

iDEN Multi-Communication Device
J2ME™ Developer Guide
2005

Version 1.98







Table of Contents

TABLE OF CONTENTS	3
1 INTRODUCTION	12
1.1 PURPOSE	12
1.2 AUDIENCE	12
1.3 DISCLAIMER	12
1.4 REFERENCES	13
1.5 REVISION HISTORY	13
1.6 ABBREVIATIONS AND ACRONYMS	14
1.7 ICON GUIDE	16
1.8 DOCUMENT OVERVIEW	18
2 J2ME™ INTRODUCTION	19
2.1 OVERVIEW	19
2.2 THE JAVA 2 PLATFORM, MICRO EDITION (J2ME™)	19
2.2.1 THE MOTOROLA J2ME™ PLATFORM	20
2.2.2 RESOURCES AND APIs AVAILABLE	20
2.2.3 PLATFORM SPECIFIC FEATURES	20
2.3 APPLICATION MANAGEMENT	23
2.3.1 MIDLET LIFECYCLE	23
2.3.2 MIDLET SUITE INSTALLATION	23
2.3.3 MIDLET SUITE DE-INSTALLATION	24
2.3.4 MIDLET SUITE UPDATING	24
2.3.5 STARTING, PAUSING, AND EXITING	24
2.3.6 JAVA SYSTEM	28
2.3.7 JAVA FROM MAIN MENU	28
2.3.8 PERSONALIZING THE NATIVE UI	29
2.3.9 LOW MEMORY INDICATION	29
2.3.10 THE MINIJIT	30
3 DEVELOPING AND PACKAGING J2ME™ APPLICATIONS	32
3.1 OVERVIEW	32
3.2 DEVELOPING FOR J2ME™	32
3.2.1 DEVELOPING – TOOLS AND EMULATION ENVIRONMENTS	32
3.2.2 PACKAGING – PUTTING THE PIECES TOGETHER	33
3.2.3 DESKTOP TO DEVICE	35
3.2.4 DEBUGGING – TERMINAL INTERFACE	36
3.2.5 BEYOND STANDARDS	40
3.2.6 RESOURCEBUNDLE	42
3.2.6.7 CODE EXAMPLES	47
3.2.7 LICENSEINFO API	55
3.3 MIDLET SUITE AND MIDLET ICON SUPPORT	61



3.3.1 TIPS	63
3.4 CLDC 1.1	64
3.4.1 OVERVIEW	64
4 MULTIMEDIA AND GRAPHICS	65
4.1 OVERVIEW	65
4.2 MIDP 2.0 LCDUI	66
4.2.1 OVERVIEW	66
4.2.2 COMMANDS	66
4.2.3 CANVAS	67
4.2.4 LIST	68
4.2.5 FORMS	68
4.2.6 ITEM COMMANDS	69
4.2.7 TEXTBOX/TEXTFIELD	69
4.3 EXTERNAL DISPLAY	72
4.3.1 OVERVIEW	72
4.3.2 CLASS DESCRIPTION	72
4.3.3 METHOD DESCRIPTIONS	73
4.3.4 CODE EXAMPLES	75
4.3.5 TIPS /	76
4.4 KEYCODE REMAPPING	77
4.5 LOOK AND FEEL (LNF)	80
4.5.1 OVERVIEW	80
4.5.2 CLASS DESCRIPTION	81
4.5.3 CODE EXAMPLES	81
4.6 SMART TEXT ENTRY	86
4.6.1 OVERVIEW	86
4.6.2 T9 FEATURES	86
4.6.3 THE T9 UI	87
4.6.4 CHANGING T9 ENTRY MODE	87
4.6.5 INFLUENCING T9	88
4.6.6 T9 ENGINE LIFECYCLE	88
4.7 LIGHTWEIGHT WINDOW TOOLKIT (LWT)	89
4.7.1 OVERVIEW	89
4.7.2 EXAMPLE: HELLO LWT WORLD	89
4.7.3 CLASS HIERARCHY AND OVERVIEW	90
4.7.4 COMPONENTSCREEN	91
4.7.5 COMPONENT	91
4.7.6 COMPONENTLISTENER	91
4.7.7 INTERACTABLECOMPONENT	91
4.7.8 THE COMPONENTSCREEN CLASS	92
4.7.9 THE COMPONENT CLASS	97
4.7.10 COMPONENT VISIBILITY, STATE & FOCUS	113
4.7.11 THE COMPONENTLISTENER INTERFACE	114
4.7.12 THE INTERACTABLECOMPONENT CLASS	114
4.7.13 THE BUTTON CLASS	116
4.7.14 THE IMAGELABEL CLASS	118
4.7.15 CHECKBOXES	121
4.7.16 THE TEXTCOMPONENT CLASS	122
4.7.17 THE TEXTFIELD CLASS	124
4.7.18 THE TEXTAREA CLASS	124
4.7.19 THE SLIDER CLASS	124



4.8 GRAPHICS ACCELERATION	126
4.8.1 OVERVIEW	126
4.8.2 iDEN-GRAPHICS-ACCELERATION: ON OFF AUTO.....	126
4.8.3 HOW IT WORKS	127
4.9 MICRO3D API	139
4.9.1 OVERVIEW	139
4.9.2 THE MICRO3D PACKAGE.....	140
4.9.3 WORKING WITH GRAPHICS AND ANIMATION.....	141
4.9.4 CREATING A FIGURE	142
4.9.5 LOADING AND USING TEXTURES	144
4.9.6 WORKING WITH VECTOR3D	147
4.9.7 CREATING PRIMITIVES.....	149
4.9.8 LOADING AND USING ACTION TABLES	155
4.9.9 SETTING THE SCENE: LIGHT.....	158
4.9.10 USING AFFINE TRANSFORMATIONS	159
4.9.11 SETTING THE SCENE: LAYOUT3D.....	162
4.9.12 AUTOMATIC VIEW TRANSFORMATION.....	164
4.9.13 MANUAL VIEW TRANSFORMATION.....	166
4.9.14 RENDERING.....	169
4.9.15 UTILITY	173
4.9.16 MEMORY	173
4.9.17 TIPS /.....	174
4.9.18 CAVEATS	174
4.9.19 COMPILING & TESTING MICRO3D MIDLETS	174
4.10 MOBILE 3D GRAPHICS API.....	175
4.10.1 IMMEDIATE MODE AND RETAINED MODE RENDERING	175
4.10.2 STEPS FOR CREATING A 3D APPLICATION USING THE MOBILE 3D GRAPHICS API.....	176
4.10.3 CODE EXAMPLES	176
4.10.4 THE CLASSIFICATION IN JSR 184.....	185
4.10.5 TIPS /.....	187
4.11 MULTIMEDIA	188
4.11.1 OVERVIEW	188
4.11.2 CLASS DESCRIPTION	191
4.11.3 METHOD DESCRIPTIONS	192
4.11.4 VIDEO PLAYBACK.....	199
4.11.5 TIPS AND CODE EXAMPLES /	202
4.11.6 COMPILING & TESTING MMA MIDLETS	204
4.11.7 TIPS /.....	204
4.12 REAL-TIME PROTOCOL	206
4.12.1 OVERVIEW.....	206
4.12.2 CLASS DESCRIPTION.....	206
4.12.2.1 Class RTPManager.....	208
4.12.2.2 Class UnsupportedFormatException.....	212
4.12.2.3 Class RTPContentDescriptor.....	213
4.12.2.4 Class SessionAddress.....	213
4.12.2.5 Interface ReceiveStream	214
4.12.2.6 Interface ReceiveStreamListener	214
4.12.2.7 Interface RTPStream.....	214
4.12.2.8 Interface SendStream.....	214
4.12.2.9 Interface SendStreamListener.....	215
4.12.2.10 Class InvalidSessionAddressException.....	215
4.12.2.11 Class SessionManagerException.....	215



4.12.2.12	Class <i>RTPEvent</i>	216
4.12.2.13	Class <i>ReceiveStreamEvent</i>	216
4.12.2.14	Class <i>NewReceiveStreamEvent</i> extends <i>ReceiveStreamEvent</i>	217
4.12.2.15	Class <i>SendStreamEvent</i>	217
4.12.2.16	Class <i>NewSendStreamEvent</i> extends <i>SendStreamEvent</i>	217
4.12.2.17	Class <i>StreamClosedEvent</i> extends <i>SendStreamEvent</i>	218
4.12.3	CODE EXAMPLE	218
4.12.4	Q & A	230
4.13	DISTRIBUTED SPEECH RECOGNITION	233
4.13.1	OVERVIEW	233
4.13.2	CLASS DESCRIPTION	233
4.13.2.1	CLASS <i>DSRDataSource</i>	234
4.13.2.2	INTERFACE <i>DSRListener</i>	238
4.13.3	CODE EXAMPLE	238
4.13.4	USING DSR WITH RTP	247
4.14	LIGHTING	249
4.14.1	OVERVIEW	249
4.14.2	CLASS DESCRIPTION	249
4.14.3	METHOD DESCRIPTION	249
4.14.4	TIPS /	251
4.15	VIBRATOR API	252
4.15.1	OVERVIEW	252
4.15.2	CLASS DESCRIPTION	252
4.15.3	METHOD DESCRIPTIONS	252
4.15.4	CODE EXAMPLES	253
4.15.5	TIP /	254
4.15.6	EMULATOR STUB CLASSES	254
4.16	JAVA IMAGE UTILITY LIBRARY	255
4.16.1	OVERVIEW	255
4.16.2	CLASS DESCRIPTION	255
4.16.3	METHOD DESCRIPTION	256
4.16.4	CODE EXAMPLE	260
5	TELEPHONY	269
5.1	OVERVIEW	269
5.2	INTERCONNECT/PHONE CALL INITIATION API	269
5.2.1	OVERVIEW	269
5.2.2	CLASS DESCRIPTION	269
5.2.3	METHOD DESCRIPTION	270
5.2.4	CODE EXAMPLES	272
5.2.5	COMPILING & TESTING INTERCONNECT CAPABLE MIDLETS	272
5.3	CALL RECEIVING API	273
5.3.1	OVERVIEW	273
5.3.2	CLASS DESCRIPTIONS	273
5.3.3	METHOD DESCRIPTIONS	274
5.3.3	CODE EXAMPLES	281
5.3.4	TIPS /	289
5.3.5	COMPILING & TESTING CALL RECEIVING MIDLETS	289
5.4	RECENTCALLS API	291
5.4.1	OVERVIEW	291
5.4.2	CLASS DESCRIPTIONS	291
5.4.3	METHOD DESCRIPTIONS	291



