,int viewportWidth,int viewportHeight,int[] totalSize)</code></td>
</tr>
<tr>
<td><code>int</code></td>
<td><code>getColumns()</code></td>
</tr>
<tr>
<td><code>❉</code></td>
<td><code>Item getTraverse(Item item,int dir)</code></td>
</tr>
<tr>
<td><code>❉</code></td>
<td><code>TableLayoutPolicy(Form form,int columns)</code></td>
</tr>
</tbody>
</table>
### TabListener

<table>
<thead>
<tr>
<th>MIDP 3.0</th>
<th>void tabAddedEvent(int index, Screen tab)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 3.0</td>
<td>void tabChangeEvent(Screen tab)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void tabRemovedEvent(int index)</td>
</tr>
</tbody>
</table>

### Text

<table>
<thead>
<tr>
<th>MIDP 3.0</th>
<th>int ALIGN_CENTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDP 3.0</td>
<td>int ALIGN_DEFAULT</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int ALIGN_JUSTIFY</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int ALIGN_LEFT</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int ALIGN_RIGHT</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void delete(int index, int length)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int DIRECTION_LTR</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int DIRECTION_NEUTRAL</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int DIRECTION_RTL</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int getAlignment()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int getBackgroundColor()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int getCaret()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void getCharExtent(int index, int[] extent)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int getCharIndex(int x, int y)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>Font getFont()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>Font getFont(int index)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int getForegroundColor()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int getForegroundColor(int index)</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int getHeight()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int getHighlightIndex()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int getHighlightLength()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int getIndent()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int getInitialDirection()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int getRequiredHeight()</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>String</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>int</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>void</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>*</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>*</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>*</td>
</tr>
<tr>
<td>MIDP 3.0</td>
<td>boolean</td>
</tr>
</tbody>
</table>

**TextBox**

`javax.microedition.lcdui`
| MIDP 1.0 | void delete(int offset, int length) |
| MIDP 1.0 | int getCaretPosition() |
| MIDP 1.0 | int getChars(char[] data) |
| MIDP 1.0 | int getConstraints() |
| MIDP 3.0 | int getHeight() |
| MIDP 1.0 | int getMaxSize() |
| MIDP 1.0 | String getString() |
| MIDP 3.0 | int getWidth() |
| MIDP 1.0 | void insert(char[] data, int offset, int length, int position) |
| MIDP 1.0 | void insert(String src, int position) |
| MIDP 3.0 | void setCaret(int index) |
| MIDP 1.0 | void setChars(char[] data, int offset, int length) |
| MIDP 1.0 | void setConstraints(int constraints) |
| MIDP 3.0 | void setHighlight(int index, int length) |
| MIDP 2.0 | void setInitialInputMode(String characterSubset) |
| MIDP 1.0 | int setMaxSize(int maxSize) |
| MIDP 1.0 | void setString(String text) |
| MIDP 1.0 | int size() |
| MIDP 1.0 | * TextBox(String title, String text, int maxSize, int constraints) |

<p>| TextEditor | javax.microedition.lcdui |
| Object | CanvasItem |
| TextEditor | |
| 3.0 | void delete(int offset, int length) |
| 3.0 | int getBackgroundColor() |
| 3.0 | int getCaretPosition() |
| 3.0 | int getConstraints() |
| 3.0 | int getContentHeight() |
| 3.0 | boolean getFocus() |
| 3.0 | Font getFont() |</p>
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int getForegroundColor()</td>
<td></td>
</tr>
<tr>
<td>int getHighlightColor()</td>
<td></td>
</tr>
<tr>
<td>int getLineMarginHeight()</td>
<td></td>
</tr>
<tr>
<td>int getMaxSize()</td>
<td></td>
</tr>
<tr>
<td>String getSelection()</td>
<td></td>
</tr>
<tr>
<td>String getString()</td>
<td></td>
</tr>
<tr>
<td>boolean getVisible()</td>
<td></td>
</tr>
<tr>
<td>int getVisibleContentPosition()</td>
<td></td>
</tr>
<tr>
<td>void insert(String text,int position)</td>
<td></td>
</tr>
<tr>
<td>void setBackgroundColor(int alpha,int red,int green,int blue)</td>
<td></td>
</tr>
<tr>
<td>void setCaretPosition(int index)</td>
<td></td>
</tr>
<tr>
<td>void setConstraints(int constraints)</td>
<td></td>
</tr>
<tr>
<td>void setFocus(boolean focused)</td>
<td></td>
</tr>
<tr>
<td>void setFont(Font font)</td>
<td></td>
</tr>
<tr>
<td>void setForegroundColor(int alpha,int red,int green,int blue)</td>
<td></td>
</tr>
<tr>
<td>void setHighlightColor(int alpha,int red,int green,int blue)</td>
<td></td>
</tr>
<tr>
<td>void setInitialInputMode(String characterSubset)</td>
<td></td>
</tr>
<tr>
<td>int setMaxSize(int maxSize)</td>
<td></td>
</tr>
<tr>
<td>void setSelection(int index,int length)</td>
<td></td>
</tr>
<tr>
<td>void setString(String text)</td>
<td></td>
</tr>
<tr>
<td>void setTextEditorListener(TextEditorChangeListener listener)</td>
<td></td>
</tr>
<tr>
<td>void setVisible(boolean visible)</td>
<td></td>
</tr>
<tr>
<td>int size()</td>
<td></td>
</tr>
</tbody>
</table>

```java
TextEditor(String text,int maxSize,int constraints,int width,int height)
```

```java
TextEditorChangeListener
```

```java
javax.microedition.lcdui
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int ACTION_CARET_MOVE</td>
<td></td>
</tr>
<tr>
<td>int ACTION_CONTENT_CHANGE</td>
<td></td>
</tr>
<tr>
<td>int ACTION_DIRECTION_CHANGE</td>
<td></td>
</tr>
<tr>
<td>int ACTION_INPUT_MODE_CHANGE</td>
<td></td>
</tr>
<tr>
<td>int ACTION_LANGUAGE_CHANGE</td>
<td></td>
</tr>
<tr>
<td>int ACTION_TRAVERSE_NEXT</td>
<td></td>
</tr>
</tbody>
</table>
### TextField

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>MIDP 1.0</th>
<th>MIDP 2.0</th>
<th>MIDP 3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>int ACTION_TRAVERSE_PREVIOUS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>void inputAction(TextEditor textEditor, int actions)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int ANY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int CONSTRAINT_MASK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int CURRENCY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int DECIMAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>void delete(int offset, int length)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int EMAILADDR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int getCaretposition()</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int getChars(char[] data)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int getConstraints()</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int getMaxSize()</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>String getString()</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int INITIAL_CAPS_SENTENCE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int INITIAL_CAPS_WORD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>void insert(char[] data, int offset, int length, int position)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>void insert(String src, int position)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int NON_PREDICTIVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int NUMERIC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int PASSWORD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int PHONENUMBER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int SENSITIVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>void setCaret(int index)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>void setChars(char[] data, int offset, int length)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>void setConstraints(int constraints)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>void setHighlight(int index, int length)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>void setInitialInputMode(String characterSubset)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int setMaxSize(int maxSize)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>void</td>
<td>setString(String text)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>int</td>
<td>size()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td>*</td>
<td>TextField(String label,String text,int maxSize,int constraints)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIDP 2.0</td>
<td></td>
<td>int UNEDITABLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIDP 1.0</td>
<td></td>
<td>int URL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Ticker

**Object**

- ➩ Ticker

| MIDP 1.0 | String | getString() |
| MIDP 1.0 | void | setString(String str) |
| MIDP 1.0 | * | Ticker(String str) |

### TiledLayer

**Object**

- ➩ Layer
- ➩ TiledLayer

| MIDP 2.0 | int | createAnimatedTile(int staticTileIndex) |
| MIDP 2.0 | void | fillCells(int col,int row,int numCols,int numRows,int tileIndex) |
| MIDP 2.0 | int | getAnimatedTile(int animatedTileIndex) |
| MIDP 2.0 | int | getCell(int col,int row) |
| MIDP 2.0 | • | getCellHeight() |
| MIDP 2.0 | • | getCellWidth() |
| MIDP 2.0 | • | getColumns() |
| MIDP 2.0 | • | getRows() |
| MIDP 2.0 | • | paint(Graphics g) |
| MIDP 2.0 | void | setAnimatedTile(int animatedTileIndex,int staticTileIndex) |
| MIDP 2.0 | void | setCell(int col,int row,int tileIndex) |
| MIDP 2.0 | void | setStaticTileSet(Image image,int tileWidth,int tileHeight) |
| MIDP 2.0 | * | TiledLayer(int columns,int rows,Image image,int tileWidth,int tileHeight) |

### UDPEndpointConnection

**UDPEndpointConnection**

| MIDP 2.0 | String | getLocalAddress() |
int getLocalPort()
Index

#  StartOfMIDP3Javadoc 16

A

Abstract Commands 191
Accept Header 55
Accept-Language Header 55
ACTION_CARET_MOVE
  of javax.microedition.lcdui.TextEditorChangeListener 595
ACTION_CONTENT_CHANGE
  of javax.microedition.lcdui.TextEditorChangeListener 595
ACTION_DIRECTION_CHANGE
  of javax.microedition.lcdui.TextEditorChangeListener 595
ACTION_INPUT_MODE_CHANGE
  of javax.microedition.lcdui.TextEditorChangeListener 595
ACTION_LANGUAGE_CHANGE
  of javax.microedition.lcdui.TextEditorChangeListener 595
ACTION_TRAVERSE_NEXT
  of javax.microedition.lcdui.TextEditorChangeListener 595
ACTION_TRAVERSE_PREVIOUS
  of javax.microedition.lcdui.TextEditorChangeListener 595
ACTIONS_ALL
  of javax.microedition.lcdui.Canvas 230
ACTIONS_NAVIGATION
  of javax.microedition.lcdui.Canvas 230
ACTIONS_NONE
  of javax.microedition.lcdui.Canvas 231
ActionsDeniedPermission
  of javax.microedition.midlet 674
ActionsDeniedPermission 674
ActionsDeniedPermission()
  of javax.microedition.midlet 674
Activation of Idle Screen MIDlets 200
addCommand(Command)
  of javax.microedition.lcdui.FileSelector 363
  of javax.microedition.lcdui.Displayable 346
  of javax.microedition.lcdui.Item 483
  of javax.microedition.lcdui.Alert 210
  of javax.microedition.lcdui.Spacer 539
addDisplayListener(DisplayListener)
  of javax.microedition.lcdui.Display 330
addedToDisplay(Display)
  of javax.microedition.lcdui.IdleItem 448
addEventListener(String, EventDataListener, boolean)
  of javax.microedition.event.EventManager 782
addEventListener(String, EventDataListener, boolean, boolean)
  of javax.microedition.event.EventManager 782
addEventListener(String, EventDataListener, boolean, double, double)
  of javax.microedition.event.EventManager 782
addEventListener(String, EventDataListener, boolean, long, long)
  of javax.microedition.event.EventManager 783
addEventListener(String, EventDataListener, boolean, String[])
  of javax.microedition.event.EventManager 784
addRecord(byte[], int, int)
  of javax.microedition.rms.RecordStore 729
addRecord(byte[], int, int, int)
  of javax.microedition.rms.RecordStore 729
addRecordListener(RecordListener)
  of javax.microedition.rms.RecordStore 730
addTab(Screen, Image)
  of javax.microedition.lcdui.TabbedPane 548
addTabListener(TabListener)
  of javax.microedition.lcdui.TabbedPane 548
alac_examples#ALAC1 800
alac_examples#ALAC2 801
alac_examples#ALAC3 802
alac_examples#ALAC4 803
alac_examples#alac_examples 800
alac_examples#Figure C-1 : Application Level Access Authorization Example 1 800
alac_examples#Figure C-2 : Application Level Access Authorization Example 2 802
alac_examples#Figure C-3 : Application Level Access Authorization Example 3 803
alac_examples#Figure C-4 : Application Level Access Authorization Example 4 804
alac_examples#figurec-1 800
alac_examples#figurec-2 802
alac_examples#figurec-4 804
ALARM
  of javax.microedition.lcdui.AlertType 216
ALERT
  of javax.microedition.lcdui.Display 321
ALERT 206
  of javax.microedition.lcdui 206
Alert#commands
    of javax.microedition.lcdui 206
Alert#indicator
    of javax.microedition.lcdui 206
Alert(String)
    of javax.microedition.lcdui 209
Alert(String, String, Image, AlertType)
    of javax.microedition.lcdui 210
AlertType 215
AlertType()
    of javax.microedition.lcdui 215
Algorithm Support 691
ALIGN_CENTER
    of javax.microedition.lcdui.Text 562
ALIGN_DEFAULT
    of javax.microedition.lcdui.Text 562
ALIGN_JUSTIFY
    of javax.microedition.lcdui.Text 562
ALIGN_LEFT
    of javax.microedition.lcdui.Text 562
ALIGN_RIGHT
    of javax.microedition.lcdui.Text 562
AnimatedImage 218
AnimatedImage
    of javax.microedition.lcdui 218
ANY
    of javax.microedition.lcdui.TextField 603
API Overview 615
append(Command)
    of javax.microedition.lcdui.Menu 513
append(Image)
    of javax.microedition.lcdui.Form 394
append(Item)
    of javax.microedition.lcdui.Form 394
append(Layer)
    of javax.microedition.lcdui.game.LayerManager 631
append(Menu)
    of javax.microedition.lcdui.Menu 514
append(String)
    of javax.microedition.lcdui.Form 394
append(String, Image)
    of javax.microedition.lcdui.List 502
    of javax.microedition.lcdui.ChoiceGroup 265
    of javax.microedition.lcdui.Choice 255
Application Attributes 794
Application Attributes 24
Application Attributes Compatibility 105
Application Implementation Notes 671
Application Level Access Authorization 71
Application Level Access Authorization Examples 800
Application Lifecycle 667
Application Resource Files 110
APPLICATION_RELAUNCH_PREFIX
    of javax.microedition.event.EventData 761
Architecture 9
Assumptions 64
Atomicity of RecordStore Access 703
Attribute Overrides in Application Descriptor 105
AUDIO_MUTE
    of javax.microedition.event.EventData 761
Authenticating a MIDlet Suite 79
AUTHMODE_ANY
    of javax.microedition.rms.RecordStore 728
AUTHMODE_APPLEVEL
    of javax.microedition.rms.RecordStore 729
AUTHMODE_PRIVATE
    of javax.microedition.rms.RecordStore 729
Authorization Model 64
Auto Start MIDlets 666
AutoStartPermission 676
AutoStartPermission
    of javax.microedition.midlet 676
AutoStartPermission()
    of javax.microedition.midlet 676

B
BACK
    of javax.microedition.lcdui.Command 276
BACKLIGHT
    of javax.microedition.event.EventData 761
BACKLIGHT_DIM
    of javax.microedition.event.EventData 761
BACKLIGHT_OFF
    of javax.microedition.event.EventData 761
BACKLIGHT_ON
    of javax.microedition.event.EventData 762
BAD_EXTENSIONS
    of javax.microedition.pki.CertificateException 698
BASELINE
    of javax.microedition.lcdui.Graphics 424
BATTERY_CHARGING
    of javax.microedition.event.EventData 762
BATTERY_LEVEL
    of javax.microedition.event.EventData 762
BATTERY_LOW
    of javax.microedition.event.EventData 762
bind(Object)
Combined usage of MIDP 2.0 and MIDP 3.0 security models 75
COMMAND
of javax.microedition.lcdui.Display 324
of javax.microedition.lcdui 271
COMMAND 271
Command#enabled
of javax.microedition.lcdui 273
Command#exact_placement
of javax.microedition.lcdui 273
Command#image
of javax.microedition.lcdui 273
Command#label
of javax.microedition.lcdui 271
Command#priority
of javax.microedition.lcdui 272
Command#type
of javax.microedition.lcdui 272
Command(String, int, int)
of javax.microedition.lcdui 277
Command(String, String, Image, int, int)
of javax.microedition.lcdui 278
Command(String, String, int, int)
of javax.microedition.lcdui 278
commandAction(Command, Displayable)
of javax.microedition.lcdui.CommandListener 285
commandAction(Command, Item)
of javax.microedition.lcdui.ItemCommandListener 489
CommandLayoutPolicy
of javax.microedition.lcdui 282
CommandLayoutPolicy 282
CommandListener 285
CommandListener
of javax.microedition.lcdui 285
CommConnection 120
CommConnection
of javax.microedition.io 120
compare(byte[], byte[])
of javax.microedition.rms.RecordComparator 717
concurrency#Figure 5-1 : Data Isolation Between Concurrent MIDlets 60
concurrency#Figure 5-2 : Concurrent MIDlet Class and Data Separation 60
concurrency#MIDP 3.0 MIDlet Concurrency 59
CONFIRMATION
of javax.microedition.lcdui.AlertType 216
CONSTRAINT_MASK
of javax.microedition.lcdui.TextField 604
CONTINUOUS_IDE
of javax.microedition.lcdui.Gauge 408
CONTINUOUS_RUNNING
of javax.microedition.lcdui.Gauge 408
Contributors 6
Controlling User actions on a MIDlet Suite 667
Coordinate System 198
copyArea(int, int, int, int, int, int, int)
of javax.microedition.lcdui.Graphics 426
createAnimatedTile(int)
of javax.microedition.lcdui.game.TiledLayer 654
createFont(InputStream)
of javax.microedition.lcdui.Font 375
createImage(byte[], int, int)
of javax.microedition.lcdui.Image 455
createlmage(Image)
of javax.microedition.lcdui.Image 455
createlmage(Image, int, int, int, int, int)
of javax.microedition.lcdui.Image 456
createlmage(Image, int, int, int, int, int, int)
of javax.microedition.lcdui.Image 457
createlmage(Image, int, int, int, int, int, int, int)
of javax.microedition.lcdui.Image 458
createlmage(Image, int, int, int, int, int, int, int, int)
of javax.microedition.lcdui.Image 459
createlmage(Image, int, int, int, int, int, int, int, int, int)
of javax.microedition.lcdui.Image 460
CURRENCY
of javax.microedition.lcdui.TextField 604
CustomItem 286

Mobile Information Device Profile v3.0 - JSR 271
© Copyright Motorola, Inc. 2002-2009. All Rights Reserved.
CustomItem
  of javax.microedition.lcdui 286

CustomItem#interaction
  of javax.microedition.lcdui 286

CustomItem#keypad
  of javax.microedition.lcdui 287

CustomItem#pointer
  of javax.microedition.lcdui 287

CustomItem(String)
  of javax.microedition.lcdui 293

Data Isolation 60

DATA_NETWORK
  of javax.microedition.event.EventData 762

DATE
  of javax.microedition.lcdui.DateField 305

DATE_TIME
  of javax.microedition.lcdui.DateField 305

DateField
  of javax.microedition.lcdui 304

DateField
  of javax.microedition.lcdui 304

DateField(String, int)
  of javax.microedition.lcdui 306

DateField(String, int, TimeZone)
  of javax.microedition.lcdui 306

DECIMAL
  of javax.microedition.lcdui.TextField 604

Declaring Application Level Access Authorization 72

defineCollisionRectangle(int, int, int, int)
  of javax.microedition.lcdui.Sprite 644

defineReferencePixel(int, int)
  of javax.microedition.lcdui.Sprite 645

Definition of Terms 77

DELAY
  of javax.microedition.io.SocketConnection 182

DELETE
  of javax.microedition.io.HttpConnection 136

delete(int)
  of javax.microedition.lcdui.ChoiceGroup 265
  of javax.microedition.lcdui.Form 394
  of javax.microedition.lcdui.List 502
  of javax.microedition.lcdui.Choice 255

delete(int, int)
  of javax.microedition.lcdui.TextBox 576
  of javax.microedition.lcdui.Text 564
  of javax.microedition.lcdui.TextEditEditor 585
  of javax.microedition.lcdui.TextField 608

deleteAll()
  of javax.microedition.lcdui.Choice 256
  of javax.microedition.lcdui.ChoiceGroup 265
  of javax.microedition.lcdui.Form 395
  of javax.microedition.lcdui.List 502

deleteRecord(int)
  of javax.microedition.rms.RecordStore 730

deleteRecordStore(String)
  of javax.microedition.rms.RecordStore 731

Deletion 48

Dependency Attributes 33

Dependency Declaration Chain 35

Dependency Expression 33

Dependency Expression Example 35

Dependency Names For Standard APIs 37

deriveFont(int)
  of javax.microedition.lcdui.Font 376

deriveFont(int, int)
  of javax.microedition.lcdui.Font 376

destroy()
  of javax.microedition.rms.RecordEnumeration 719

destroyApp(boolean)
  of javax.microedition.midlet.MIDlet 680

Developer Recommendations for Dealing With
Overlapping Configuration APIs 57

Device Identification and Request Headers 51

Device Identifier System Properties 108

Device Requirements 10

Device-Provided Operations 192

DIRECTION_LTR
  of javax.microedition.lcdui.Text 563
  of javax.microedition.lcdui.FormLayoutPolicy 400

DIRECTION_NEUTRAL
  of javax.microedition.lcdui.Text 563

DIRECTION_RTL
  of javax.microedition.lcdui.FormLayoutPolicy 400
  of javax.microedition.lcdui.Text 563

DIRECTORY_CREATE
  of javax.microedition.lcdui.FileSelector 361

DIRECTORY_SELECT
  of javax.microedition.lcdui.FileSelector 361

Discovery 40

DISMISS_COMMAND
  of javax.microedition.lcdui.Alert 209

Display 308

Display
  of javax.microedition.lcdui 308

Display#colorscheme
  of javax.microedition.lcdui 312

Display#displaycapabilities
  of javax.microedition.lcdui 310

Display#displayresources
of javax.microedition.lcdui 308
Display#displaystates
of javax.microedition.lcdui 308
Display#exactplacement
of javax.microedition.lcdui 312
Display#hardwarestates
of javax.microedition.lcdui 308
Display#obtainingdisplayobjects
of javax.microedition.lcdui 310
Display#showingdisplayables
of javax.microedition.lcdui 311
Display#systemscreens
of javax.microedition.lcdui 311