5.4.4 CODE EXAMPLES	296
5.5 PHONEBOOK	300
5.5.1 OVERVIEW	300
5.5.2 CLASS DESCRIPTIONS	300
5.5.3 CLASS METHODS	300
5.5.4 CODE EXAMPLES	307
5.5.5 COMPILING & TESTING PHONEBOOK MIDLETS	311
5.6 JAVA PIM PACKAGE	312
5.6.1 OVERVIEW	312
5.6.2 PACKAGE DESCRIPTION	313
5.6.3 CODE EXAMPLES	320
6 FILE SYSTEM AND STORAGE	324
6.1 OVERVIEW	324
6.2 MIDP 2.0 RECORD MANAGEMENT SYSTEM (RMS)	324
6.2.1 OVERVIEW	324
6.2.2 CLASS DESCRIPTION	324
6.2.3 CODE EXAMPLES	325
6.2.4 TIPS /	325
6.2.5 CAVEATS	326
6.2.6 COMPILING AND TESTING RMS MIDLETS	326
6.3 MIDP 2.0 FILE I/O AND SECURE FILE I/O	327
6.3.1 OVERVIEW	327
6.3.2 CLASS DESCRIPTION	327
6.3.3 METHOD DESCRIPTION	327
6.3.4 CODE EXAMPLES	329
6.3.5 TIPS /	338
6.3.6 CAVEATS	338
6.3.7 COMPILING AND TESTING FILE/SECURE FILE MIDLETS	339
6.4 FILECONNECTION	340
6.4.1 OVERVIEW	340
6.4.2 PACKAGE JAVAX.MICROEDITION.IO.FILE	340
6.5 JAVA ZIP	358
6.5.1 OVERVIEW	358
6.5.2 CLASS DESCRIPTION	358
6.5.3 METHOD DESCRIPTIONS	358
6.5.4 CODE EXAMPLE	358
7 NETWORKING AND SECURITY	360
7.1 OVERVIEW	360
7.2 J2ME™ NETWORKING	360
7.2.1 OVERVIEW	360
7.2.2 TIMEOUTS	361
7.2.3 PROTOCOLS	361
7.2.3.1 HTTP	361
7.2.3.2 HTTPS	362
7.2.3.3 TCP Sockets	362
7.2.3.4 SSL Secure Sockets	363
7.2.3.5 Server Sockets	363
7.2.3.6 UDP Sockets	363
7.2.3.7 Serial Port Access	364
7.2.3.8 RFCOMM, L2CAP, and OBEX	366



7.2.4 IMPLEMENTATION NOTES I.....	366
7.2.5 TIPS I.....	366
7.3 WIRELESS MESSAGING	367
7.3.1 OVERVIEW	367
7.3.2 METHOD DESCRIPTIONS	367
7.3.3 CAVEAT FOR WMA OVER SMS	369
7.4 WMA OVER MMS	370
7.4.1 PACKAGE DESCRIPTION	370
7.4.2 PACKAGE TREE	370
7.4.2.1 <i>Class Hierarchy</i>	370
7.4.2.2 <i>Interface Hierarchy</i>	371
7.4.2.3 <i>javax.microedition.io.Connector</i>	371
7.4.2.4 <i>javax.wireless.messaging.TextMessage</i>	372
7.4.2.5 <i>javax.wireless.messaging.MessageConnection</i>	373
7.4.2.6 <i>javax.wireless.messaging.MessagePart</i>	373
7.4.2.7 <i>javax.wireless.messaging.MultipartMessage</i>	375
7.4.3 CODE EXAMPLES	376
7.5 MIDP 2.0 PUSH REGISTRY.....	382
7.5.1 OVERVIEW	382
7.5.2 NETWORK LAUNCH.....	382
7.5.3 TIME-BASED LAUNCH	382
7.5.4 CLASS DESCRIPTION	382
7.5.5 METHOD DESCRIPTION	383
7.5.6 TIPS I.....	383
7.6 MIDP 2.0 SECURITY API.....	384
7.6.1 OVERVIEW	384
7.6.2 CLASS DESCRIPTIONS	384
7.6.3 METHOD DESCRIPTIONS	385
7.6.4 CODE EXAMPLES	385
7.6.5 TIPS I.....	387
7.7 CRYPTOGRAPHY APIS	388
7.7.1 OVERVIEW	388
7.7.2 CLASS DESCRIPTIONS	388
7.7.2.1 <i>MessageDigest Description</i>	388
7.7.2.2 <i>Cipher Description</i>	389
7.7.2.3 <i>Signature Description</i>	389
7.7.2.4 <i>KeyAgreement Description</i>	389
7.7.3 METHOD DESCRIPTIONS	390
7.7.4 EXAMPLE CODE	394
7.7.5 TIPS I.....	399
7.7.6 COMPILING & TESTING CRYPTOGRAPHY ENHANCED MIDLETS	399
7.8 JAXP.....	400
7.8.1 OVERVIEW	400
7.8.2 PACKAGE JAVAX.XML.PARSERS	400
7.8.3 PACKAGE ORG.XML.SAX	401
7.8.4 PACKAGE ORG.XML.SAX.HELPERS	401
7.8.5 PACKAGE TREE.....	402
7.8.5.1 <i>Class Hierarchy</i>	402
7.8.5.2 <i>Interface Hierarchy</i>	402
7.8.5.3 <i>Class javax.xml.parsers.SAXParser</i>	402
7.8.5.4 <i>Class javax.xml.parsers.SAXParserFactory</i>	403
7.8.6 CODE EXAMPLES	404



7.8.7	COMPILING & TESTING JAXP MIDLETS.....	409
7.9	JAX-RPC.....	410
7.9.1	OVERVIEW.....	410
7.9.2	PACKAGE JAVAX.MICROEDITION.XML.RPC	410
7.9.3	PACKAGE JAVAX.XML.NAMESPACE	410
7.9.4	PACKAGE JAVAX.XML.RPC	411
7.9.5	PACKAGE JAVA.RMI.....	411
7.9.6	CLASS AND INTERFACE HEIRARCHY.....	412
7.9.7	DEVELOPMENT PROCEDURE.....	413
7.9.8	BACKGROUND KNOWLEDGE.....	413
7.9.9	DEVELOPMENT STEPS.....	415
7.9.10	SAMPLE APPLICATION	415
7.10	JAVA™ APIS FOR BLUETOOTH™ WIRELESS TECHNOLOGY AND OBJECT PUSH PROTOCOL.....	425
7.10.1	OVERVIEW.....	425
7.10.2	DEVICE AND SERVICE DISCOVERY	425
7.10.2.1	PACKAGE DESCRIPTION	425
7.10.2.2	PLATFORM SPECIFIC LIMITATIONS	426
7.10.3	SERVICE REGISTRATION	426
7.10.3.1	PACKAGE DESCRIPTION	427
7.10.3.2	PLATFORM SPECIFIC LIMITATIONS	427
7.10.3.3	SAMPLE CODE	427
7.10.4	GENERIC ACCESS PROFILE	431
7.10.4.1	PACKAGE DESCRIPTION	431
7.10.4.2	PLATFORM SPECIFIC LIMITATIONS	431
7.10.4.3	SAMPLE CODE	431
7.10.5	SERIAL PORT PROFILE	433
7.10.5.1	PACKAGE DESCRIPTION	433
7.10.5.2	USING JAVAX.MICROEDITION.IO.CONNECTOR FOR RFCOMM.....	433
7.10.5.3	SAMPLE CODE	434
7.10.6	LOGICAL LINK CONTROL AND ADAPTATION PROTOCOL (L2CAP)	441
7.10.6.1	PACKAGE DESCRIPTION	441
7.10.6.2	USING JAVAX.MICROEDITION.IO.CONNECTOR FOR L2CAP	441
7.10.6.3	PLATFORM SPECIFIC LIMITATIONS	441
7.10.7	OBJECT EXCHANGE PROTOCOL	442
7.10.7.1	PACKAGE DESCRIPTION	442
7.10.7.2	USING JAVAX.MICROEDITION.IO.CONNECTOR FOR OBEX	443
7.10.7.2.1	OBEX OVER RFCOMM.....	443
7.10.7.2.2	OBEX OVER TCP/IP	444
7.10.7.3	CREATING AN OBEX CLIENT.....	444
7.10.7.4	CREATING AN OBEX SERVER.....	445
7.10.7.5	PLATFORM SPECIFIC LIMITATIONS	446
7.10.8	OBJECT PUSH PROTOCOL.....	446
7.10.8.1	PACKAGE DESCRIPTION	446
7.10.8.1.1	CLASS OPPCLIENT	447
7.10.8.1.2	CLASS OPPCLIENTREQUESTHANDLER.....	448
7.10.8.1.3	CLASS OPPSERVER	452
7.10.8.1.4	CLASS OPPSERVERREQUESTHANDLER.....	455
7.10.8.1.5	CLASS OPPOBJECT	456
7.10.8.2	SAMPLE CODE	456
8	HANDSET FEATURES.....	471
8.1	OVERVIEW	471



8.2 MIDP 2.0 PLATFORM REQUEST	471
8.2.1 OVERVIEW	471
8.2.2 CLASS DESCRIPTION	471
8.2.2.1 Method Description	472
8.2.3 CODE EXAMPLES	472
8.2.4 TIPS /	473
8.3 DATEBOOK	474
8.3.1 OVERVIEW	474
8.3.2 CLASS DESCRIPTIONS	474
8.3.3 METHOD DESCRIPTIONS	475
8.3.3.1 UDM Method	475
8.3.3.2 DateBookEvent Methods	475
8.3.3.3 DateBookRepeatEvent Methods	477
8.3.3.4 DateBook Methods	479
8.3.4 CODE EXAMPLES	481
8.3.5 COMPILING & TESTING DATEBOOK MIDLETS	484
8.4 STATUS MANAGER	485
8.4.1 OVERVIEW	485
8.4.2 CLASS DESCRIPTION	485
8.4.3 METHOD DESCRIPTIONS	485
8.5 LOCATION API	487
8.5.1 OVERVIEW	487
8.5.2 CLASS DESCRIPTION	488
8.5.3 METHOD DESCRIPTIONS	489
8.5.4 CODE EXAMPLES	492
8.5.5 TIPS /	496
8.6 JAVAX LOCATION PACKAGE	499
8.6.1 OVERVIEW	499
8.6.2 PACKAGE DESCRIPTION	499
8.6.3 JAVAX.MICROEDITION.LOCATION.LOCATION	501
8.6.4 JAVAX.MICROEDITION.LOCATION.LOCATIONPROVIDER	502
8.6.5 JAVAX.MICROEDITION.LOCATION.ORIENTATION	504
8.6.6 CODE EXAMPLES	505
8.6.7 COMPARING WITH OEM AGPS API	514
8.7 CUSTOMER CARE API	517
8.7.1 OVERVIEW	517
8.7.2 CLASS DESCRIPTION	517
8.7.3 METHOD DESCRIPTIONS	517
8.7.4 CODE EXAMPLES	520
8.7.5 COMPILING & TESTING CUSTOMER CARE MIDLETS	522
APPENDIX A: SPECIFICATION SHEETS	523
iDEN MULTI-COMMUNICATION DEVICE SPECIFICATIONS	523
APPENDIX B: JAVA APIS	525
FEATURE MATRIX FOR iDEN MULTI-COMMUNICATION DEVICES	525
APPENDIX C: KEY MAPS FOR THE iDEN MULTI-COMMUNICATION DEVICES	528
OVERVIEW	528
1730 / 1710 MULTI-COMMUNICATION DEVICE	529
1830 / 1830E MULTI-COMMUNICATION DEVICE	531
1285 MULTI-COMMUNICATION DEVICE	533



I325 MULTI-COMMUNICATION DEVICE.....	534
I860 MULTI-COMMUNICATION DEVICE.....	535
I265 MULTI-COMMUNICATION DEVICE.....	538
I605 MULTI-COMMUNICATION DEVICE.....	538



1

Introduction

1.1 Purpose

This guide describes the procedures used to develop a J2ME™ compliant application for iDEN Multi-Communication Devices. Also included is information on developing and packaging applications for installation as well as a step-by-step procedure for setting up a debug environment. Detailed information on the Java 2 Micro Edition environment is not provided.

1.2 Audience

This document is intended for application developers involved with the development of J2ME applications for iDEN Multi-Communication Devices using iDEN APIs.

1.3 Disclaimer

Motorola reserves the right to make changes without notice to any products or services described herein. "Typical" parameters, which may be provided in Motorola Data sheets and/or specifications can and do vary in different applications and actual performance may vary. Customer's technical experts will validate all "Typicals" for each customer application.

Motorola makes no warranty with regard to the products or services contained herein. Implied warranties, including without limitation, the implied warranties of merchantability and fitness for a particular purpose, are given only if specifically required by applicable law. Otherwise, they are specifically excluded.

No warranty is made as to coverage, availability, or grade of service provided by the products or services, whether through a service provider or otherwise.

No warranty is made that the software will meet your requirements or will work in combination with any hardware or applications software products provided by third parties, that the operation of the software products will be uninterrupted or error free, or that all defects in the software products will be corrected.

In no event shall Motorola be liable, whether in contract or tort (including negligence), for any damages resulting from use of a product or service described herein, or for any indirect, incidental,



special or consequential damages of any kind, or loss of revenue or profits, loss of business, loss of information or data, or other financial loss arising out of or in connection with the ability or inability to use the Products, to the full extent these damages may be disclaimed by law.

Some states and other jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, or limitation on the length of an implied warranty, so the above limitations or exclusions may not apply to you.

This warranty gives you specific legal rights, and you may also have other rights, which vary from jurisdiction to jurisdiction.

Motorola products or services are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product or service could create a situation where personal injury or death may occur.

Should the buyer purchase or use Motorola products or services for any such unintended or unauthorized application, buyer shall release, indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the designing or manufacture of the product or service.

1.4 References

Reference	Link
Motorola, Inc. IDEN Subscriber Group Phone: 1-800-453-0920	https://commerce.motorola.com/idenonline/ideveloper/index.cfm
CLDC 1.1 Library	http://jcp.org/aboutJava/communityprocess/final/jsr139/index.html .
Sun™ J2ME™	http://java.sun.com/j2me/

1.5 Revision History

Version	Date	Reason
1.7	08/23/04	Updated to include the i860.
1.85	10/09/04	Updated to include the i830e.
1.93	2/23/2005	Updated to include the i605.
1.98	3/22/2005	Updated to include the i275 and i355.



1.6 Abbreviations and Acronyms







Acronym	Description
AGPS	Assisted Global Positioning System
AMS	Application Management Software
API	Application Programming Interface
CLDC	Connected Limited Device Configuration
CSD	Customer Specific Data
CSTN	Color Super Twisted Nematic LCD
DB	Database
DRM	Data Resource Manager
DSR	Distributed Speech Recognition
DTMF	Dual Tone Multi-Frequency
FPS	Frames Per Second
GCF	Generic Connection Framework
GOEP	General Object Exchange Profile
GPS	Global Positioning System
GSM	Global System for Mobile Communications
IANA	Internet Assigned Numbers Authority
iDEN	Integrated Digital Enhanced Networks
IMEI	International Mobile Equipment Identity
JAXP	Java API for XML Processing
JAX-RPC	Java API for XML-based Remote Procedure Call
JSR	Java Specification Report
L2CAP	Logical Link Control and Adaptation Protocol
LWT	Lightweight Windowing Toolkit
MIDP	Mobile Information Device Profile
MMA	Mobile Media API
MMS	Multimedia Messaging Service
NMEA	National Marine Electronics Association
OBEX	Object Exchange
PIM	Personal Information Management
PKI	Public Key Infrastructure
RFCOMM	Radio Frequency Communications Protocol
RMS	Record Management System
RS-232	Recommended Standard 232
RTP	Real-time Transport Protocol
SAX	Simple API for XML Parser
SDDDB	Service Discovery Database
SDG	Selective Dynamic Group
SDGC	Selective Dynamic Group Call
SDP	Service Discovery Protocol
SIG	Special Interest Group
SOAP	Simple Object Access Protocol
SPI	Service Provider Interface






Acronym	Description
SPP	Serial Port Profile
SSL	Secure Socket Layer
UDM	User Data Manager
UFMI	Universal Fleet Member Identifier
WMA	Wireless Messaging API
WSDL	Web Services Description Language
XML	Extensible Markup Language



1.7 Icon Guide

Icon	Description
	Presence of this icon indicates that the information presented applies to the i730 and i710 Multi-Communication Devices. When this icon is present the information also applies to updated i730 handsets.
	Presence of this icon indicates that the information presented applies to certain i730 Multi-Communication Devices with later software updates enabling new functionality. These handsets contain all of the functionality of older i730 handsets as well. To determine if an i730 handset has the later software see the Java System section.
	Presence of this icon indicates that the information presented applies to the i830 and i830e Multi-Communication Devices. When the icon is accompanied by "i830e only" the information does not apply to standard i830 handsets.
	Presence of this icon indicates that the information presented applies to the i860 Multi-Communication Device.
	Presence of this icon indicates that the information presented applies to the i285 Multi-Communication Device.
	Presence of this icon indicates that the information presented applies to the i325 Multi-Communication Device.



	<p>Presence of this icon indicates that the information presented applies to the i265 and i275 Multi-Communication Devices. When the icon is accompanied by "i275 only" the information does not apply to i265 handsets.</p>
	<p>Presence of this icon indicates that the information presented applies to the i605 Multi-Communication Device.</p>
	<p>Presence of this icon indicates that the information presented applies to the i355 Multi-Communication Device.</p>



1.8 Document Overview

This developer's guide is organized into the following chapters and appendices:

Chapter 1 – Introduction: this chapter has general information about this document, including purpose, scope, references, and definitions.

Chapter 2 – J2ME Introduction: this chapter describes the J2ME™ platform and the available resources on the iDEN Multi-Communication Device.

Chapter 3 – Developing and Packaging J2ME™ Applications: this chapter describes the processes for developing applications and creating an emulation environment for J2ME™ applications.

Chapter 4 – Multimedia and Graphics: this chapter describes the features of the iDEN Multi-Communication Device as pertains to multimedia and graphics.

Chapter 5 – Telephony: this chapter describes the features of the iDEN Multi-Communication Device as pertains to telephony.

Chapter 6 – File System and Storage : this chapter describes the features of the iDEN Multi-Communication Device as pertains to file system and storage.

Chapter 7 – Networking and Security: this chapter describes the features of the iDEN Multi-Communication Device as pertains to networking and security.

Chapter 8 – Handset Features: this chapter describes the features of the iDEN Multi-Communication Device as pertains to handset features.

Appendix A: Specification Sheets

Appendix B: Java APIs

Appendix C: Key Maps for the iDEN Multi-Communication Devices



2

J2ME™ Introduction

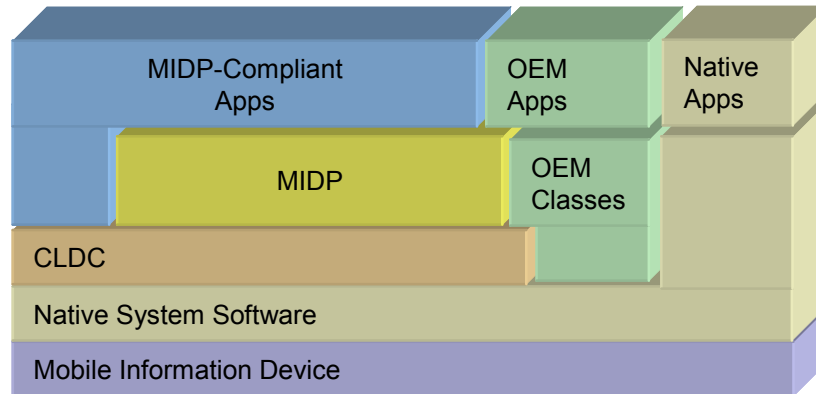
2.1 Overview

Most iDEN handsets include the Java™ 2 Platform, Micro Edition (or J2ME™) platform. The J2ME™ platform enables developers to easily create a variety of Java applications ranging from business applications to games. Prior to its inclusion, services or applications residing on small consumer devices like cell phones could not be upgraded or added to without significant effort. By implementing the J2ME™ platform on iDEN handsets service providers, as well as customers, can easily add and remove applications allowing for quick and easy personalization of each device. This chapter of the guide presents a quick overview of the J2ME™ environment and the tools that can be used to develop applications for the iDEN handsets.

2.2 The Java 2 Platform, Micro Edition (J2ME™)

The J2ME™ platform is a very small application environment. It is a framework for the deployment and use of Java technology in small devices (such as cell phones and pagers) and includes a set of APIs and a virtual machine that is designed in a modular fashion allowing for scalability among a wide range of devices.

The J2ME™ architecture contains three layers consisting of the Java Virtual Machine, a Configuration Layer, and a Profile Layer. The Virtual Machine (VM) supports the Configuration Layer by providing an interface to the host operating system. Above the VM is the Configuration Layer, which can be thought of as the lowest common denominator of the Java Platform available across devices of the same “horizontal market.” Built upon this Configuration Layer is the Profile Layer, typically encompassing the presentation layer of the Java Platform.

**Figure 2.1**

The Configuration Layer used in the iDEN handsets is either the Connected Limited Device Configuration 1.1 (CLDC 1.1) or Connected Limited Device Configuration 1.0 (CLDC 1.0), depending on the phone model. The Profile Layer used is the Mobile Information Device Profile 2.0 (MIDP 2.0). Together, the CLDC and MIDP provide common APIs for I/O, simple math functionality, UI, and more.

For more information on J2ME™, see the Sun™ J2ME™ documentation (<http://java.sun.com/j2me/>).