DISPLAY_HARDWARE_ABSENT
of javax.microedition.lcdui.Display 324
DISPLAY_HARDWARE_DISABLED
of javax.microedition.lcdui.Display 324
DISPLAY_HARDWARE_ENABLED
of javax.microedition.lcdui.Display 324
Displayable
of javax.microedition.lcdui 344
Displayable
of javax.microedition.lcdui 344
Displayable#placement
of javax.microedition.lcdui 344
displayAdded(Display)
of javax.microedition.lcdui.DisplayListener 354
DisplayCapabilityException 352
DisplayCapabilityException
of javax.microedition.lcdui 352
DisplayCapabilityException()
of javax.microedition.lcdui 352
DisplayCapabilityException(String)
of javax.microedition.lcdui 352
DisplayListener 354
DisplayListener
of javax.microedition.lcdui 354
displayStateChanged(Display, int)
of javax.microedition.lcdui.DisplayListener 354
Document Conventions 3
doLayout(int, int, int, int, int[])
of javax.microedition.lcdui.FormLayoutPolicy 400
of javax.microedition.lcdui.TableLayoutPolicy 554
domain_policies#domain_policies 797
DOTTED
of javax.microedition.lcdui.Graphics 424
DOWN
of javax.microedition.lcdui.Canvas 231
DOWN_PRESSED
of javax.microedition.lcdui.game.GameCanvas 621
Downloadable Fonts 198
drawArc(int, int, int, int, int)
of javax.microedition.lcdui.Graphics 427
drawARGB16(short[], int, int, int, int, int)
of javax.microedition.lcdui.Graphics 427
drawChar(char, int, int)
of javax.microedition.lcdui.Graphics 428
drawChars(char[], int, int, int, int)
of javax.microedition.lcdui.Graphics 429
drawImage(Image, int, int, int)
of javax.microedition.lcdui.Graphics 429
drawLine(int, int, int, int)
of javax.microedition.lcdui.Graphics 430
drawRect(int, int, int, int)
of javax.microedition.lcdui.Graphics 430
drawRegion(Image, int, int, int, int, int, int, int, int, int)
of javax.microedition.lcdui.Graphics 430
drawRegion(Image, int, int, int, int, int, int, int, int, int, int, int)
of javax.microedition.lcdui.Graphics 432
drawRGB(int[], int, int, int, int, int, boolean)
of javax.microedition.lcdui.Graphics 433
drawRGB16(short[], int, int, int, int, int)
of javax.microedition.lcdui.Graphics 434
drawRoundRect(int, int, int, int, int)
of javax.microedition.lcdui.Graphics 435
drawString(String, int, int, int)
of javax.microedition.lcdui.Graphics 436
drawSubstring(String, int, int, int, int, int)
of javax.microedition.lcdui.Graphics 436
drawText(Text, int, int)
of javax.microedition.lcdui.Graphics 436
E
EMAIL
of javax.microedition.lcdui.NotificationType 530
EMAILADDR
of javax.microedition.lcdui.TextField 605
enumerateRecords(RecordFilter, RecordComparator, boolean)
of javax.microedition.rms.RecordStore 731
enumerateRecords(RecordFilter, RecordComparator, boolean, int[])
of javax.microedition.rms.RecordStore 732
equals(Object)
of javax.microedition.event.EventPermission 792
of javax.microedition.event.EventData 769
of javax.microedition.io.PushRegistryPermission 170
of javax.microedition.lcdui.Font 377
EQUIVALENT
ERROR
  of javax.microedition.lcdui.AlertType 216
Error Handling 61
event
  of javax.microedition 754
Event Handling 190
EventData 756
EventData
  of javax.microedition.event 756
EventData#validinfotypes
  of javax.microedition.event 756
EventData(String, boolean, String, Object)
  of javax.microedition.event 768
EventData(String, double, String, Object)
  of javax.microedition.event 768
EventData(String, long, String, Object)
  of javax.microedition.event 768
EventData(String, String, String, Object)
  of javax.microedition.event 769
EventDataListener
  of javax.microedition.event 774
EventDataListener 774
EventManager
  of javax.microedition.event 775
EventManager 775
EventManager#ApplicationLaunch
  of javax.microedition.event 776
EventPermission
  of javax.microedition.event 791
EventPermission 791
EventPermission(String, String)
  of javax.microedition.event 792
Example 1 - Accessing Shared Record Store 800
Example 2 - IMC 801
Example 3 - IMC 802
Example 4 - Events 803
Example Mapping Named Permissions to Class
Permissions 104
Example MIDlet Application 671
Example of Granting Permissions to MIDlet Suites 69
Example: HTTP Request for Application Descriptor
  55
Example: HTTP Request to Install/Update a MIDlet
  Suite 55
Example: Install Status via HTTP Post Request 56
Examples 703
EXCLUSIVE
  of javax.microedition.lcdui.Choice 254
Execution Of More Than One Instance Of A MIDlet 62
EXIT
  of javax.microedition.lcdui.Command 276
EXPIRED
  of javax.microedition.pki.CertificateException 698
exportRecordStore(OutputStream, String, String,
  String)
  of javax.microedition.rms.RecordStore 733
External Domain Policy Format 70
EXTERNAL_POWER
  of javax.microedition.event.EventData 763
F
FACE_MONOSPACE
  of javax.microedition.lcdui.Font 372
FACE_PROPORTIONAL
  of javax.microedition.lcdui.Font 373
FACE_SYSTEM
  of javax.microedition.lcdui.Font 373
FileSelector
  of javax.microedition.lcdui.lcdui 356
FileSelector 356
FileSelector(String, int)
  of javax.microedition.lcdui.lcdui 362
fillArc(int, int, int, int, int)
  of javax.microedition.lcdui.Graphics 437
fillCells(int, int, int, int)
  of javax.microedition.lcdui.game.TiledLayer 654
fillRect(int, int, int, int)
  of javax.microedition.lcdui.Graphics 437
fillRoundRect(int, int, int, int, int, int)
  of javax.microedition.lcdui.Graphics 438
fillTriangle(int, int, int, int, int, int)
  of javax.microedition.lcdui.Graphics 438
FIRE
  of javax.microedition.lcdui.lcdui.Canvas 231
FIRE_PRESSED
  of javax.microedition.lcdui.game.GameCanvas 621
flashBacklight(int)
  of javax.microedition.lcdui.Display 332
FLIGHT_MODE
  of javax.microedition.event.EventData 763
flushGraphics()
  of javax.microedition.lcdui.game.GameCanvas 623
flushGraphics(int, int, int)
  of javax.microedition.lcdui.game.GameCanvas 623
FOLLOWS
  of javax.microedition.rms.RecordComparator 717
Font 367
Font
  of javax.microedition.lcdui 367
Font Support 198
Font#charactersandglyphs
of javax.microedition.lcdui 367
Font#fontfacesandnames
of javax.microedition.lcdui 368
Font#fontsizes
of javax.microedition.lcdui 369
Font#physicalandlogicalfonts
of javax.microedition.lcdui 367
FONT_IDLE_HIGHLIGHTED_TEXT
of javax.microedition.lcdui.Font 373
FONT_IDLE_TEXT
of javax.microedition.lcdui.Font 373
FONT_INPUT_TEXT
of javax.microedition.lcdui.Font 373
FONT_STATIC_TEXT
of javax.microedition.lcdui.Font 373
FontFormatException 386
FontFormatException
of javax.microedition.lcdui 386
FontFormatException()
of javax.microedition.lcdui 386
FontFormatException(String)
of javax.microedition.lcdui 386
FOREVER
of javax.microedition.lcdui.Alert 209
Form 388
Form
of javax.microedition.lcdui 388
Form#layout
of javax.microedition.lcdui 388
Form#layout_expand
of javax.microedition.lcdui 390
Form#layout_shrink
of javax.microedition.lcdui 390
Form#layout_vertical
of javax.microedition.lcdui 390
Form#layoutdefault
of javax.microedition.lcdui 389
Form#linebreak
of javax.microedition.lcdui 391
Form(String)
of javax.microedition.lcdui 393
Form(String, Item[])
of javax.microedition.lcdui 393
FormLayoutPolicy 398
FormLayoutPolicy
of javax.microedition.lcdui 398
FormLayoutPolicy(Form)
of javax.microedition.lcdui 400

Functional Requirements 39

G

game
of javax.microedition.lcdui 615
GAME_A
of javax.microedition.lcdui.Canvas 231
GAME_A_PRESSED
of javax.microedition.lcdui.game.GameCanvas 621
GAME_B
of javax.microedition.lcdui.Canvas 231
GAME_B_PRESSED
of javax.microedition.lcdui.game.GameCanvas 621
GAME_C
of javax.microedition.lcdui.Canvas 231
GAME_C_PRESSED
of javax.microedition.lcdui.game.GameCanvas 621
GAME_D
of javax.microedition.lcdui.Canvas 231
GAME_D_PRESSED
of javax.microedition.lcdui.game.GameCanvas 621
GameCanvas 618
GameCanvas
of javax.microedition.lcdui.game 618
GameCanvas(boolean)
of javax.microedition.lcdui.game 622
GameCanvas(boolean, boolean)
of javax.microedition.lcdui.game 622
Gauge 405
Gauge
of javax.microedition.lcdui 405
Gauge(String, boolean, int, int)
of javax.microedition.lcdui 409

General Guidelines for MIDlet Behavior on Different
Execution Environments 57

GET
of javax.microedition.io.HttpConnection 136
get(int)
of javax.microedition.lcdui.Form 395
getActions()
of javax.microedition.event.EventPermission 792
of javax.microedition.io.PushRegistryPermission 170
getActivityMode()
of javax.microedition.lcdui.Display 332
getAddress()
of javax.microedition.io.SocketConnection 183
getAlignment()
of javax.microedition.lcdui.Text 564
getAlpha()
of javax.microedition.lcdui.Graphics 438
getAlphaColor()
of javax.microedition.lcdui.Graphics 438
getAltText()
of javax.microedition.lcdui.ImageItem 472
getAnimatedTile(int)
of javax.microedition.lcdui.game.TiledLayer 654
getAppearanceMode()
of javax.microedition.lcdui.ImageItem 472
of javax.microedition.lcdui.StringItem 543
getAppProperty(String)
of javax.microedition.midlet.MIDlet 680
getAppProperty(String, String, String, String)
of javax.microedition.midlet.MIDlet 680
getARGB16(short[], int, int, int, int, int, int)
of javax.microedition.lcdui.Image 460
getAscent()
of javax.microedition.lcdui.Font 377
getAuthMode()
of javax.microedition.rms.RecordStoreInfo 747
getAvailableFonts()
of javax.microedition.lcdui.Font 377
getAvailableFonts(int)
of javax.microedition.lcdui.Font 378
getAvailableFonts(int, int, int, int, int)
of javax.microedition.lcdui.Image 460
getBackgroundColor()
of javax.microedition.lcdui.Text 564
of javax.microedition.lcdui.TextEditor 586
getBaselinePosition()
of javax.microedition.rms.RecordStoreInfo 747
getBaudRate()
of javax.microedition.io.CommConnection 123
getBestImageHeight(int)
of javax.microedition.lcdui.Display 332
getBestImageWidth(int)
of javax.microedition.lcdui.Display 333
getBlendingMode()
of javax.microedition.lcdui.Graphics 439
getBlueComponent()
of javax.microedition.lcdui.Graphics 439
goBoolean()
of javax.microedition.event.EventData 769
goBorderStyle(boolean)
of javax.microedition.lcdui.Display 333
goCapabilities()
of javax.microedition.lcdui.Display 333
goCaret()
of javax.microedition.lcdui.Text 565
goCaretPosition()
of javax.microedition.lcdui.Text 565
goCharIndex(int, int)
of javax.microedition.lcdui.Text 565
goChars(char[])
of javax.microedition.lcdui.Text 565
goCipherSuite()
of javax.microedition.io.SecurityInfo 175
goClipHeight()
of javax.microedition.lcdui.Graphics 439
goClipWidth()
of javax.microedition.lcdui.Graphics 439
goClipX()
of javax.microedition.lcdui.Graphics 439
goClipY()
of javax.microedition.lcdui.Graphics 439
goColor()
of javax.microedition.lcdui.Graphics 440
goColor(int)
of javax.microedition.lcdui.Display 333
goColumns()
of javax.microedition.lcdui.game.TiledLayer 655
goColumns()
of javax.microedition.lcdui.TableLayoutPolicy 554
goCommand(int)
of javax.microedition.lcdui.Menu 514
goCommand(int)
of javax.microedition.lcdui.Displayable 346
goCommandLayoutPolicy()
of javax.microedition.lcdui.Displayable 346
goCommand()
of javax.microedition.lcdui.Displayable 346
goCommandPreferredPlacements(int)
of javax.microedition.lcdui.Display 334
goCommands()
of javax.microedition.lcdui.Displayable 346
goCommandItem()
of javax.microedition.lcdui.Item 483