2.2.1 The Motorola J2ME™ Platform

Functionality not covered by the CLDC and MIDP APIs is left for individual OEMs to implement and support. By adding to the standard APIs, manufacturers can allow developers to access and take advantage of the unique functionality of their handsets.

The iDEN Multi-Communication Device contain OEM APIs for extended functionality ranging from enhanced UI to advanced data security. While the iDEN Multi-Communication Device can run any application written in standard MIDP, it can also run applications that take advantage of the unique functionality provided by these APIs. These OEM APIs are described in this guide.

2.2.2 Resources and APIs Available

MIDP 2.0 will provide support to myriad functional areas on the iDEN Multi-Communication Device: For more information on these resources and Java APIs, see Appendix A: Specification Sheets

on page 523 and Appendix B:
Java APIs

on page 525.

2.2.3 Platform Specific Features

2.2.3.1 Concurrency

iDEN's J2ME™ platform supports the concurrent execution of up to three MIDlets at a time. The three MIDlets must be from different MIDlet Suites. You can't concurrently execute two MIDlets from the same MIDlet Suite.

Only one of the MIDlets can be in the foreground at once, leaving the other two suspended in the background. As with other iDEN products, the MIDlets in the background can still execute while



they're suspended. Since resources are limited it is recommended that MIDlets be written in such a way that they release any resources such as files, large temporary heap storage, and threads.

A single instance of the VM is used to run all three MIDlets. This means that all of the MIDlets' threads are scheduled together in a round robin fashion, but the time slices vary. The threads belonging to the foreground (active) MIDlet have the largest time slices. The threads belonging to the background (suspended) MIDlets have smaller time slices. The thread's actual time slice is calculated based on this foreground/background designation as well as the thread's priority. A MIDlet can set that thread priority by using the `Thread.setPriority()` method as described in the CLDC specification. This allows the MIDlet to customize the performance of its own threads with respect to the other threads with the same foreground/background designation, basically meaning that no threads in a background MIDlet can be a higher priority than the threads in the foreground MIDlet.

All three MIDlets also operate out of the same heap. This raises the possibility of an out of memory exception because there are other applications whose memory usage is not known. The best way to safeguard against this is for application developers to minimize the amount of heap they consume while their MIDlet is in the background. Another way to avoid an out-of-memory problem is to catch the `OutOfMemoryException` while creating large resource objects. This prevents unexpected behaviors from the MIDlet.

Although MIDlets can run concurrently, they still don't have an awareness of each other. The purpose of the concurrency feature is not cooperation amongst MIDlets. The purpose is to allow the user to run multiple MIDlets, such as a game in the foreground and an instant messenger application in the background.

2.2.3.2 Multiple Key Presses

iDEN handsets support the ability to have multiple key presses passed to a Java application. This allows for applications to receive notification of another key being pressed while one key is still being pressed. Due to hardware limitations, only 2 keys presses can be guaranteed at any given time, while many key combinations of more keys can be pressed at the same time.

Multiple key press support works as follows:

- When a key is pressed, the application's `keyPressed()` method is called.
- After 650ms, the application's `keyRepeated()` method is called every 250 ms.
- When a second key is pressed, it stops the first key from repeating. The second key will generate another call to the application's `keyPressed()` method.
- After 650ms, the application's `keyRepeated()` method is called every 250 ms with the second key's key code.
- When either of the keys are released, the application's `keyReleased()` method is called with the key code of the key that was released.
- If two keys are currently pressed and if the key that is released is the repeating key, no more calls to the application's `keyRepeated()` method occur until another key is pressed.

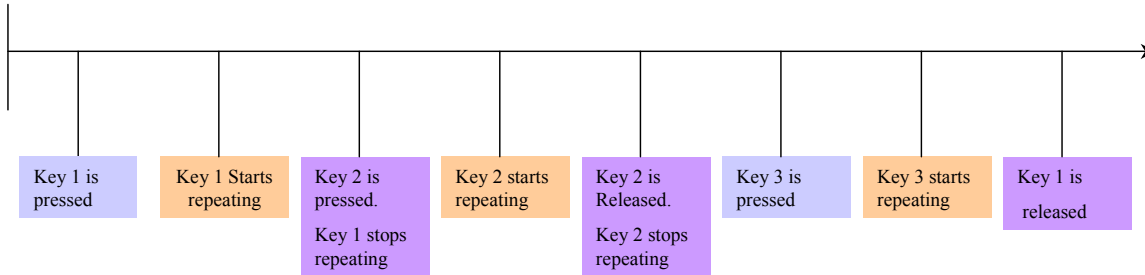
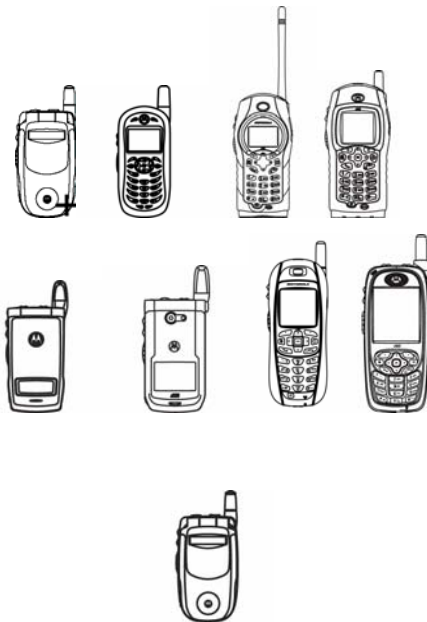


Figure 2.2 Key press timeline

2.2.3.3 Timezone and Daylight Savings Support



Local time zone and daylight savings time support has been enhanced on certain handsets. These handsets provide the locally determined time zone as the default time zone, as well as daylight savings information through the CLDC Time Zone class. For instance, if it is 5:00PM locally, and the local time zone is GMT-05:00, then `System.currentTimeMillis()` returns 10:00PM (9:00PM if daylight savings is in effect).

On handsets without enhanced locale support, the default time zone is always GMT, and `System.currentTimeMillis()` returns local time interpreted as GMT. For instance, if it is 5:00PM locally, then `System.currentTimeMillis()` returns the millisecond value of 5:00PM GMT without accounting for daylight savings.



2.3 Application Management

2.3.1 MIDlet Lifecycle

A MIDlet's lifecycle begins once its MIDlet suite is downloaded to the device. From that point, the Application Management Software (AMS) manages the MIDlet suite and its MIDlets. The user's primary user interface for the AMS is the Java Apps feature built into the device's firmware.

From the Java Apps feature, the user can see each MIDlet suite on the device or access the Java System menu item. If a MIDlet suite has only a single MIDlet, then the MIDlet's name is displayed in the Java Apps menu for that MIDlet suite. Otherwise the MIDlet suite name is displayed. Then when that MIDlet suite is highlighted, the user can open the MIDlet suite and view the MIDlets in that MIDlet suite.



Figure 2.3 The Java Apps Menu

From the Java Apps menu, the user can highlight a MIDlet suite and bring up the About dialog for that MIDlet suite. The About dialog contains:

- MIDlet Suite Name
- MIDlet Suite Vendor
- MIDlet Suite Version
- JAR Size (not installed only)
- The number of MIDlets in the MIDlet Suite
- MIDP and CLDC version requirements (not installed only)
- Flash usage, Program and Data Space (installed only)

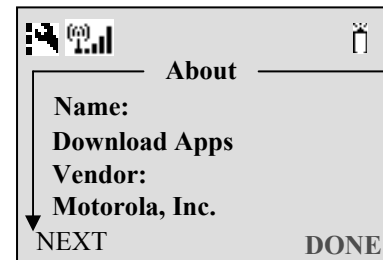


Figure 2.4 About Properties for a MIDlet

2.3.2 MIDlet Suite Installation

From the Java Apps menu, the user can install MIDlet suites. A MIDlet suite must be installed before any of its MIDlets can be executed. Installation involves extracting the classes from the JAR file and creating an image that will be placed into Program Space. The resources are then extracted from the JAR file and placed into Data Space. The JAR file is then removed from the device, thus freeing up some Data Space where it was originally downloaded.



The space savings from removing the JAR file is one advantage of Installation. However, perhaps an even greater advantage is that class loading is not done during run time. This means that a MIDlet won't experience slow-down when a new class is accessed. Furthermore, the MIDlet won't have to share the heap with classes have been class-loaded from the JAR file.

2.3.3 MIDlet Suite De-installation

An installed MIDlet can be removed from the device only by de-installing it from the Java Apps menu. De-installing a MIDlet suite removes the installed image from Program Space. The resources are then removed from Data Space along with the JAD file.

2.3.4 MIDlet Suite Updating

When a MIDlet suite is de-installed, all of its resources are removed including any resources that were created by the MIDlets in the suite, such as RMS databases. If a user gets a new version of a MIDlet suite, the user can simply download that new version to the device that has the older version. Once that new version is downloaded, the user has the option to update the MIDlet suite. This de-installs the old version and immediately installs the new MIDlet suite. The only difference is that the device asks the user whether resources such as RMS databases should be preserved while de-installing the old version. This prompt occurs only if such resources exist.

Such a scheme places the burden of compatibility on the developer. A newer version of the MIDlet suite should know how to use, upgrade, or remove the data in the RMS databases that the older versions created. This idea of forward compatibility should also extend to backward compatibility, because the device allows a user to replace a version of a MIDlet suite with an older version of that MIDlet suite.

2.3.5 Starting, Pausing, and Exiting

2.3.5.1 AMS Control of MIDlet State Transitions

A MIDlet has three different states: destroyed, active, and paused. A MIDlet's natural state is destroyed. The AMS typically controls the transition through these states. When a user decides to launch a MIDlet, the device puts up a screen indicating that the MIDlet is transitioning through these states. The AMS controls the MIDlets through those states by calling the MIDlet's methods, `startApp()`, `pauseApp()`, and `destroyApp()`.



Figure 2.5 MIDlet Starting Screen

First, the constructor of the MIDlet's class that extends MIDlet is invoked. Then its `startApp()` method is called to indicate that it's being started. The MIDlet has focus when its `startApp()` method finishes execution. If a MIDlet takes too long initializing state variables and preparing to be run in its constructor or `startApp()` methods, it may appear to be stalled to users.

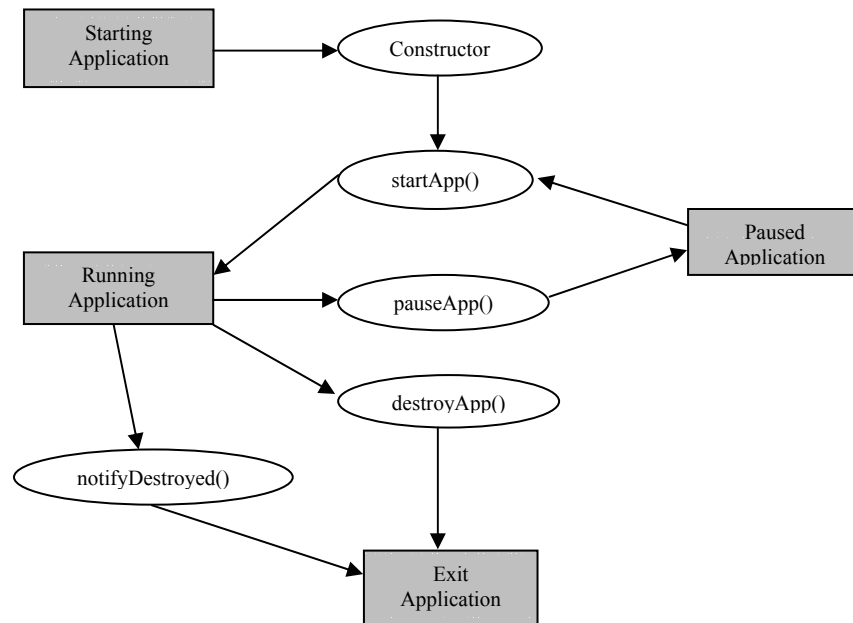


Figure 2.6 MIDlet State Transitions

State Transition Methods

Method	Caller	Purpose
Constructor	AMS	Initializes the MIDlet – should return quickly
<code>startApp()</code>	AMS	<ol style="list-style-type: none"> 1. The <code>startApp()</code> method is called to start the application either from a newly constructed state or from a paused state. 2. If the <code>startApp()</code> is called from a paused state, the MIDlet should not re-initialize the instance variables (unless that's the desired behavior). 3. The <code>startApp()</code> method may be called multiple times during the lifespan of the MIDlet. 4. The MIDlet may set the current display to its own <code>Displayable</code> from the <code>startApp()</code> method, but is shown only after the <code>startApp()</code> returns. 5. When exiting a suspended application, the KVM first calls <code>startApp()</code> followed by a call to <code>destroyApp()</code>.



Method	Caller	Purpose
<code>pauseApp()</code>	AMS, MIDlet	<ol style="list-style-type: none"> 1. The <code>pauseApp()</code> method is called from either AMS or from within the MIDlet. 2. The <code>pauseApp()</code> should pause active threads, and prepare for <code>startApp()</code> to be called. 3. If the application is to be resumed with a screen other than the present, then the Displayable should be set current in the <code>pauseApp()</code>.
<code>destroyApp()</code>	AMS	<p>The <code>destroyApp()</code> method is called from AMS and signals the MIDlet to clean up any resources and prepare for termination. For example, open RMS records should be closed, threads should be stopped, and any other housekeeping chores should be performed.</p> <p>The MIDlet should not call <code>destroyApp()</code>.</p>
<code>notifyDestroyed()</code>	MIDlet	<p>The <code>notifyDestroyed()</code> method is called by the MIDlet to exit and terminate itself.</p> <p>All housekeeping such as stopping active threads and closing RMS records should be performed before calling <code>notifyDestroyed()</code>.</p> <p><code>notifyDestroyed()</code> notifies AMS to terminate the calling MIDlet.</p>

Focus is an important concept. On a device without a windowing system, only one application can have focus at a time. When an application has focus, it receives keypad input, and has access to the display, speakers, LED lights, vibrator, and so on. MIDlets share focus with the system user interface. That user interface is a higher priority than the MIDlet, so the MIDlet will immediately lose focus when the system needs to handle a phone call or some other interrupt.

	Processing Space	
	Device Foreground	Device Background
Generic MIDP	Multiple MIDlets running in the active state.	<p>Multiple MIDlets running in the active state.</p> <p>Multiple MIDlets running limitedly in the paused state.</p> <p>Multiple MIDlets in the destroyed state.</p>
iDEN MIDP Implementation	Up to one MIDlet running in the active state.	<p>Multiple MIDlets running in the paused state.</p> <p>Multiple MIDlets in the destroyed state.</p>



Generic MIDP vs. iDEN Devices

On iDEN devices, the concept of focus correlates directly with the MIDlet state. For example, when a MIDlet loses focus because of a phone call, the MIDlet is immediately suspended. Conversely to the example of starting the MIDlet, the MIDlet loses focus immediately, then its `pauseApp()` method is called. Standard MIDP allows multiple MIDlets, where a MIDlet can be active in the foreground or active in the background. However, on the iDEN phones, an active MIDlet implies foreground and a paused MIDlet implies background.

The paused state is not clearly defined by MIDP. The only requirement placed on the device manufacturer is that a paused MIDlet must be able to respond to network events and timer events. On iDEN devices, the paused state simply implies that the MIDlet is in the background as mentioned above, but it doesn't force any of the threads to stop execution. Essentially, a paused MIDlet is a MIDlet without focus and whose `pauseApp()` method has been called. It's up to the developer to control their threads, such as making them sleep for longer periods, completely pausing game threads, or terminating threads that can be restarted when the MIDlet is made active again.

Similarly to the example of losing focus immediately before the `pauseApp()` method is called, a MIDlet's focus is also immediately lost immediately before its `destroyApp()` method is called. It's interesting to note how an iDEN device manages the transition to the destroyed state. The user's opportunity to exit a MIDlet using the AMS, is from the MIDlet's Suspended screen. Typically a MIDlet is suspended, then the user exits it. Even though it appears the MIDlet is going immediately from the paused state to the destroyed state, it actually transitions through the active state first, but it never gains focus during that transition.



Figure 2.8 MIDlet Suspended Screen

2.3.5.2 MIDlet Control of MIDlet State Transitions

A MIDlet has a lot of flexibility to control its own state. A MIDlet can call its own `startApp()`, `pauseApp()`, and `destroyApp()` methods. However those are the methods that the AMS uses to indicate a state transition to the MIDlet, so this won't actually cause the state transition. The MIDlet can simply call those methods if it wishes to perform the work that it would typically do during that state transition.

There is another set of methods that the MIDlet can use to cause the actual state transitions. They are `resumeRequest()`, `notifyPaused()`, and `notifyDestroyed()`. Since the system user interface has priority, a MIDlet cannot force itself into the active state, but it can request that it be resumed with `resumeRequest()`. If the system is not busy, then it will automatically grant the request. However if the device isn't in the idle screen, then it displays an alert dialog to ask whether the user would like to resume the MIDlet. If the user denies the request, the MIDlet is not notified. If the user grants the request, the MIDlet's `startApp()` method is called, and it gains focus when that method finishes.

The MIDlet does have more control when it decides that it wants to be paused or destroyed. It would perform the necessary work by calling its own `pauseApp()` or `destroyApp()` method, then it notifies the AMS of its intentions by calling `notifyPaused()` and `notifyDestroyed()` appropriately. Once notified, the AMS changes the MIDlet's state and revokes focus.



2.3.6 Java System

Besides managing MIDlet suites from the Java Menu, you can also perform system maintenance. The Java System feature gives statistics about the system such as:

- CLDC Version
- MIDP Version



- WMA Version
- MMAPI Version

- Data Space Free
- Program Space Free
- Total Heap Space

Besides getting statistics, you can reset the Java System or format the Java System. Resetting the Java System simply re-initializes the components of each MIDlet suite as if the device was just powered up. Formatting the Java System actually removes every MIDlet suite by completely formatting the components of each MIDlet suite. These features can be accessed by pressing the Menu key while highlighting Java System then selecting "Delete All".



Note that "Delete All" was formerly labeled "Format System".



Some i730 handsets have software updates. These handsets feature CLDC 1.1. To determine if your i730 has been updated check the reported CLDC version in Java System.

2.3.7 Java From Main Menu

Previously the Java Apps menu was the only interface into the Java functionality of an iDEN device. However, the main menu has been enhanced to allow the user to add links to MIDlets or entire MIDlet suites to the main menu. When a MIDlet is added to the main menu, the name of the individual MIDlet is used for the main menu text. When a MIDlet suite has multiple MIDlets, the



MIDlet suite itself can be added and the main menu text will be the MIDlet suite name. When a MIDlet suite is selected from main menu, the device opens that MIDlet suite and displays the MIDlets as if it were opened from the Java Apps menu.

2.3.8 Personalizing the Native UI

In addition to adding MIDlets and MIDlet suites to the main menu, MIDlets and MIDlet suites can be accessed via the following user interfaces:



- Home Screen Soft keys
- Shortcuts
- Power Up App
- Datebook Events
- Home Screen Five-Way Navigation Keys

The first three items are additional places where the user can press a key or key sequence to launch the MIDlet or MIDlet suite. The Power Up App feature allows you to launch a MIDlet suite when the phone powers up. The user can also specify a MIDlet to be launched when a Datebook event occurs.

These various features are options for the user, but as a developer you may want to encourage users to set your MIDlet up with one of these features. However, from your MIDlet you can specify a MIDlet to be added to a datebook event through the Datebook API (see “8.3 DateBook” on page 474).

2.3.9 Low Memory Indication



A low memory warning notice will be displayed to the user when the Virtual Machine is running very low on memory. Two thresholds have been defined. The first one is referred to as Low Memory Threshold Value, which is the smaller value of 64kb and the value of 10% of the Java Heap Size. The second is referred to as Extremely Low Memory Threshold, and is the smaller value of 16k bytes and the value of 2.5% of the Java Heap Size.

A Java Memory Low notice will be displayed when the available free Java memory is less than or equal to the Low Memory Threshold Value, but greater than the Extremely Low Memory Threshold Value. Any value between these two limits are said to be within the Java Low Memory boundary.



A Java Memory Extremely Low notice will be displayed to the user when the available free memory is less than or equal to the Extremely Low Memory Threshold Value. This feature is intended to inform the user that the current Java system is experiencing a low memory situation and that java applications that are currently running might potentially be adversely affected.

2.3.9.1 Notes I

Each notice will only be displayed once per Java VM lifecycle. Even during this lifecycle, the Java system might hit into the Low Memory or Extremely Low Memory scenario multiple times. (The start of a Java VM lifecycle is when the user starts the first MIDlet, and it ends when the last MIDlet exits.)

If a MIDlet attempts to allocate a block of memory that cannot be obtained even after garbage collection and compaction of the Java heap, then one of these two notices will be shown depending on how much heap is remaining.

2.3.10 The miniJIT

The miniJIT is iDEN's Just In Time (JIT) compiler technology that optimizes computation intensive code within a MIDlet suite. The miniJIT works by identifying code that can be optimized and compiling the Java byte codes to native code because native code executes faster. During the compilation, the miniJIT can also eliminate some validity checks required in the Java interpreter. Finally, the miniJIT can accelerate method calls, one of the most common constructs in any Java application, using a technique called the fast method call.