goCommandType()
of javax.microedition.lcdui.Command 279
goConstraints()
of javax.microedition.lcdui.Text 577
goConstraints()
of javax.microedition.lcdui.TextEditor 586
of javax.microedition.lcdui.TextField 609
getContentPane() of javax.microedition.lcdui.TextEditor 586
getCount() of javax.microedition.lcdui.TabbedPane 548
getCurrent() of javax.microedition.lcdui.Display 334
of javax.microedition.lcdui.Form 395
getCurrent(String) of javax.microedition.lcdui.event.EventManager 784
getCurrentDisplay() of javax.microedition.lcdui.Displayable 346
getDate() of javax.microedition.io.HttpConnection 141
of javax.microedition.lcdui.DateField 306
getDefaultFont() of javax.microedition.lcdui.Font 379
getDefaultTimeout() of javax.microedition.lcdui.Alert 210
getDescent() of javax.microedition.lcdui.Font 379
getDisplay(MIDlet) of javax.microedition.lcdui.Display 334
getDisplayColor(int) of javax.microedition.lcdui.Graphics 440
getDisplays(int) of javax.microedition.lcdui.Display 335
getDisplayState() of javax.microedition.lcdui.Display 335
getch() of javax.microedition.lcdui.Command 279
getAddress(String) of javax.microedition.lcdui.TextEditor 586
of javax.microedition.lcdui.Text 566
of javax.microedition.lcdui.Menu 514
getFont(int) of javax.microedition.lcdui.List 502
of javax.microedition.lcdui.Text 566
of javax.microedition.lcdui.Font 379
of javax.microedition.lcdui.ChoiceGroup 265
of javax.microedition.lcdui.Choice 256
getFont(int, int, int) of javax.microedition.lcdui.Font 380
gsetFont(String, int, int) of javax.microedition.lcdui.Font 380
getFontName() of javax.microedition.lcdui.Font 381
getForegroundColor() of javax.microedition.lcdui.Text 566
of javax.microedition.lcdui.TextEditor 587
getForegroundColor(int) of javax.microedition.lcdui.Text 566
getFont() of javax.microedition.lcdui.FormLayoutPolicy 401
getFrame() of javax.microedition.lcdui.game.Sprite 645
getFrame(int) of javax.microedition.lcdui.AnimatedImage 221
getFrameCount() of javax.microedition.lcdui.AnimatedImage 222
getCodeDelay(int) of javax.microedition.lcdui.AnimatedImage 222
getFrameSequenceLength() of javax.microedition.lcdui.game.Sprite 645
gameAction(int) of javax.microedition.lcdui.Canvas 236
of javax.microedition.lcdui.CustomButton 293
getGraphics() of javax.microedition.lcdui.game.GameCanvas 624
of javax.microedition.lcdui.Image 462
greyScale()
of javax.microedition.lcdui.Graphics 440
getGreenComponent()
of javax.microedition.lcdui.Graphics 441
generateHardwareState()
of javax.microedition.lcdui.Display 336
generateHeaderField(int)
of javax.microedition.io.HttpConnection 142
generateHeaderField(String)
of javax.microedition.io.HttpConnection 142
generateHeaderFieldDate(String, long)
of javax.microedition.lcdui.Display 142
generateHeaderFieldInt(String, int)
of javax.microedition.io.HttpConnection 143
generateHeaderFieldKey(int)
of javax.microedition.io.HttpConnection 143
getHeight()
of javax.microedition.lcdui.game.Layer 627
of javax.microedition.lcdui.Canvas 236
of javax.microedition.lcdui.Display 336
of javax.microedition.lcdui.Image 462
of javax.microedition.lcdui.Form 395
of javax.microedition.lcdui.FileSelector 363
of javax.microedition.lcdui.TextBox 577
of javax.microedition.lcdui.Displayable 347
of javax.microedition.lcdui.CanvasItem 247
of javax.microedition.lcdui.TabbedPane 549
of javax.microedition.lcdui.List 503
of javax.microedition.lcdui.Text 566
of javax.microedition.lcdui.Alert 211
of javax.microedition.lcdui.Font 381
getHeight(Item)
of javax.microedition.lcdui.FormLayoutPolicy 401
getHighlightColor()
of javax.microedition.lcdui.TextEditor 587
getHighlightIndex()
of javax.microedition.lcdui.Text 567
getHighlightLength()
of javax.microedition.lcdui.Text 567
getHost()
of javax.microedition.io.HttpConnection 143
getIdleItem()
of javax.microedition.lcdui.Display 337
getImage()
of javax.microedition.lcdui.Alert 211
of javax.microedition.lcdui.Command 279
of javax.microedition.lcdui.Menu 514
of javax.microedition.lcdui.ImageItem 472
getImage(int)
of javax.microedition.lcdui.List 503
of javax.microedition.lcdui.ChoiceGroup 266
of javax.microedition.lcdui.LcduiChoice 256
getIncrementValue()
of javax.microedition.lcdui.Gauge 409
getIndent()
of javax.microedition.lcdui.Text 567
getIndicator()
of javax.microedition.lcdui.Alert 211
getInfo()
of javax.microedition.event.EventData 770
getInitialDirection()
of javax.microedition.lcdui.Text 567
getInputMode()
of javax.microedition.lcdui.DateField 306
getInstance()
of javax.microedition.event.EventManager 784
getInt()
of javax.microedition.event.EventData 770
getInteractionModes()
of javax.microedition.lcdui.CustomButton 293
getIssuer()
of javax.microedition.pki.Certificate 694
getKeyCode(int)
of javax.microedition.lcdui.Canvas 236
of javax.microedition.lcdui.CustomButton 294
getKeyLabel(int)
of javax.microedition.lcdui.Canvas 237
getKeyStates()
of javax.microedition.lcdui.game.GameCanvas 624
getLabel()
of javax.microedition.lcdui.Item 483
of javax.microedition.lcdui.Command 279
of javax.microedition.lcdui.Menu 515
getLastModified()
of javax.microedition.io.HttpConnection 144
of javax.microedition.rms.RecordStore 733
getLayerAt(int)
of javax.microedition.lcdui.game.LayerManager 631
getLayout()
of javax.microedition.lcdui.Item 483
of javax.microedition.lcdui.ImageItem 472
getLayoutDirection()
of javax.microedition.lcdui.FormLayoutPolicy 401
getLayoutHint()
of javax.microedition.lcdui.Item 484
getLayoutPolicy()
of javax.microedition.lcdui.Form 395
getLeading()
of javax.microedition.lcdui.Font 381
generateLineHeight()
of javax.microedition.lcdui.TextEditor 587
getLocalAddress()
of javax.microedition.io.SocketConnection 183
of javax.microedition.io.UDPDatagramConnection 186
of javax.microedition.io.ServerSocketConnection 179
getLocalPort()
of javax.microedition.io.SocketConnection 183
of javax.microedition.io.UDPDatagramConnection 186
of javax.microedition.io.ServerSocketConnection 179
getLong()
of javax.microedition.event.EventData 771
getLongLabel()
of javax.microedition.lcdui.Command 279
of javax.microedition.lcdui.Menu 515
getLoopCount()
of javax.microedition.lcdui.AnimatedImage 222
getMaxAscent()
of javax.microedition.lcdui.Font 381
getMaxDescent()
of javax.microedition.lcdui.Font 382
getMaxMenuDepth()
of javax.microedition.lcdui.Menu 515
getMaxSize()
of javax.microedition.lcdui.TextBox 577
of javax.microedition.lcdui.TextField 609
of javax.microedition.lcdui.TextEditor 587
getMaxValue()
of javax.microedition.lcdui.Gauge 410
getMenu(int)
of javax.microedition.lcdui.Displayable 347
of javax.microedition.lcdui.Menu 515
getMenuDepth()
of javax.microedition.lcdui.Menu 515
getMenuPreferredPlacements()
of javax.microedition.lcdui.Display 337
getMenuSupportedPlacements()
of javax.microedition.lcdui.Display 337
getMessage()
of javax.microedition.event.EventData 771
getMIDlet(String)
of javax.microedition.io.PushRegistry 166
getMIDletIdentity()
of javax.microedition.midlet.MIDlet 681
getMinContentHeight()
of javax.microedition.lcdui.CustomButton 294
getMinContentWidth()
of javax.microedition.lcdui.CustomButton 294
getMinimumHeight()
of javax.microedition.lcdui.Item 484
getMinimumWidth()
of javax.microedition.lcdui.Item 484
getMinValue()
of javax.microedition.lcdui.Gauge 410
getMode()
of javax.microedition.lcdui.FileSelector 363
getName()
of javax.microedition.event.EventData 771
of javax.microedition.io.IMCServerConnection 156
of javax.microedition.lcdui.Font 382
of javax.microedition.midlet.MIDletIdentity 686
of javax.microedition.rms.RecordStore 733
getNextRecordID()
of javax.microedition.rms.RecordStore 734
getNotAfter()
of javax.microedition.pki.Certificate 694
getNotBefore()
of javax.microedition.pki.Certificate 694
getNumMenuRecords()
of javax.microedition.rms.RecordStore 734
getOrientation()
of javax.microedition.lcdui.Display 337
getParent()
of javax.microedition.lcdui.CanvasItem 247
getPreferredSize()
of javax.microedition.lcdui.Font 382
getPreferredSize(String)
of javax.microedition.lcdui.Font 382
getPort()
of javax.microedition.io.HttpConnection 144
of javax.microedition.io.HttpsConnection 149
of javax.microedition.io.SocketConnection 183
getPositionX()
of javax.microedition.lcdui.CanvasItem 247
getPositionY()
of javax.microedition.lcdui.CanvasItem 248
getPrefContentHeight(int)
of javax.microedition.lcdui.CustomButton 294
getPrefContentWidth(int)
of javax.microedition.lcdui.CustomButton 295
getPreferredHeight()
of javax.microedition.lcdui.Item 484
getPreferredWidth()
of javax.microedition.lcdui.Item 484
getPriority()
of javax.microedition.lcdui.Command 280
getProtocol()
of javax.microedition.io.HttpConnection 144
getProtocolName()
of javax.microedition.io.SecurityInfo 175
getProtocolVersion()
of javax.microedition.io.SecurityInfo 176
getQuery()
of javax.microedition.io.HttpConnection 144
getRawFrameCount()
of javax.microedition.lcdui.game.Sprite 646
getReason()
of javax.microedition.pki.CertificateException 700
getRecord(int)
of javax.microedition.rms.RecordStore 734
getRecord(int, byte[], int)
of javax.microedition.rms.RecordStore 734
getRecordId(int)
of javax.microedition.rms.RecordEnumeration 719
getRecordSize(int)
of javax.microedition.rms.RecordStore 735
getRecordStoreInfo()
of javax.microedition.rms.RecordStore 735
getRedComponent()
of javax.microedition.lcdui.Graphics 441
getRef()
of javax.microedition.io.HttpConnection 144
getRefPixelX()
of javax.microedition.lcdui.game.Sprite 646
getRefPixelY()
of javax.microedition.lcdui.game.Sprite 646
getRemoteIdentity()
of javax.microedition.io.IMCConnection 153
getRequestedServerVersion()
of javax.microedition.io.IMCConnection 153
getRequestMethod()
of javax.microedition.io.HttpConnection 144
getRequestProperty(String)
of javax.microedition.io.HttpConnection 145
getRequiredHeight()
of javax.microedition.lcdui.Text 567
getRequiredLineCount()
of javax.microedition.lcdui.Text 568
getResponseCode()
of javax.microedition.io.HttpConnection 145
getSecurityDomain()
of javax.microedition.midlet.MIDletIdentity 686
getSecurityInfo()
of javax.microedition.io.HttpsConnection 149
of javax.microedition.io.SecureConnection 173
getSelectedFlags(boolean[])
of javax.microedition.lcdui.ChoiceGroup 266
of javax.microedition.lcdui.Choice 257
of javax.microedition.lcdui.List 503
getSelectedIndex()
of javax.microedition.lcdui.TabbedPane 549
of javax.microedition.lcdui.List 503
of javax.microedition.lcdui.Choice 257
of javax.microedition.lcdui.ChoiceGroup 266
generateSelection()
of javax.microedition.lcdui.TextEditor 587
getPreferredSize()
of javax.microedition.pki.Certificate 695
generateServerCertificate()
of javax.microedition.io.SecurityInfo 176
generateServerName()
of javax.microedition.io.IMCConnection 153
generateSignature()
of javax.microedition.pki.Certificate 695
generateSize()
of javax.microedition.lcdui.game.LayerManager 631
of javax.microedition.lcdui.Font 383
of javax.microedition.rms.RecordStore 735
of javax.microedition.rms.RecordStoreInfo 748
getSpaceAbove()
of javax.microedition.lcdui.Text 568
getSocketOption(byte)
of javax.microedition.io.SocketConnection 184
getSoftkeyLabelCoordinates(int)
of javax.microedition.lcdui.Canvas 237
getSourceInfo()
of javax.microedition.lcdui.TextBox 578
of javax.microedition.lcdui.TextField 609
of javax.microedition.lcdui.TextEditor 587
of javax.microedition.lcdui.Alert 211
of javax.microedition.lcdui.Ticker 614
getString(int)
of javax.microedition.lcdui.Choice 257
of javax.microedition.lcdui.ChoiceGroup 267
of javax.microedition.lcdui.List 504
getStrokeStyle()
of javax.microedition.lcdui.Graphics 441
getStyle()
of javax.microedition.lcdui.Font 383
getStyle(String)
of javax.microedition.lcdui.Font 383
getSubject()
of javax.microedition.pki.Certificate 695
getSystemEvents()
of javax.microedition.event.EventManager 784
getTabIcon(int)
of javax.microedition.lcdui.TabbedPane 549
getTag(int)
of javax.microedition.rms.RecordStore 736
getText()
of javax.microedition.lcdui.StringItem 543
textLength()
of javax.microedition.lcdui.Text 569
getTicker()
of javax.microedition.lcdui.Displayable 347
timeout()
of javax.microedition.Alert 211
timestamp()
of javax.microedition.event.EventData 772
title()
of javax.microedition.lcdui.Displayable 347
translateX()
of javax.microedition.lcdui.Graphics 441
translateY()
of javax.microedition.lcdui.Graphics 441
traverse(Item, int)
of javax.microedition.lcdui.TableLayoutPolicy 554
of javax.microedition.lcdui.FormLayoutPolicy 401
type()
of javax.microedition.lcdui.Notification 523
of javax.microedition.lcdui.Alert 211
of javax.microedition.pki.Certificate 695
getUrl()
of javax.microedition.io.HttpConnection 146
of javax.microedition.lcdui.FileSelector 363
gValue()
of javax.microedition.event.EventData 772
of javax.microedition.lcdui.Gauge 410
gVendor()
of javax.microedition.midlet.MIDletIdentity 687
gVersion()
of javax.microedition.io.IMCServerConnection 156
of javax.microedition.midlet.MIDletIdentity 687
of javax.microedition.pki.Certificate 695
of javax.microedition.rms.RecordStore 736
gViewportHeight()
of javax.microedition.lcdui.ScalableImage 534
gViewportWidth()
of javax.microedition.lcdui.ScalableImage 534
gVisible()
of javax.microedition.lcdui.TextEditor 587
gVisibleContentPosition()
of javax.microedition.lcdui.TextEditor 588
gWidth()
of javax.microedition.lcdui.game.Layer 627
of javax.microedition.lcdui.TextBox 578
of javax.microedition.lcdui.TabbedPane 550
of javax.microedition.lcdui.Displayable 348
of javax.microedition.lcdui.Form 396
of javax.microedition.lcdui.Image 466
of javax.microedition.lcdui.Alert 212
of javax.microedition.lcdui.FileSelector 364
of javax.microedition.lcdui.CanvasItem 248
of javax.microedition.lcdui.Text 569
of javax.microedition.lcdui.Display 337
of javax.microedition.lcdui.List 504
of javax.microedition.lcdui.Canvas 237
gWidth(Item)
of javax.microedition.lcdui.FormLayoutPolicy 402
gx()
of javax.microedition.lcdui.game.Layer 627
getX(Item)
of javax.microedition.lcdui.FormLayoutPolicy 402
gY()
of javax.microedition.lcdui.game.Layer 627
getY(Item)
of javax.microedition.lcdui.FormLayoutPolicy 402
gZPosition()
of javax.microedition.lcdui.CanvasItem 248
Glossary 3
Granting Application Level Access 73
Granting Permissions to MIDlet Suites 68
Graphics
of javax.microedition.lcdui 413
Graphics 413
Graphics and Text in Low-Level API 196
Graphics#16ARGB
of javax.microedition.lcdui
Graphics#16RGB
of javax.microedition.lcdui
Graphics#24RGB
of javax.microedition.lcdui
Graphics#32ARGB
of javax.microedition.lcdui
Graphics#clip
of javax.microedition.lcdui

H
handleEvent(EventData)
of javax.microedition.event.EventDataListener

Handling Localized Attribute Versions 32

Hardware 10
hardwareStateChanged(Display, int)
of javax.microedition.lcdui.DisplayListener

hasAlpha()
of javax.microedition.lcdui.Image

hashCode()
of javax.microedition.event.EventData

of javax.microedition.event.EventPermission

of javax.microedition.io.PushRegistryPermission

of javax.microedition.lcdui.Font

hasNextElement()
of javax.microedition.rms.RecordEnumeration

hasPointerEvents()
of javax.microedition.lcdui.Canvas

hasPointerMotionEvents()
of javax.microedition.lcdui.Display

hasPreviousElement()
of javax.microedition.rms.RecordEnumeration

hasRepeatEvents()
of javax.microedition.lcdui.Canvas

HCENTER
of javax.microedition.lcdui/Graphics

HEAD
of javax.microedition.io.HttpConnection

HELP
of javax.microedition.lcdui.Command

hideNotify()
of javax.microedition.lcdui.Canvas

of javax.microedition.lcdui.CustomItem

High-Level API for Events 194

How This Specification Is Organized 2

HTTP Network Connection 112

HTTP Networking 112

HTTP Request Header Example 114

HTTP_ACCEPTED
of javax.microedition.io.HttpConnection

HTTP_BAD_GATEWAY
of javax.microedition.io.HttpConnection

HTTP_BAD_METHOD
of javax.microedition.io.HttpConnection

HTTP_BAD_REQUEST
of javax.microedition.io.HttpConnection

HTTP_CLIENT_TIMEOUT
of javax.microedition.io.HttpConnection

HTTP_CONFLICT
of javax.microedition.io.HttpConnection

HTTP CREATED
of javax.microedition.io.HttpConnection

HTTP_ENTITY_TOO_LARGE
of javax.microedition.io.HttpConnection

HTTP_EXPECT_FAILED
of javax.microedition.io.HttpConnection

HTTP_FORBIDDEN
of javax.microedition.io.HttpConnection

HTTP_GATEWAY_TIMEOUT
of javax.microedition.io.HttpConnection

HTTP_GONE
of javax.microedition.io.HttpConnection

HTTP INTERNAL_ERROR
of javax.microedition.io.HttpConnection

HTTP_LENGTH_REQUIRED
of javax.microedition.io.HttpConnection

HTTP_MOVED_PERM
of javax.microedition.io.HttpConnection

HTTP_MOVED_TEMP
of javax.microedition.io.HttpConnection

HTTP_MULT_CHOICE
of javax.microedition.io.HttpConnection

HTTP_NO_CONTENT
of javax.microedition.io.HttpConnection

HTTP_NOT_ACCEPTABLE
of javax.microedition.io.HttpConnection

HTTP_NOT_AUTHORITATIVE
of javax.microedition.io.HttpConnection

HTTP_NOT_FOUND
of javax.microedition.io.HttpConnection

HTTP_NOT_IMPLEMENTED
of javax.microedition.io.HttpConnection

HTTP_NOT_MODIFIED
of javax.microedition.io.HttpConnection

HTTP_OK
of javax.microedition.io.HttpConnection

HTTP_PARTIAL
Image
   of javax.microedition.lcdui
Image#alpha
   of javax.microedition.lcdui
Image#GIF
   of javax.microedition.lcdui
Image#GIF89a
   of javax.microedition.lcdui
Image#ISOIECJPEG
   of javax.microedition.lcdui
Image#JFIF
   of javax.microedition.lcdui
Image#JPEG
   of javax.microedition.lcdui
Image#PNG
   of javax.microedition.lcdui
Image#SVG
   of javax.microedition.lcdui
Image#SVGTiny
   of javax.microedition.lcdui
ImageItem
   of javax.microedition.lcdui
ImageItem(String)
   of javax.microedition.lcdui
ImageItem(String, Image, int, String)
   of javax.microedition.lcdui
ImageItem(String, Image, int, String, int)
   of javax.microedition.lcdui
IMCConnection
   of javax.microedition.io
IMCConnection
   of javax.microedition.io
IMCServerConnection
   of javax.microedition.io
Implementation Notes
   201
IMPLICIT
   of javax.microedition.lcdui
implies(Permission)
   of javax.microedition.event
importRecordStore(InputStream, String, String)
   of javax.microedition.rms
INCREMENTAL_IDELE
   of javax.microedition.lcdui
INCREMENTAL_UPDATING
   of javax.microedition.lcdui
Inserting Certificates into the Application Descriptor

78

Installation 43

Inter-MIDlet Communications (IMC) 116

Interplay of High-Level Commands and the Low-Level API 196

Interplay with Application Manager 189

Introduction 84

invalidate()
of javax.microedition.lcdui.CustomButton 295

invalidateCommandLayout()
of javax.microedition.lcdui.Displayable 348

InvalidRecordIDException()
of javax.microedition.rms 715

InvalidRecordIDException(String)
of javax.microedition.rms 715

Invocation 48

io
of javax.microedition.midlet 112

IP Version Support 116

isAnimated()
of javax.microedition.lcdui.Image 466

isAuthorized()
of javax.microedition.lcdui.Display 687

isBold()
of javax.microedition.lcdui.CustomButton 295

isBuiltIn()
of javax.microedition.lcdui.Display 338

isColor()
of javax.microedition.lcdui.Display 338

isCommand(int)
of javax.microedition.lcdui.CustomButton 295

isDoubleBuffered()
of javax.microedition.lcdui.Canvas 238

isEnabled()
of javax.microedition.lcdui.CustomButton 295

isEnabled(int)
of javax.microedition.lcdui.List 504

of javax.microedition.lcdui.Choice 258

of javax.microedition.lcdui.ChoiceGroup 267

isEncrypted()
of javax.microedition.rms.RecordStoreInfo 748

isInteractive()
of javax.microedition.lcdui.CustomButton 295

isItalic()
of javax.microedition.lcdui.CustomButton 295

isKeptUpdated()
of javax.microedition.rms.RecordEnumeration 720

isMutable()
of javax.microedition.lcdui.CustomButton 295

isPlain()
of javax.microedition.lcdui.CustomButton 295

isScalable()
of javax.microedition.lcdui.CustomButton 295

isSelected(int)
of javax.microedition.lcdui.List 505

of javax.microedition.lcdui.ChoiceGroup 267

of javax.microedition.lcdui.Choice 258

isScreenSaver()
of javax.microedition.midlet.MIDlet 681

isShown()
of javax.microedition.lcdui.Displayable 348
isUnderlined()
of javax.microedition.lcdui.Font 384
isValid(Item)
of javax.microedition.lcdui.FormLayoutPolicy 403
isVisible()
of javax.microedition.lcdui.game.Layer 627
of javax.microedition.lcdui.Menu 517
isWriteable()
of javax.microedition.rms.RecordStoreInfo 748
Item 474
Item
of javax.microedition.lcdui.Command 276
of javax.microedition.lcdui 474
Item#appearance
of javax.microedition.lcdui 476
Item#sizes
of javax.microedition.lcdui 475
ItemCommandListener
of javax.microedition.lcdui 489
ItemCommandListener 489
ItemLayoutHint 490
ItemLayoutHint
of javax.microedition.lcdui 490
itemStateChanged(Item)
of javax.microedition.lcdui.ItemStateListener 491
ItemStateListener
of javax.microedition.lcdui 491
ItemStateListener 491
ItemTraversalListener
of javax.microedition.lcdui 492
ItemTraversalListener 492
ItemTraversedIn(Item)
of javax.microedition.lcdui.ItemTraversalListener 492
itemTraversedOut(Item)
of javax.microedition.lcdui.ItemTraversalListener 492
}

J
jad_attributes#jad_attributes 794
jad_attributes#Table A-1 : Application Descriptor Attributes 794
jad_attributes#tableA-1 794
JAR Signature 104
java.lang.package#app_resource_files 110
java.lang.package#StartOfIMIDP3Javadoc 107
java.lang.package#Table 10-1 : MIDP 3.0 System Properties 107
java.lang.package#table10-1 107
javax.microedition.io.package#Figure 11-1 : HTTP Network Connection Over Various Stacks 112
javax.microedition.io.package#figure11-1 112
javax.microedition.io.package#push 117
javax.microedition.io.package#Table 11-1 : System Properties Used for User-Agent and Accept-Language Request Headers 114
javax.microedition.io.package#table11-1 114
javax.microedition.io.package#UserAgentHeaders 114
javax.microedition.lcdui.package#concurrency 199
javax.microedition.lcdui.package#downloadablefonts 198
javax.microedition.lcdui.package#events 190
javax.microedition.midlet.package#appMgrDefined 668
javax.microedition.midlet.package#execution 661
javax.microedition.midlet.package#Figure 15-1: MIDlet Lifecycle State Diagram 669