- The miniJIT is made of two separate components:
- The ahead of time code analyzer
- The optimized Java to native compiler

The ahead of time code analyzer identifies the performance crucial code in a MIDlet suite during the installation of the MIDlet suite. The optimized compiler compiles the code that the code analyzer identifies as performance crucial into native code. During the installation of the MIDlet suite, the compiler also searches for methods that can use the miniJIT's fast method call capabilities and modifies the code appropriately to use this capability.

Unlike most JIT compilers, the miniJIT compiles code only at install time. Most JIT compilers compile the code when the application is running. The miniJIT does not do so because it runs on a deeply embedded device. By compiling ahead of time, the miniJIT also eliminates the unpredictability of the dynamic behavior of most JIT compilers.

As with most optimization technologies, there are specific trade-offs in using the miniJIT. The miniJIT increases the size of the installed MIDlet suite by approximately 20% on device since it compiles the compact Java byte codes to native code. Using the miniJIT also increases install time of the MIDlet suite since the code analyzer and compiler execute during this time.

While the miniJIT does optimize computationally intensive tasks, many common capabilities are already optimized for the iDEN platform and therefore the miniJIT provides little or no additional benefit. These capabilities include:

- Graphics (LCDUI)
- Image Processing
- RMS



2.3.10.1 Using the miniJIT



On these handsets the miniJIT does not compile code during installation by default. The following must be added to the JAD file of the MIDlet suite to use the miniJIT:

```
iDEN-MIDlet-miniJIT: on
```

Other handsets use the miniJIT by default.

The iDEN-MIDlet-miniJIT attribute is checked during the installation of a MIDlet suite to determine if the miniJIT should be used. If the value is set to "on", the miniJIT is used to compile the MIDlet suite code. If the attribute is absent or set to "off", the miniJIT is not used.

2.3.10.2 Tips /

- 💡 The miniJIT optimizes only the code within the MIDlet suite.
- 💡 The miniJIT excels at optimizing loops and computation intensive code.
- 💡 The miniJIT can use the fast method call if the method that is being called meets the following conditions:
 - The method must be performance crucial.
 - The method is not abstract or synchronized.
 - The method does not have any exception handlers.
 - The method must be static or have no overriding methods.
 - The method must not allocate any memory from the Java heap.
 - All the methods that this method calls must also meet all the criteria listed here.
- 💡 The miniJIT will not improve the performance of LCDUI code, loading of Images, and RMS code. It may improve the performance of the supporting code within the MIDlet suite.
- 💡 In order to use the miniJIT, the MIDlet suite must be installed.
- 💡 As with all optimization techniques, only on-device testing lets a developer predict the changes to the user experience the optimization may have.



3

Developing and Packaging J2ME™ Applications

3.1 Overview

The iDEN Multi-Communication Device includes the J2ME™ platform. This chapter of the guide presents a comprehensive guide on developing and packaging J2ME™ applications for the iDEN Multi-Communication Device.

3.2 Developing for J2ME™

3.2.1 Developing – Tools and Emulation Environments

In order to develop applications for a J2ME™ enabled device, a developer needs some specialized tools to improve development time and prepare the application for distribution. There are several tools available in the space, so this overview is included to help enable developers to make an informed decision on these tools.

3.2.1.1 Features to Look For

Numerous tools for developing J2ME™ applications are readily and freely available on the market. Some of their features include:

- *Class libraries.* J2ME™ tools include class files for the standard CLDC/MIDP specifications and may also contain class files needed to compile device specific code.

One of the main characteristics of the MIDP standards is the lack of device specific functionality. As a solution, many MIDP device manufacturers have implemented Licensee Open Classes that provide the features requested by developers. In order to take advantage of these APIs, choose an SDK that natively supports them or one that can be upgraded to support them.
- *API documentation.* In addition to providing the class files, most SDKs include reference documentation for the supported APIs. These documents, typically found in either a HTML



or PDF format, cover the standard CLDC/MIDP specifications as well as the device specific APIs.

- *Emulation environment.* Although not an absolute necessity if the device is available, most toolkits provide this functionality for multiple devices. The main benefits of an emulation environment are the reduction in development time as well as the ability to develop for devices not yet on the market. The extent to which the toolkits emulate the device can vary greatly.

If most of the development is going to take place on the device, then this may not be a big consideration, but if access to the target device is limited or unavailable, accurate emulation is a must. Look for accuracy in the font representation, display dimensions, and pixel aspect ratio, as many wireless devices do not have square pixels.

Along the same lines as accurate look and feel, the tool should also provide accurate performance emulation. A comprehensive tool should provide individual adjustments for performance aspects such as network throughput, network latency, persistent file system access time, and graphics performance. Ideally, these attributes should not only match the target device, but also have the ability to be manually adjusted.

- *Application packaging utility.* Most SDKs automatically package the application for deployment onto the target device. Although many tools include this feature, flexibility varies widely. Look for a tool that generates both the manifest and JAD files with the required tags as well as custom tags. The packaging steps required to deploy an application are described in a later section.

3.2.2 Packaging – Putting the Pieces Together

Once an application has been tested on an emulator and is ready for testing on the actual device, the next step is to package the application and associated components into a JAD/JAR file pair. The files contain both the MIDlet's executable byte code along with the required resources. Although this process is automatically performed by most SDKs and IDEs supporting J2ME™, the steps are explained and outlined here.

3.2.2.1 Compiling .java Files to .class Files

Compiling a J2ME™ application is no different than any other J2SE™/J2EE™ application. By adding the CLDC/MIDP files (whether functional or stubbed out) to the classpath, any standard Java compiler that is JDK1.2 compliant or greater is sufficient to produce .class files suitable for the preverification step.

3.2.2.2 Preverifying .class Files

Class files destined for the KVM must undergo a modified verification step before deployment to the actual device. In the standard JVM found in J2SE™, the class verifier is responsible for rejecting invalid classes, classes that are not compatible, and classes that have been modified manually. Since this verification step is processor and time intensive, it is not ideal to perform verification on the device. In order to preserve the low-level security model offered by the standard JVM, the bulk of the verification step is performed on a desktop/workstation before loading the class files onto the device. This step is known as preverification.

During the preverification step, the class file is analyzed and a stack map is appended to the front of the file. Although this may increase the class file size by approximately 5%, it is necessary to ensure the class file is still valid when it reaches the target device. The standard J2SE™ class verifier ignores these attributes, so they are still valid J2SE™ classes.



3.2.2.3 Creating a Manifest File with J2ME™ Specific Attributes

In addition to the class files, a manifest file for a MIDlet needs to be created. Although most J2ME™ tools will auto generate the manifest file, it can also be created manually using a plain text editor. The following is a sample manifest file for a HelloWorld MIDlet:

```
MIDlet-Name: HelloWorld
MIDlet-Version: 1.0.0
MIDlet-Vendor: Motorola, Inc.
MIDlet-1: HelloWorld, , com.motorola.midlets.helloworld.HelloWorld
MicroEdition-Profile: MIDP-2.0
MicroEdition-Configuration: CLDC-1.0
```

The device's AMS uses the manifest file to determine the number of MIDlets present within the suite as well as the entry point to each MIDlet. Additionally, the manifest files may contain optional tags that are accessible by the MIDlets within the MIDlet suite. For more information, refer to the MIDP 2.0 specifications.

Keep in mind these notes when creating a manifest file:

- The following attributes are mandatory and must be duplicated in both the JAR file manifest and the JAD file. If the attributes are not identical, the application will not install.

```
MIDlet-Name
MIDlet-Version
MIDlet-Vendor
```

- The manifest contains `MIDlet-<n>` arbitrary attributes each describing a MIDlet in an application suite.
- The `MIDlet-1` attribute contains three comma-separated fields: the application name, the application icon, and the application class file (entry point). The name is displayed in the AMS user interface to represent the n^{th} application. To take advantage of MIDlet icon support, please refer to section 3.3. The application class file is the class extending the `javax.microedition.midlet.MIDlet` class for the n^{th} MIDlet in the suite.
- The manifest file is case sensitive.
- The manifest must be saved in a file called `MANIFEST.MF` (case sensitive) within the `meta-inf` directory.

3.2.2.4 JARing .class Files and Other Resources

Once the application is ready to be packaged for the device, its class files and associated resources must be bundled in a Java Archive (JAR) file. The JAR file format enables a developer to bundle multiple class files and auxiliary resources into a single compressed file format. The JAR file format provides the following benefits to the developer and end-user:

- *Portability.* The file format is platform independent.
- *Package Sealing.* All classes in a package must be found in the same JAR file.
- *Compression.* Files in the JAR may be compressed, reducing the amount of storage space required. Additionally, the download time of an application or application suite is reduced.



3.2.2.5 Creating the JAD File

Although the Java Application Descriptor (also known as an Application Descriptor File) is optional in the MIDP 1.0 specification, J2ME™ applications targeted for Motorola iDEN devices must include a JAD/JAR pair. The following is a sample JAD file for a simple HelloWorld application.

```
MIDlet-Name: HelloWorld
MIDlet-Version: 1.0.0
MIDlet-Vendor: Motorola, Inc.
MIDlet-Jar-URL: http://www.motorola.com
MIDlet-Jar-Size: 1939
MIDlet-Description: A sample HelloWorld application.
```

The JAD file may be created with any text editor and saved with the same file name prefix as the JAR file. The mandatory `MIDlet-Name`, `MIDlet-Version`, `MIDlet-Vendor` must be duplicated from the JAR file manifest. JAR files containing manifests that do not match the JAD file will not be installed.

Keep these notes in mind when creating the JAD file:

- The file names of the JAD and JAR are required to be identical except for the file extension. For example the JAR file for the `HelloWorld.jar` must be named `HelloWorld.jar`.
- The JAD file is case sensitive. All required attributes in the JAD file must start with "MIDlet-" followed by the attribute name.
- The total file length is limited to 16 characters, including the `.jad` and `.jar` extensions. For example, `HelloWorld.jar` occupies 14 characters.
- The `MIDlet-Jar-Size` must contain the accurate size of the associated JAR file. The number is in bytes.
- It's also important to note that these fields must have associated values with them. Example: "MIDlet-Name: " is not valid but "MIDlet-Name: Snake" is valid.

For more information regarding the JAD file, please refer to the MIDP 1.0 specification.

3.2.3 Desktop to Device

Now that the application is packaged, it is ready to be loaded on to the device. Applications are categorized into two distinct categories: networked and walled garden. Applications that fall into the networked category use services such as packet data (examples include HTTP, sockets, UDP, etc) to retrieve information from a remote server. Typical examples of this include applications like web browsers that use packet data services to retrieve content from remote web servers located on the open Internet. Walled garden applications, on the other hand, are stand-alone applications that do not make use of packet data services. These applications contain all the necessary data locally. Typical applications in this category include games, conversion utilities, etc.