javax.microedition.midlet.package#Figure 15-2: Idle Screen MIDlet Example 665
javax.microedition.midlet.package#figure15-1 669
javax.microedition.midlet.package#figure15-2 665
javax.microedition.midlet.package#IdleScreenMIDlet 665
javax.microedition.midlet.package#lifecycle 667
javax.microedition.midlet.package#persistent_midlet _suites 667
javax.microedition.midlet.package#splashscreens 672
javax.microedition.midlet.package#Table 15-1: MIDlet States 668
javax.microedition.midlet.package#Table 15-2: MIDlet Lifecycle Sequence 670
javax.microedition.midlet.package#table15-1 668
javax.microedition.midlet.package#table15-2 670
javax.microedition.midlet.package#version_numberi ng 667
javax.microedition.pki.package#profile 690
javax.microedition.rms.package#PersistentStorage 701
javax.microedition.rms.package#RMSAppLevelAccess 702
javax.microedition.rms.package#rmsencryptionparams 706
javax.microedition.rms.package#RMSEncryptionStandards 705
javax.microedition.rms.package#rmsfileheader 706
javax.microedition.rms.package#RMSInterchangeFormat 705
javax.microedition.rms.package#RMSLocalEncryption 703
javax.microedition.rms.package#rmsrecordsstore
javax.microedition.rms.package#rmsrecordsstoredata 707
javax.microedition.rms.package#Table 18-1 : RMS Interchange File Format 706
javax.microedition.rms.package#Table18-1 706
Jog Dial Interaction 193

K
KEEPALIVE
of javax.microedition.io.SocketConnection 182
keepUpdated(boolean)
of javax.microedition.rms.RecordEnumeration 720
KEY_BACKSPACE
of javax.microedition.lcdui.Canvas 232
KEY_DELETE
of javax.microedition.lcdui.Canvas 232
KEY_DOWN
of javax.microedition.lcdui.Canvas 232
KEY_ENTER
of javax.microedition.lcdui.Canvas 232
KEY_ESCAPE
of javax.microedition.lcdui.Canvas 232
KEY_LEFT
of javax.microedition.lcdui.Canvas 232
KEY_NUM0
of javax.microedition.lcdui.Canvas 232
KEY_NUM1
of javax.microedition.lcdui.Canvas 233
KEY_NUM2
of javax.microedition.lcdui.Canvas 233
KEY_NUM3
of javax.microedition.lcdui.Canvas 233
KEY_NUM4
of javax.microedition.lcdui.Canvas 233
KEY_NUM5
of javax.microedition.lcdui.Canvas 233
KEY_NUM6
of javax.microedition.lcdui.Canvas 233
KEY_NUM7
of javax.microedition.lcdui.Canvas 233
KEY_NUM8
of javax.microedition.lcdui.Canvas 234
KEY_NUM9
of javax.microedition.lcdui.Canvas 234
KEY_POUND
of javax.microedition.lcdui.Canvas 234
KEY_PRESS
of javax.microedition.lcdui.CustomButton 291
KEY_RELEASE
of javax.microedition.lcdui.CustomButton 291
KEY_REPEAT
of javax.microedition.lcdui.CustomButton 292
KEY_RIGHT
of javax.microedition.lcdui.Canvas 234
KEY_SELECT
of javax.microedition.lcdui.Canvas 234
KEY_SPACE
of javax.microedition.lcdui.Canvas 234
KEY_STAR
of javax.microedition.lcdui(Canvas 234
KEY_TAB
of javax.microedition.lcdui.Canvas 235
KEY_UP
of javax.microedition.lcdui.Canvas 235
KeyListener 494
KeyListener
of javax.microedition.lcdui 494
keyPressed(int)
of javax.microedition.lcdui.CustomButton 295
of javax.microedition.lcdui.Canvas 239
keyPressed(int, int)
of javax.microedition.lcdui.CustomButton 295
of javax.microedition.lcdui.Canvas 239
keyReleased(int)
of javax.microedition.lcdui.CustomButton 296
of javax.microedition.lcdui.Canvas 239
keyReleased(int, int)
of javax.microedition.lcdui.CustomButton 296
of javax.microedition.lcdui.Canvas 239
keyRepeated(int)
of javax.microedition.lcdui.CustomButton 296
of javax.microedition.lcdui.Canvas 239
keyRepeated(int, int)
of javax.microedition.lcdui.CustomButton 296
of javax.microedition.lcdui.Canvas 239

L
lang
of java 107
lastRenderedIndex()
of javax.microedition.lcdui.Text 569
Layer
of javax.microedition.lcdui.game 626
Layer 626
LayerManager
of javax.microedition.lcdui.game 629
LayerManager 629
LayerManager()
of javax.microedition.lcdui.game 631
LAYOUT_2
of javax.microedition.lcdui.Item 480
LAYOUT_BOTTOM
microedition.locale 109
microedition.platfrom 109
microedition.profiles 109
midlet
    of javax.microedition 660
        of javax.microedition.midlet 678
midlet 678
 MIDlet Classes 667
 MIDlet Download and Execution While Roaming and
 After Changing the Smart Card 101
 MIDlet Lifecycle 668
 MIDlet Lifecycle Definitions 668
 MIDlet Lifecycle Model 670
 MIDlet States 668
 MIDlet Suite Deletion 48
 MIDlet Suite Lifecycle 40
 MIDlet Suite Packaging 23
 MIDlet Suite Provisioning 40
 MIDlet Suite Security 660
 MIDlet Suite Trust Model 63
 MIDlet Suites and LiBlets 17
 MIDlet Suites Trust Model Using X.509 PKI 77
 MIDlet()
    of javax.microedition.midlet 679
 MIDletIdentity
    of javax.microedition.midlet 685
 MIDletIdentity 685
 MIDlets 660
 MIDletStateChangeException 688
 MIDletStateChangeException
    of javax.microedition.midlet 688
 MIDletStateChangeException()
    of javax.microedition.midlet 688
 MIDletStateChangeException(String)
    of javax.microedition.midlet 688
 MIDP 3.0 Example Domain Policies 797
 MIDP 3.0 MIDlet Concurrency 59
 MIDP 3.0 Packaging 17
 MIDP 3.0 Platform 57
 MIDP 3.0 Provisioning 39
 MIDP 3.0 Security Policy 84
 MIDP Configuration Profile for UAProf 52
 MIDP Execution Environment 661
 MIDP X.509 Certificate Profile 690
 MIDP X.509 Certificate Profile for Trusted MIDlet
 Suites 81
 MISSING_SIGNATURE
    of javax.microedition.pki.CertificateException 698
 MMS
    of javax.microedition.lcdui.NotificationType 530
Mobile Information Device Profile, v3.0 (JSR 271) 1
Mobile Media Third Party Domain Policy 99
 MODE_ACTIVE
    of javax.microedition.lcdui.Display 325
 MODE_NORMAL
    of javax.microedition.lcdui.Display 325
 MODIFIER_ALT
    of javax.microedition.lcdui.KeyListener 495
 MODIFIER_CHR
    of javax.microedition.lcdui.KeyListener 495
 MODIFIER_COMMAND
    of javax.microedition.lcdui.KeyListener 495
 MODIFIER_CTRL
    of javax.microedition.lcdui.KeyListener 495
 MODIFIER_MASK
    of javax.microedition.lcdui.KeyListener 495
 MODIFIER_SHIFT
    of javax.microedition.lcdui.KeyListener 495
 move(int, int)
    of javax.microedition.lcdui.game.Layer 628
 moveCaret(int)
    of javax.microedition.lcdui.Text 569
MULTIPLE
    of javax.microedition.lcdui.Choice 254
 N
 Named Permission to Class Permission Mapping 103
 Naming Record Stores 701
 Network Access Requirements 97
 NETWORK_3GPP_CSD
    of javax.microedition.event.EventData 763
 NETWORK_3GPP_PD
    of javax.microedition.event.EventData 763
 NETWORK_3GPP_PD_3G
    of javax.microedition.event.EventData 763
 NETWORK_3GPP_PD_EDGE
    of javax.microedition.event.EventData 763
 NETWORK_3GPP_PD_HSDPA
    of javax.microedition.event.EventData 763
 NETWORK_802DOT11
    of javax.microedition.event.EventData 763
 NETWORK_802DOT16
    of javax.microedition.event.EventData 764
 NETWORK_CDMA
    of javax.microedition.event.EventData 764
 New Features 8
 nextFrame()
    of javax.microedition.lcdui.game.Sprite 646
 nextRecord()
    of javax.microedition.rms.RecordEnumeration 720
<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packaging#MicroEdition-Profile</td>
<td>30</td>
</tr>
<tr>
<td>Packaging#MicroEdition-Config</td>
<td>30</td>
</tr>
<tr>
<td>Packaging#MIDlet-Access-Auth-Cert-N</td>
<td>30</td>
</tr>
<tr>
<td>packaging#MIDlet-Access-Auth-Type-N</td>
<td>30</td>
</tr>
<tr>
<td>packaging#MIDlet-Certificate-N-M</td>
<td>28</td>
</tr>
<tr>
<td>packaging#MIDlet-Data-Size</td>
<td>27</td>
</tr>
<tr>
<td>packaging#MIDlet-Delete-Confirm</td>
<td>28</td>
</tr>
<tr>
<td>packaging#MIDlet-Delete-Notify</td>
<td>28</td>
</tr>
<tr>
<td>packaging#MIDlet-Dependency-JAD-URL-N</td>
<td>34</td>
</tr>
<tr>
<td>packaging#MIDlet-Dependency-Jar-SHA1-N</td>
<td>34</td>
</tr>
<tr>
<td>packaging#MIDlet-Description</td>
<td>25</td>
</tr>
<tr>
<td>packaging#MIDlet-Description-locale</td>
<td>25</td>
</tr>
<tr>
<td>packaging#MIDlet-Event-Launch-N</td>
<td>30</td>
</tr>
<tr>
<td>packaging#MIDlet-Font</td>
<td>32</td>
</tr>
<tr>
<td>packaging#MIDlet-Icon</td>
<td>25</td>
</tr>
<tr>
<td>packaging#MIDlet-Icon-locale</td>
<td>25</td>
</tr>
<tr>
<td>packaging#MIDlet-Info-URL</td>
<td>27</td>
</tr>
<tr>
<td>packaging#MIDlet-Install-Notify</td>
<td>28</td>
</tr>
<tr>
<td>packaging#MIDlet-Jar-RSA-SHA1-N</td>
<td>28</td>
</tr>
<tr>
<td>packaging#MIDlet-Jar-Size</td>
<td>27</td>
</tr>
<tr>
<td>packaging#MIDlet-Jar-Jar-URL</td>
<td>27</td>
</tr>
<tr>
<td>packaging#MIDlet-Maximum-Canvas-Size</td>
<td>32</td>
</tr>
<tr>
<td>packaging#MIDlet-Minimum-Canvas-Size</td>
<td>32</td>
</tr>
<tr>
<td>packaging#MIDlet-N</td>
<td>25</td>
</tr>
<tr>
<td>packaging#MIDlet-N-Category</td>
<td>29</td>
</tr>
<tr>
<td>packaging#MIDlet-N-locale</td>
<td>26</td>
</tr>
<tr>
<td>packaging#MIDlet-N-Scalable-Icon</td>
<td>26</td>
</tr>
<tr>
<td>packaging#MIDlet-N-Scalable-Icon-locale</td>
<td>26</td>
</tr>
<tr>
<td>packaging#MIDlet-N-Type</td>
<td>31</td>
</tr>
<tr>
<td>packaging#MIDlet-N-UserDenied</td>
<td>30</td>
</tr>
<tr>
<td>packaging#MIDlet-Name</td>
<td>25</td>
</tr>
<tr>
<td>packaging#MIDlet-Name-locale</td>
<td>25</td>
</tr>
<tr>
<td>packaging#MIDlet-Operator-Allowed</td>
<td>31</td>
</tr>
<tr>
<td>packaging#MIDlet-Permission-N</td>
<td>30</td>
</tr>
<tr>
<td>packaging#MIDlet-Permission-Opt-N</td>
<td>30</td>
</tr>
<tr>
<td>packaging#MIDlet-Persistent-Data-URL-N</td>
<td>28</td>
</tr>
<tr>
<td>packaging#MIDlet-Profile-Request</td>
<td>29</td>
</tr>
<tr>
<td>packaging#MIDlet-Push-N</td>
<td>30</td>
</tr>
<tr>
<td>packaging#MIDlet-Required-IP-Version</td>
<td>31</td>
</tr>
<tr>
<td>packaging#MIDlet-Scalable-Icon</td>
<td>25</td>
</tr>
<tr>
<td>packaging#MIDlet-Scalable-Icon-locale</td>
<td>25</td>
</tr>
<tr>
<td>packaging#MIDlet-Splash-Screen-Image</td>
<td>32</td>
</tr>
<tr>
<td>packaging#MIDlet-Update-URL</td>
<td>27</td>
</tr>
<tr>
<td>packaging#MIDlet-UserDenied</td>
<td>31</td>
</tr>
<tr>
<td>packaging#MIDlet-Vendor</td>
<td>25</td>
</tr>
<tr>
<td>packaging#MIDlet-Version</td>
<td>25</td>
</tr>
<tr>
<td>packaging#midlet_suite_packaging</td>
<td>23</td>
</tr>
<tr>
<td>packaging#MIDP 3.0 Packaging</td>
<td>17</td>
</tr>
<tr>
<td>packaging#OperationalAttributes</td>
<td>30</td>
</tr>
<tr>
<td>packaging#packaging</td>
<td>18</td>
</tr>
<tr>
<td>packaging#ProvisioningAttributes</td>
<td>27</td>
</tr>
<tr>
<td>packaging#Table 2-3 : Dependency Names For Standard APIs</td>
<td>37</td>
</tr>
<tr>
<td>packaging#Table 2-4 : Informational Attributes for LIBlets</td>
<td>38</td>
</tr>
<tr>
<td>packaging#Table 2-a : MIDlet Identification Attributes</td>
<td>25</td>
</tr>
<tr>
<td>packaging#Table 2-b : Identification Attributes for LIBlets</td>
<td>26</td>
</tr>
<tr>
<td>packaging#Table 2-c : Provisioning Attributes for MIDlet Suites and MIDlets</td>
<td>27</td>
</tr>
<tr>
<td>packaging#Table 2-d : Provisioning Attributes for LIBlets</td>
<td>29</td>
</tr>
<tr>
<td>packaging#Table 2-e : Operational Attributes for MIDlet Suites and MIDlets</td>
<td>30</td>
</tr>
<tr>
<td>packaging#Table 2-f : Operational Attributes for LIBlets</td>
<td>32</td>
</tr>
</tbody>
</table>

**paint(Graphics)**
- of javax.microedition.lcdui.game.TiledLayer 655
- of javax.microedition.lcdui.game.GameCanvas 625
- of javax.microedition.lcdui.game.Layer 628
- of javax.microedition.lcdui.game.Sprite 646
- of javax.microedition.lcdui.Canvas 239

**PAUSEAPP()**
- of javax.microedition.midlet.MIDlet 682

**Permission Example Using Only Required Permissions**

**Permission Example Using Required and Optional Permissions**

**Permissions for MIDlet Suites**

**Persistent Data Interchange**

**Persistent MIDlet Suites**

**Persistent Storage**

**PHONENUMBER**
- of javax.microedition.lcdui.TextField 606

**pki**
- of javax.microedition 690

**PLAIN**
- of javax.microedition.lcdui.Item 483

**Platform Default Character Encoding**

**platform#MIDP 3.0 Platform**
- of javax.microedition 57

**platformRequest(String)**
- of javax.microedition.midlet.MIDlet 682
<table>
<thead>
<tr>
<th>Method/Field</th>
<th>Class/Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>playSound(Display)</td>
<td>javax.microedition.lcdui.AlertType</td>
</tr>
<tr>
<td>POINTER_DRAG</td>
<td>javax.microedition.lcdui.CustomButton</td>
</tr>
<tr>
<td>POINTER_PRESS</td>
<td>javax.microedition.lcdui.CustomButton</td>
</tr>
<tr>
<td>POINTER_RELEASE</td>
<td>javax.microedition.lcdui.CustomButton</td>
</tr>
<tr>
<td>pointerDragged(int, int)</td>
<td>javax.microedition.lcdui.CustomButton</td>
</tr>
<tr>
<td>pointerPressed(int, int)</td>
<td>javax.microedition.lcdui.CustomButton</td>
</tr>
<tr>
<td>pointerReleased(int, int)</td>
<td>javax.microedition.lcdui.CustomButton</td>
</tr>
<tr>
<td>POPUP</td>
<td>javax.microedition.lcdui.Canvas</td>
</tr>
<tr>
<td>POST</td>
<td>javax.microedition.lcdui.Canvas</td>
</tr>
<tr>
<td>profileDragged(int, int)</td>
<td>javax.microedition.lcdui.Canvas</td>
</tr>
<tr>
<td>profilePressed(int, int)</td>
<td>javax.microedition.lcdui.Canvas</td>
</tr>
<tr>
<td>profileReleased(int, int)</td>
<td>javax.microedition.lcdui.Canvas</td>
</tr>
<tr>
<td>PRECEDES</td>
<td>javax.microedition.rms.RecordComparator</td>
</tr>
<tr>
<td>Protection Domain</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>Protection Domains</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>Provisioned Record Stores</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>Provisioning Attributes</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>provisioning#app_integrity_failure</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>provisioning#AppDiscovery</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>provisioning#circular_dependencies</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>provisioning#dependencies_limit</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>provisioning#DevldReqHdr</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>provisioning#DevUAPROF</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>provisioning#display_size_failure</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>provisioning#Figure 3-1: LIBlet Provisioning Overview</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>provisioning#installation</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>provisioning#InstStatExample</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>provisioning#MIDletSuiteDelete</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>provisioning#MIDletSuiteUpdate</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>provisioning#MIDP 3.0 Provisioning</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>provisioning#RMSDataProvisioning</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>provisioning#StatusCodes</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>provisioning#StatusReports</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>Push Applications</td>
<td>javax.microedition.event.EventData</td>
</tr>
<tr>
<td>PushRegistry</td>
<td>javax.microedition.io</td>
</tr>
<tr>
<td>PushRegistry</td>
<td>javax.microedition.io</td>
</tr>
<tr>
<td>PushRegistry#PushAttr</td>
<td>javax.microedition.io</td>
</tr>
</tbody>
</table>
PushRegistryPermission 169
PushRegistryPermission
of javax.microedition.io 169
PushRegistryPermission(String, String)
of javax.microedition.io 170
PUT
of javax.microedition.io.HttpConnection 141

R

RCVBUF
of javax.microedition.io.SocketConnection 182
rebuild()
of javax.microedition.rms.RecordEnumeration 722
Recommendation for Permissions for Optional Packages 104
Record Store 701
Record Tags 704
recordAdded(RecordStore, int)
of javax.microedition.rms.RecordListener 724
recordChanged(RecordStore, int)
of javax.microedition.rms.RecordListener 724
RecordComparator
of javax.microedition.rms 716
RecordComparator 716
recordDeleted(RecordStore, int)
of javax.microedition.rms.RecordListener 724
RecordEnumeration 718
RecordEnumeration
of javax.microedition.rms 718
RecordFilter 723
RecordFilter
of javax.microedition.rms 723
RecordListener
of javax.microedition.rms 724
RecordListener 724
Records 705
RecordStore 726
RecordStore
of javax.microedition.rms 726
RecordStoreException
of javax.microedition.rms 744
RecordStoreException 744
RecordStoreException()
of javax.microedition.rms 744
RecordStoreException(String)
of javax.microedition.rms 744
RecordStoreFullException
of javax.microedition.rms 746
RecordStoreFullException()
of javax.microedition.rms 746
RecordStoreFullException(String)
of javax.microedition.rms 746
RecordStoreInfo 747
RecordStoreInfo
of javax.microedition.rms 747
RecordStoreNotFoundException
of javax.microedition.rms 750
RecordStoreNotFoundException()
of javax.microedition.rms 750
RecordStoreNotFoundException(String)
of javax.microedition.rms 750
RecordStoreNotOpenException
of javax.microedition.rms 751
RecordStoreNotOpenException()
of javax.microedition.rms 751
RecordStoreNotOpenException(String)
of javax.microedition.rms 751
RecordStoreNotOpenException()
of javax.microedition.rms 751
RecordStoreNotFoundException
of javax.microedition.rms 750
RecordStoreNotFoundException()
of javax.microedition.rms 750
RecordStoreNotFoundException(String)
of javax.microedition.rms 750
RecordStoreNotOpenException
of javax.microedition.rms 751
RecordStoreNotOpenException()
of javax.microedition.rms 751
RecordStoreNotOpenException(String)
of javax.microedition.rms 751
RecordStoreNotOpenException()
of javax.microedition.rms 751
References 12
registerAlarm(String, long)
of javax.microedition.io.PushRegistry 166
registerApplication(String, String, boolean)
of javax.microedition.event.EventManager 785
registerApplication(String, String, boolean, boolean)
of javax.microedition.event.EventManager 786
registerApplication(String, String, boolean, double, double)
of javax.microedition.event.EventManager 787
registerApplication(String, String, boolean, long, long)
of javax.microedition.event.EventManager 788
registerApplication(String, String, boolean, String[])
of javax.microedition.event.EventManager 788
registerConnection(String, String, String)
of javax.microedition.io.PushRegistry 167
REMINDER
of javax.microedition.lcdui.NotificationType 530
remove()
of javax.microedition.lcdui.Item 524
remove(Command)
of javax.microedition.lcdui.Drawable 517
remove(Layer)
of javax.microedition.lcdui.game.LayerManager 632
remove(Menu)
of javax.microedition.lcdui.Item 517
removeCommand(Command)
of javax.microedition.lcdui.Item 485
of javax.microedition.lcdui.Alert 212
of javax.microedition.lcdui.Drawable 348
of javax.microedition.lcdui.List 505
of javax.microedition.lcdui.FileSelector 364
removeCommandOrMenu(int)
of javax.microedition.lcdui.FileSelector 364
of javax.microedition.lcdui.Drawable 348
removeCurrent()
of javax.microedition.lcdui.Drawable 348
removedFromDisplay(Display)
of javax.microedition.lcdui.Display 339
removeDisplayListener(DisplayListener)
of javax.microedition.lcdui.Display 339
removeEventListener(String, EventDataListener)
of javax.microedition.event.EventManager 789
removeRecordListener(RecordListener)
of javax.microedition.rms.RecordStore 741
removeTab(int)
of javax.microedition.lcdui.TabbedPane 550
repaint()
of javax.microedition.lcdui.CustomItem 297
of javax.microedition.lcdui.Canvas 241
repaint(int, int, int, int)
of javax.microedition.lcdui.Canvas 241
of javax.microedition.lcdui.CustomItem 298
S
SAVE
of javax.microedition.lcdui.FileSelector 362
ScalableImage
of javax.microedition.lcdui.Drawable 532
ScalableImage 532
Scenario 1a 82
Scenario 1b 82
Scenario 2 82
Scenario 3 83
Scenarios of MIDlet Suite Signing 82
Scheduling 61
Scope 8
SCREEN
of javax.microedition.lcdui.Command 277
of javax.microedition.lcdui.Drawable 537
SCREEN 537
Screen Saver Deactivation 664
Screen Saver Identification 663
Screen Saver MIDlet Execution 664
Screen Saver MIDlets 663
Screen Saver Provisioning 663
Screen Saver Security 665
Screen Size Requirements 663
SCREENSAVER_MODE
of javax.microedition.event.EventData 766
SCREENSAVER_MODE_ACTIVATED
of javax.microedition.event.EventData 766
SCREENSAVER_MODE_DEACTIVATED
of javax.microedition.event.EventData 766
Secure Networking 115
Secure Record Stores 703
SecureConnection
of javax.microedition.io 172
SecureConnection 172
SecureRecordStoreException 752
SecureRecordStoreException
of javax.microedition.rms 752
SecureRecordStoreException()
of javax.microedition.rms 752
SecureRecordStoreException(String)
of javax.microedition.rms 752
Security Compatibility for MIDP 1.0 and MIDP 2.x 102
Security for MIDP 1.0 MIDlet Suites 102
Security for MIDP 2.x MIDlet Suites 102
Security for MIDP Applications 63
Security of Alternative Application Representation Formats 81
Security of Networking Functions 117
Security of PushRegistry 117
security_framework#AppLevelAccessAuth 71
security_framework#CombinedSecurityModels 75
security_framework#domain 64
security_framework#Figure 6-1 : Verification Steps for Granting & Denying Application Level Access 74
security_framework#granting_permissions 68
security_framework#MIDlet-Access-Auth-Cert-N 73
security_framework#MIDlet-Access-Auth-Type-N 72
security_framework#MIDletSuites Trust Model 63
security_framework#permissions 65
security_framework#policy_format 70
security_framework#request-perm 65
security_framework#Table 6-1 : Definition Of Security Terms 63
security_framework#Table 6-2 : Determining Application Credentials 72
security_framework#Table 6-3 : Application Attributes Required for Application Level Access Authorization 72
security_framework#TrustedMIDletSuite 64
security_framework#UntrustedMIDletSuite 64
security_framework#user_interaction_modes 67
security_pkitrust#AppAccessRootCert 77
security_pkitrust#authen 79
security_pkitrust#ProtDomainRootCert 77
security_pkitrust#RootCert 77
security_pkitrust#Table 7-1 : Definition Of PKI Security Terms 77
security_pkitrust#Table 7-2 : Actions Upon Completion of Signing Certificate Verification 79
security_pkitrust#Table 7-3 : Summary of MIDlet Suite Source Verification 80
security_pkitrust#table7-1 77
security_pkitrust#table7-2 79
security_pkitrust#table7-3 80
security_pkitrust#verifying_signing_cert 79
security_pkitrust#x509pki 77
security_policy#_ftn1 101
security_policy#change_def_permission_setting 99
security_policy#Figure 8-1 : Assigning Root Certificates to Operator & Identified Third Party Protection Domains 88