Motorola distributes two Java Application Loading tools: JAL Lite and WebJAL. Walled Garden applications can be loaded with either WebJAL or JAL Lite. Networked applications, on the other hand, can only be loaded via the WebJAL utility. Both tools are available at www.idendev.com.



3.2.4 Debugging – Terminal Interface

As with most application development lifecycles, 10 percent of the time is spent doing the first 90 percent and 90 percent of the time is spent doing the last 10 percent. Since debugging the application is inevitable, setting up a debug environment on the phone is quite desirable.

3.2.4.1 HyperTerminal

The tool typically used to debug on the device is HyperTerminal, found at <http://www.hilgraeve.com>. HyperTerminal is also included with Windows NT and is accessible under the accessories menu. The HyperTerminal acts as a terminal for J2ME™ applications residing on the device. For example, in the standard Java 2 Platform, if a

```
System.out.println("Something")
```

is issued, the output is displayed in the terminal from which the Java application was launched. The same reasoning applies to applications residing on the device. The following instructions describe the necessary steps required to setup HyperTerminal for an iDEN MIDP device. Note – a data cable is also required for debugging.

1. Start the HyperTerminal applications by selecting Start -> Accessories -> HyperTerminal -> HyperTerminal from the “Start” menu.
2. From within the HyperTerminal application, select the File -> New Connection menu item from the drop down menus located at the top of the application.
3. Choose a name as well as an icon for the new connection, something like “iDEN”, and click on the “Ok” button.



Figure 3.1 Creating a New HyperTerminal Connection



4. Select the Communication port the data cable is connected to, typically COM1 or COM2. Click OK. A properties dialog box appears.



Figure 3.2 Setting The Connection Communication Port

5. Configure the bits per second to coincide with the Baud Rate set on the iDEN phone. Set the Data bits to 8, Parity to None, Stop bits to 1, and Flow control to Hardware.

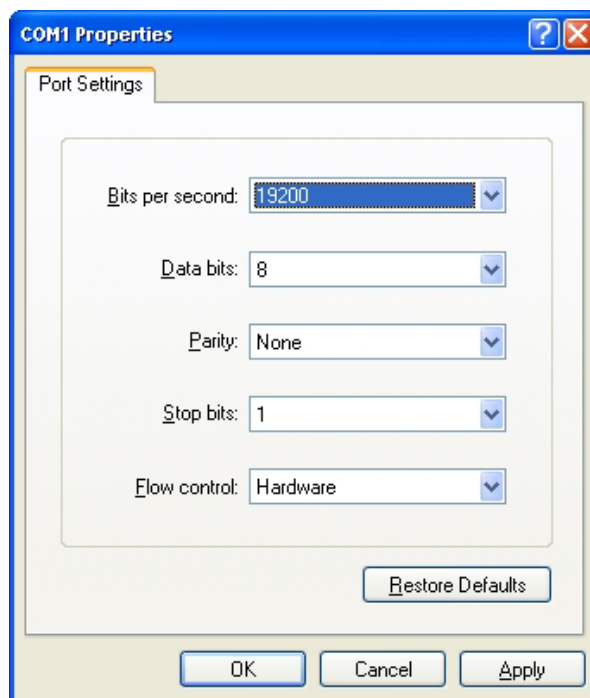


Figure 3.3 Setting The Communication Port Properties



6. Once all the parameters have been set, save the profile. To save the profile, select File->Save. The profile is saved as the connection name plus an .ht extension. For the example, the profile is saved as emulator.ht.

The profile can be launched from the Start->Accessories>HyperTerminal->iDEN menu choice.

3.2.4.2 Java Debug

To turn on Java debug, the following AT commands must be issued to the phone via the HyperTerminal in one of two ways:

3.2.4.2.1 Keyboard Input

From the keyboard, type the following AT command to turn on Java debug.

```
AT+WS46=252;+WS45=0;+IAPPL=2;D
```

The previous command turns on the Java debug statements for the current HyperTerminal session.

3.2.4.2.2 Text File Transfer

The previous AT command listed above can also be saved in a text file and transferred to the device via the HyperTerminal. To transfer the text file, select the Transfer->Send Text File menu command or press <alt> + t.

3.2.4.2.3 Notes I:

Java debug is turned on only for a particular HyperTerminal session. If the data cable is disconnected or 'Disconnect' button on the HyperTerminal is pressed, the previous sequence must be repeated to re-enable Java debug.

Debug information may not appear on the HyperTerminal if extra control characters are buffered. Type "AT" in the HyperTerminal to check the connection status. If an "OK" is returned then the connection does not contain buffered characters. To turn the echo on, type "ATE1".

Ensure the data communication rate for the phone coincides with the bits per second on the HyperTerminal. If the data rates are different, debug messages will not appear



3.2.4.3 Method Tracing

Once Java Debug is turned on, method tracing can be turned on. To see the menu of commands available, type <m> on the keyboard. The following is a sample of navigating through this menu.

```
At
OK
AT+WS46=252;+WS45=0;+IAPPL=2;D
OK
m
    M          - Menu
    TM [On/Off] - Trace Methods
    TMM [On/Off] - Trace Motorola Methods
    TMJ [On/Off] - Trace J2ME™ Methods

>tm on
Method Tracing On
>tmm on
Motorola Method Tracing On
>tmj on
JAVAX Method Tracing On
>
```

Motorola method tracing tracks any methods within Motorola extensions to the base classes. J2ME™ method tracing will track all method calls within the standard J2ME™ methods.

3.2.4.4 Debug Statements

Debugging J2ME™ applications is very similar to debugging typical Java 2 Platform applications. The most common method of debugging applications is to place `System.out.println()` statements in strategic locations. A simple way to create a debug and production version of the application at compile time is to encapsulate the `System.out.println()` statements in `if...then` clauses, with the `if` conditional checking a static Boolean variable. See the following code example:

```
class TestClass{

    private static final boolean debug = true;

    public static void main(String args[]){
        if (TestClass.debug){
            System.out.println("Debug turned on");
        }
    }
}
```

By changing the debug flag at compile time, debug statements can be easily turned on and off. For more information on debugging J2ME™ Java applications, see

<http://developer.java.sun.com/developer/onlineTraining/Programming/JDCBook/collect.html>.



3.2.5 Beyond Standards

In addition to supporting the MIDP 2.0 specification, the iDEN Java platform contains extensions in the JAD file that reflect the increased capabilities of the device. These new extensions were created to support features such as Internationalization (I18n). Although these specifications are beyond the standard MIDP 2.0, their existence is necessary to provide features requested by both the international and domestic development community. The following sections detail the specifics of each extensions as well as format and syntax.

3.2.5.1 Making it Global – Internationalization (I18n)

This phone is an I18n-ized J2ME™ platform, enabling developers to provide and display different languages. Since the MIDP 2.0 specification does not address the issue of multi-language support for MIDlet attributes, iDEN specific attribute tags were created to provide developers with this functionality. Prior to the I18n-ized J2ME™ platform, developers could only display MIDlet suite, vendor, and friendly MIDlet names in one language, and only in basic and extended ASCII characters.

Since, the MIDlet suite, vendor, and friendly names can be in different languages, the ADF file must support multi-language friendly format such as Unicode. The UTF-8 format is used to support multiple languages. Using this I18n-ized J2ME™, you can present English, Spanish, French, Portuguese, and Korean text for your MIDlet attributes. Please note that the domestic phone will only support English, French, Portuguese, and Spanish.

For more information on UTF-8 format, visit: www.unicode.org.

The following are additional MIDlet attributes in the I18n-ized J2ME™:

I18n MIDlet Attributes For iDEN Handsets

Attribute Name	Attribute Description
iDEN-MIDlet-Name-xx	The name of the MIDlet suite that identifies the MIDlets to the user in xx language.
iDEN-Vendor-xx	The organization that provides the MIDlet suite in xx language.
iDEN-MIDlet-xx-<n>	The name, icon and class of the n th MIDlet in the JAR file separated by a comma in xx language.
xx represents the language code. For example, en for English, es for Spanish, pt for Portuguese, and fr for French	



The following ADF for I18n-ized MIDlet contains following attributes:

```
MIDlet-Name: Snake
MIDlet-Vendor: Motorola
MIDlet-Version: 1.0.0
MIDlet-Jar-URL: Snake.jar
MIDlet-Jar-Size: 5000
MIDlet-1: Snake, , com.motorola.snake.Snake
```

```
iDEN-MIDlet-Name-ko: 스네이크
iDEN-MIDlet-Vendor-ko: 모토로라
iDEN-MIDlet-ko-1: 스네이크, , com.motorola.snake.Snake
```

```
iDEN-MIDlet-Name-es: Serpiente
iDEN-MIDlet-Vendor-es: Motorola
iDEN-MIDlet-es-1: Serpiente, , com.motorola.snake.Snake
```

```
Gray    - MIDP Specification
Green   - iDEN Korean Extensions
Yellow  - iDEN Spanish Extensions
```

The format of the JAD file is a sequence of lines consisting of an attribute name followed by a colon, the value of the attribute, and a carriage return. The attributes iDEN-MIDlet-Name-ko, iDEN-MIDlet-Vendor-ko, and iDEN-MIDlet-ko-1 display the MIDlet suite, vendor, and friendly names in Korean accordingly. Language specific suite name, vendor name, and MIDlet name tags will be used when the phone's language setting matches specified language attributes in the JAD file. If special language attributes are not specified in the JAD file, the phone will use the default English MIDlet suite, vendor, and friendly names.

Manifest files remain in ASCII format and must follow the specifications in MIDP 2.0.



3.2.6 ResourceBundle

3.2.6.1 Overview



This API is only available on these handsets.

Resource Bundle acts as a transparent interpretation mechanism between MIDlets and data resources. MIDlets are unaware of how the data resources are managed. MIDlets can access their data resources according to a particular locale using the APIs provided by Resource Bundle. The Resource Bundle API has no knowledge of the content of the data resources. The Resource Bundle API simply transfers stored data resources to MIDlets, which have the exclusive responsibility of reading, processing and utilizing data resources.

3.2.6.2 Package `com.motorola.iden.resourcebundle`

The API for ResourceBundle is located in package `com.motorola.iden.resourcebundle`.

Class Summary	
Locale	A Locale object represents a specific geographical, political, or cultural region.
ResourceBundle	ResourceBundle contains locale-specific string resources for MIDlets.
ListResourceBundle	ListResourceBundle is an abstract subclass of ResourceBundle that manages resources for a locale in a convenient and easy means to use list.
PropertyResourceBundle	PropertyResourceBundle is a concrete subclass of ResourceBundle that manages resources for a locale using a set of static strings from a property file.
SystemResourceBundle	SystemResourceBundle contains locale-specific system string resources for applications.
Exception Summary	
MissingResourceException	Signals that a resource is missing.



3.2.6.3 Package Tree

The following is the Class Hierarchy for the Resource Bundle API.