security_policy#Figure 8-2 : Recommended Security Prompt Icons 90
security_policy#Figure8-1 88
security_policy#Figure8-2 82
security_policy#smartcardchange 100
security_policy#Table 8-1 : Packages Allowed Without Confirmation to Unidentified Third Party Protection Domain 91
security_policy#Table 8-2 : Packages Allowed With Confirmation to Unidentified Third Party Protection Domain 91
security_policy#Table 8-3 : Function Groups & User Settings for Third Party Protection Domains 93
security_policy#Table 8-4 : Mapping Permissions to Function Groups 95
security_policy#Table 8-5 : Permissions Not Mapped To Function Groups 96
security_policy#Table8-1 91
security_policy#Table8-2 91
security_policy#Table8-3 93
security_policy#Table8-4 95
security_policy#Table8-5 96
security_policy#unidentified 90
security_policy#UserPrompts 100
SecurityInfo 175
SecurityInfo
of javax.microedition.io 175
SELECT_COMMAND
of javax.microedition.lcdui.List 500
SENSITIVE
of javax.microedition.lcdui.TextField 607
Serial Port Communications 116
ServerSocketConnection 177
ServerSocketConnection
of javax.microedition.io 177
serviceRepaints()
of javax.microedition.lcdui.Canvas 242
set(int, Item)
of javax.microedition.lcdui.Form 396
set(int, String, Image)
of javax.microedition.lcdui.List 506
of javax.microedition.lcdui.Choice 258
of javax.microedition.lcdui.ChoiceGroup 268
setActivityMode(int)
of javax.microedition.lcdui.Display 339
setAlignment(int)
of javax.microedition.lcdui.Text 570
setAlpha(int)
of javax.microedition.lcdui.Graphics 441
setAlphaColor(int)
of javax.microedition.lcdui.Graphics 442
setAlphaColor(int, int, int, int)
of javax.microedition.lcdui.Graphics 442
setAltText(String)
of javax.microedition.lcdui.ImageItem 472
setAnimatedTile(int, int)
of javax.microedition.lcdui.game.TiledLayer 656
setBackgroundColor(int)
of javax.microedition.lcdui.Text 570
setBackgroundColor(int, int, int, int)
of javax.microedition.lcdui.TextEditor 588
setBaudRate(int)
of javax.microedition.io.CommConnection 123
setBlendingMode(int)
of javax.microedition.lcdui.Graphics 442
setCaret(int)
of javax.microedition.lcdui.TextBox 579
setCell(int, int, int)
of javax.microedition.lcdui.game.TiledLayer 656
setChars(char[], int, int)
of javax.microedition.lcdui.TextField 610
setClip(int, int, int, int)
of javax.microedition.lcdui.Graphics 443
setCollisionAlpha(int)
of javax.microedition.lcdui.game.Sprite 647
setColor(int)
of javax.microedition.lcdui.Graphics 443
setColor(int, int, int)
of javax.microedition.lcdui.Displayable 349
setColor(int, int, int, int)
of javax.microedition.lcdui.TextEditor 591
setCommand(Command, int)
of javax.microedition.lcdui.Spacer 539
setCommandLayoutPolicy(CommandLayoutPolicy)
of javax.microedition.lcdui.Display 339
setCommandListener(CommandListener)
of javax.microedition.lcdui.Alert 212
setConstraints(int)
of javax.microedition.lcdui.TextEditor 589
setDate(Date)
of javax.microedition.lcdui.DateField 307
setDefaultCommand(Command)
of javax.microedition.lcdui.Spacer 539
setFilterExtensions(String[])
of javax.microedition.lcdui.FileSelector 365
setFitPolicy(int)
of javax.microedition.lcdui.ChoiceGroup 268
setFont(Font)
of javax.microedition.lcdui.Text 570
setFont(Font, int, int)
of javax.microedition.lcdui.TextEditor 590
setForegroundColor(int)
of javax.microedition.lcdui.Text 571
setForegroundColor(int, int, int)
of javax.microedition.lcdui.TextEditor 591
setForegroundColor(int, int, int, int)
of javax.microedition.lcdui.TextEditor 591
setFrame(int)
of javax.microedition.lcdui.game.Sprite 647
setFrameSequence(int[])
of javax.microedition.lcdui.game.Sprite 647
setFullScreenMode(boolean)
of javax.microedition.lcdui.Canvas 242
setCurrent(Alert, Displayable)
of javax.microedition.lcdui.Display 340
setCurrent(Item)
of javax.microedition.lcdui.Display 342
setEnabled(Command)
of javax.microedition.lcdui.Command 280
setFocus(boolean)
of javax.microedition.lcdui.GameCanvas 242
setFocus(int)
of javax.microedition.lcdui.TabbedPane 550
setFont(Font)
of javax.microedition.lcdui.Displayable 349
setFont(Font, int, int)
of javax.microedition.lcdui.Text 570
setGrayScale(int)
of javax.microedition.lcdui.Graphics 444
setHeight(int)
of javax.microedition.lcdui.Text 571
setHeight(int, boolean)
of javax.microedition.lcdui.ScalableImage 535
setHighlight(int, int)
of javax.microedition.lcdui.TextBox 580
setHighlightColor(int, int, int, int)
of javax.microedition.lcdui.TextEditor 591
setIdleItem(IdleItem)
of javax.microedition.lcdui.Display 342
setImage(Image)
of javax.microedition.lcdui.Menu 518
of javax.microedition.lcdui.Command 280
of javax.microedition.lcdui.ImageItem 473
of javax.microedition.lcdui.Notification 524
of javax.microedition.lcdui.Alert 212
setLayout(int)
of javax.microedition.lcdui.Item 487
of javax.microedition.lcdui.ImageItem 473
of javax.microedition.lcdui.ItemLayoutHint 488
of javax.microedition.lcdui.FormLayoutPolicy 497
of javax.microedition.lcdui.FormLayoutPolicy 497
of javax.microedition.lcdui.Displayable 349
setPosition(int, int)
of javax.microedition.lcdui.game.Layer 628
setPositionX(int)
of javax.microedition.lcdui.CanvasItem 248
setPositionY(int)
of javax.microedition.lcdui.CanvasItem 248
setPreferredOrientation(int)
of javax.microedition.lcdui.Display 343
setMaxValue(int)
of javax.microedition.lcdui.Spacer 539
setMaxSize(int)
of javax.microedition.lcdui.Spacer 539
setMenu(Menu, int)
of javax.microedition.lcdui.Command 281
SUPPORTS_ORIENTATION_LANDSCAPE
of javax.microedition.lcdui.Display 329
SUPPORTS_ORIENTATION_LANDSCAPE180
of javax.microedition.lcdui.Display 329
SUPPORTS_ORIENTATION_PORTRAIT
of javax.microedition.lcdui.Display 329
SUPPORTS_ORIENTATION_PORTRAIT180
of javax.microedition.lcdui.Display 329
SUPPORTS_TABBPANES
of javax.microedition.lcdui.Display 329
SUPPORTS_TEXTBOXES
of javax.microedition.lcdui.Display 329
SUPPORTS_TICKER
of javax.microedition.lcdui.Display 330
SUPPORTS_TITLE
of javax.microedition.lcdui.Display 330
System Functions 107
System Properties 107
System Properties Used for User-Agent and Accept-Language Request Headers 114
System.exit 107
SYSTEM_STATE
of javax.microedition.event.EventData 767
SYSTEM_STATE_NORMAL
of javax.microedition.event.EventData 767
SYSTEM_STATE_SHUTDOWN
of javax.microedition.event.EventData 767
SYSTEM_STATE_STANDBY
of javax.microedition.event.EventData 767
SYSTEM_STATE_STARTUP
of javax.microedition.event.EventData 767

T

TAB
of javax.microedition.lcdui.Display 330
tabAddedEvent(int, Screen)
of javax.microedition.lcdui.TabListener 556
TabbedPane
of javax.microedition.lcdui 545
TabbedPane(String, boolean, boolean)
of javax.microedition.lcdui 548
tabChangeEvent(Screen)
of javax.microedition.lcdui.TabListener 556
TableLayoutPolicy
of javax.microedition.lcdui 552
TableLayoutPolicy 552
TableLayoutPolicy(Form, int)
of javax.microedition.lcdui 553
TabListener 556

TabListener
of javax.microedition.lcdui.TabListener 556
tabRemovedEvent(int)
of javax.microedition.lcdui.TabListener 556

Text 558

Text
of javax.microedition.lcdui 558

Text Truncation in UI Components 200

Text()
of javax.microedition.lcdui 563

Text(String, int, int)
of javax.microedition.lcdui 563

Text(String, int, int, int)
of javax.microedition.lcdui 564

TEXT_WRAP_DEFAULT
of javax.microedition.lcdui.Choice 254

TEXT_WRAP_OFF
of javax.microedition.lcdui.Choice 255

TEXT_WRAP_ON
of javax.microedition.lcdui.Choice 255

TextBox
of javax.microedition.lcdui 574
TextBox 574

TextBox(String, String, int, int)
of javax.microedition.lcdui 576

TextEditor 582

TextEditor
of javax.microedition.lcdui 582

TextEditor(String, int, int, int, int)
of javax.microedition.lcdui 585

TextEditorChangeListener 594

TextEditorChangeListener
of javax.microedition.lcdui 594

TextField 597

TextField
of javax.microedition.lcdui 597

TextField#constraints
of javax.microedition.lcdui 597

TextField#modes
of javax.microedition.lcdui 599

TextField(String, String, int, int)
of javax.microedition.lcdui 608
textFits()
of javax.microedition.lcdui.Text 573

The Application Descriptor 20

The Dependency Hash Attribute 35

The JAR Manifest 18

The LIBlet JAD Download URL Attribute 34

The Redrawing Scheme 196

Third Party Protection Domains 89
Thread Model 199
Ticker
  of javax.microedition.lcdui 613
Ticker 613
Ticker(String)
  of javax.microedition.lcdui 614
TiledLayer 650
TiledLayer
  of javax.microedition.lcdui.game 650
TiledLayer(int, int, Image, int, int)
  of javax.microedition.lcdui.game 653
TIME
  of javax.microedition.lcdui.DateField 305
TOP
  of javax.microedition.lcdui.Graphics 425
toString()
  of javax.microedition.event.EventData 773
  of javax.microedition.midlet.MIDletIdentity 687
TRANS_MIRROR
  of javax.microedition.lcdui.game.Sprite 641
TRANS_MIRROR_ROT180
  of javax.microedition.lcdui.game.Sprite 641
TRANS_MIRROR_ROT270
  of javax.microedition.lcdui.game.Sprite 641
TRANS_MIRROR_ROT90
  of javax.microedition.lcdui.game.Sprite 641
TRANS_NONE
  of javax.microedition.lcdui.game.Sprite 641
TRANS_ROT180
  of javax.microedition.lcdui.game.Sprite 642
TRANS_ROT270
  of javax.microedition.lcdui.game.Sprite 642
TRANS_ROT90
  of javax.microedition.lcdui.game.Sprite 642
translate(int, int)
  of javax.microedition.lcdui.Graphics 444
Transport & Security Standards 77
traverse(int, int, int[])
  of javax.microedition.lcduiListItemIcon 300
TRAVEL_horizontal
  of javax.microedition.lcduiListItemIcon 293
TRAVEL_vertical
  of javax.microedition.lcduiListItemIcon 293
traverseOut()
  of javax.microedition.lcduiListItemIcon 303

U
UAPL for Device Identification 51
UDPDatagramConnection 185
UDPDatagramConnection
  of javax.microedition.io 185
UNAUTHORIZED_INTERMEDIATE_CA
  of javax.microedition.pki.CertificateException 699
unbind()
  of javax.microedition.lcdui.ScaleImage 535
UNEDITABLE
  of javax.microedition.lcdui.TextField 607
Unidentified Third Party Protection Domain 90
UNIDENTIFIED_THIRD_PARTY
  of javax.microedition.midlet.MIDletIdentity 686
UNRECOGNIZED_ISSUER
  of javax.microedition.pki.CertificateException 699
unregisterApplication(String, String)
  of javax.microedition.event.EventManager 789
unregisterConnection(String)
  of javax.microedition.io.PushRegistry 168
UNSUPPORTED_PUBLIC_KEY_TYPE
  of javax.microedition.pki.CertificateException 699
UNSUPPORTED_SIGALG
  of javax.microedition.pki.CertificateException 699
UP
  of javax.microedition.lcdui.Canvas 235
UP_PRESSED
  of javax.microedition.lcdui.game.GameCanvas 622
Update 47
URL
  of javax.microedition.lcdui.TextField 608
User Interface 187
User Permission Interaction Modes 67
User Prompting 98
User Prompts and Notifications 100
User-Agent and Accept-Language Request Headers 114
User-Agent Product Tokens 55

V
Vcenter
  of javax.microedition.lcdui.Graphics 425
VERIFICATION_FAILED
  of javax.microedition.pki.CertificateException 699
Verifying Signing Certificate 79
Verifying the Midlet Suite JAR 80
Version Numbering 667
vibrate(int)
  of javax.microedition.lcdui.Display 343
VOICE_CALL
  of javax.microedition.event.EventData 767

W
WARNING