- o `java.lang.Object`
 - `com.motorola.iden.resourcebundle.Locale`
 - `com.motorola.iden.resourcebundle.ResourceBundle`
 - `com.motorola.iden.resourcebundle.PropertyResourceBundle`
 - `com.motorola.iden.resourcebundle.ListResourceBundle`
 - `com.motorola.iden.resourcebundle.SystemResourceBundle`
 - `java.lang.Throwable`
 - `java.lang.Exception`
 - ▣ `java.lang.RuntimeException`
 - ❖ `com.motorola.iden.resourcebundle.MissingResourceException`

3.2.6.4 CLASS Locale

3.2.6.4.1 getDefault()

Gets the current value of the default locale.

```
public static Locale getDefault()
```

3.2.6.4.2 Locale(String language)

Constructs a locale from a language code.

```
public Locale(String language)
```

The language code should not be null, otherwise a `NullPointerException` will be thrown.

Tip: The language code should be lowercase two-letter ISO-639 code.



3.2.6.4.3 Locale(String language, String country)

Constructs a locale from a language code, and a country code.

```
public Locale(String language, String country)
```

The language code and the country code should not be null, otherwise a `NullPointerException` will be thrown.

Tips:

- The language code should be lowercase two-letter ISO-639 code.
- The country code should be uppercase two-letter ISO-3166 code.
- The language code can be retrieved by the method `getLanguage`.
- The country code can be retrieved by the method `getCountry`.
- The method `toString` will output the language code combined with the country code separated by “_”.

3.2.6.5 CLASS ResourceBundle

3.2.6.5.1 getBundle(String baseName)

Gets a bundle using the specified base name and the default locale.

```
public static final ResourceBundle getBundle(String baseName)
```

3.2.6.5.2 getBundle(String baseName, Locale locale)

Gets a bundle using the specified base name and locale.

```
public static final ResourceBundle getBundle(String baseName,  
Locale locale)
```

Tips:

- The `baseName` should be the base name of the bundle, and it should be a qualified class name.
- The locale is for the bundle that is desired. If the locale is not provided, the method `getBundle` will use the default locale instead.
- You should confirm that the resources are available before `getBundle` is called. Otherwise, a `MissingResourceException` will be thrown.
- After calling `getBundle`, you can use the method `getLocale` to test which bundle is loaded.
- The class `ResourceBundle` is abstract, so you must extend it and implement the two abstract methods `handleGetObject` and `getKeys`. We have implemented one subclass for you, which is `PropertyResourceBundle`. You only need to provide the resources in a property-format text file with its name following the bundle naming convention. You can also extend the subclass `ListResourceBundle` by implementing the abstract method `getContents`. The resource contents for `ListResourceBundle` should be in a two-dimension object array.



3.2.6.5.3 getString(String key)

Gets a string for the given key from this bundle or one of its parents.

```
public final String getString(String key)
```

Tip: After getting the bundle, getString can be called to get the specific resource corresponding to the particular key. Alternatively, getObject could be called to get the same resource.



3.2.6.6 CLASS SystemResourceBundle

3.2.6.6.1 getAvailableLocales ()

Returns the set of Locales for which the system supports.

```
public static Locale[] getAvailableLocales ()
```

Tips:

- This method can retrieve the supported locales of the handset. After getting the locales we can construct an instance of SystemResourceBundle using one of these locales. If you use a locale beyond these locales, the constructor will automatically use "en_US" as default.
- After creating the instance, the method getString can be called to get a specific resource from the handset. The resource ID should be one of the following:

```
SystemResourceBundle.STRING_SKEY_OK,  
SystemResourceBundle.STRING_SKEY_BACK,  
SystemResourceBundle.STRING_SKEY_NEXT,  
SystemResourceBundle.STRING_SKEY_EXIT,  
SystemResourceBundle.STRING_SKEY_RETRY,  
SystemResourceBundle.STRING_SKEY_SAVE,  
SystemResourceBundle.STRING_SKEY_ON,  
SystemResourceBundle.STRING_SKEY_OFF,  
SystemResourceBundle.STRING_SKEY_PLAY,  
SystemResourceBundle.STRING_SKEY_PAUSE,  
SystemResourceBundle.STRING_SKEY_START,  
SystemResourceBundle.STRING_SKEY_STOP,  
SystemResourceBundle.STRING_SKEY_SELECT,  
SystemResourceBundle.STRING_SKEY_DONE,  
SystemResourceBundle.STRING_COMM_YES,  
SystemResourceBundle.STRING_COMM_NO,  
SystemResourceBundle.STRING_COMM_ENTRY_METHOD,  
SystemResourceBundle.STRING_COMM_LANGUAGES,  
SystemResourceBundle.STRING_COMM_NUMERIC,  
SystemResourceBundle.STRING_COMM_IGNORE,  
SystemResourceBundle.STRING_COMM_CANCEL,  
SystemResourceBundle.STRING_COMM_LEFT,  
SystemResourceBundle.STRING_COMM_CENTER,  
SystemResourceBundle.STRING_COMM_RIGHT,  
SystemResourceBundle.STRING_COMM_DATE,  
SystemResourceBundle.STRING_PREDICTIVE,  
SystemResourceBundle.STRING_MULTI_TAP,  
SystemResourceBundle.STRING_SYMBOLS,  
SystemResourceBundle.STRING_KOREAN_SYL,  
SystemResourceBundle.STRING_HELP,  
SystemResourceBundle.STRING_MENU_TITLE_MENU,  
SystemResourceBundle.STRING_JUSTIFICATION.
```

- You can also use the method getLocale to obtain the current locale setting of the instance of SystemResourceBundle, and use setLocale to modify the locale setting.



3.2.6.7 Code Examples

The following is the code example of ResourceBundle. The code example is divided into five parts: example for Locale, example for ResourceBundle, example for PropertyResourceBundle, example for ListResourceBundle, and example for SystemResourceBundle.

3.2.6.7.1 Code Examples for Locale

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import com.motorola.iden.resourcebundle.Locale;

public class DemoLocale extends MIDlet
    implements CommandListener{
    private Locale locale1;
    private Command getLanguageCommand, getCountryCommand,
    toStringCommand, getDefaultCommand, backCommand, exitCommand;
    private Display display;
    private List mainList, dispList;

    public DemoLocale(){
        getLanguageCommand = new Command("getLanguage",
        Command.SCREEN,1);
        getCountryCommand = new Command("getCountry",
        Command.SCREEN,1);
        toStringCommand = new Command("toString",
        Command.SCREEN,1);
        getDefaultCommand = new Command("getDefault",
        Command.SCREEN,1);
        backCommand = new Command("Back", Command.BACK,1);
        exitCommand = new Command("Exit", Command.EXIT, 1);

        display = Display.getDisplay(this);

        mainList = new List("Locales", List.IMPLICIT);
        mainList.addCommand(getLanguageCommand);
        mainList.addCommand(getCountryCommand);
        mainList.addCommand(toStringCommand);
        mainList.addCommand(getDefaultCommand);
```



```
mainList.addCommand(exitCommand);
mainList.setCommandListener(this);

dispList = new List("Results", List.IMPLICIT);
dispList.addCommand(backCommand);
dispList.setCommandListener(this);
display.setCurrent(dispList);
}

public void startApp(){
    mainList.append("\zh_CN", null);

    locale1 = new Locale("zh", "CN");
    dispList.append("Constructing(\zh\", \"CN\"):
successful", null);
}

public void pauseApp(){
}

public void destroyApp(Boolean unconditional){
}

public void commandAction(Command c, Displayable s){
    if (c == getLanguageCommand){
        display.setCurrent(dispList);

        dispList.append("(\zh\", \"CN\")'s language code is: "
+ "\"\" + locale1.getLanguage() + "\"\", null);
    } else if (c == getCountryCommand){
        display.setCurrent(dispList);

        dispList.append("(\zh\", \"CN\")'s country code is: "
+ "\"\" + locale1.getCountry() + "\"\", null);
    } else if (c == toStringCommand){
        display.setCurrent(dispList);
    }
}
```



```

        dispList.append("("zh","CN")'s toString is: " +
        "\"" + locale1.toString()+ "\",null);
    } else if (c == getDefaultCommand){
        display.setCurrent(dispList);

        dispList.append("getDefault's language code is: " + "\"" +
        Locale.getDefault().getLanguage()+ "\",null);
        dispList.append("getDefault's country code is: " + "\"" +
        Locale.getDefault().getCountry()+ "\",null);
        dispList.append("getDefault's toString is: " + "\"" +
        Locale.getDefault().toString()+ "\",null);
    } else if (c == backCommand){
        dispList.deleteAll();
        display.setCurrent(mainList);
    } else if (c == exitCommand){
        destroyApp(false);
        notifyDestroyed();
    }
}
}

```

3.2.6.7.2 Code Examples for ResourceBundle

```

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.Enumeration;
import com.motorola.iden.resourcebundle.ResourceBundle;
import com.motorola.iden.resourcebundle.Locale;
import com.motorola.iden.resourcebundle.MissingResourceException;

public class DemoRB extends MIDlet
    implements CommandListener{
    private Locale myLocale1 = new Locale("zh","CN");

    private String resBaseName1;
    private ResourceBundle myBundle1_1;

```



```
private Enumeration myEnum1_1;

private Command getStringCommand, getKeysCommand, exitCommand,
backCommand;

private Display display;

private List mainList, dispList;

public DemoRB(){

    getStringCommand = new Command("getString",
Command.SCREEN,1);

    getKeysCommand = new Command("getKeys", Command.SCREEN,1);
    exitCommand = new Command("Exit", Command.EXIT, 1);
    backCommand = new Command("Back", Command.BACK, 1);
    display = Display.getDisplay(this);
    mainList = new List("Resource Bundle Demo", List.IMPLICIT);
    dispList = new List("Results", List.IMPLICIT);
    mainList.addCommand(getStringCommand);
    mainList.addCommand(getKeysCommand);
    mainList.addCommand(exitCommand);
    dispList.addCommand(backCommand);
    mainList.setCommandListener(this);
    dispList.setCommandListener(this);
    display.setCurrent(dispList);
}

public void startApp(){
    dispList.append("Constructing bundles for \"res1\"",null);
    resBaseName1 = "res1";
    try {
        myBundle1_1 = ResourceBundle.getBundle(resBaseName1,
new Locale(""));
        dispList.append("bundles for \"res1\"_\""+\"\"+\"\": \" +
myBundle1_1.getLocale().toString(),null);
    } catch(MissingResourceException e){
        dispList.append("bundles for \"res1\"_\""+\"\"+\"\":
MissingResourceException",null);
    }
}
```



```

    }

    public void pauseApp(){
    }

    public void destroyApp(Boolean unconditional){
    }

    public void commandAction(Command c, Displayable s){
        if (c == getStringCommand){
            String temp;
            dispList.append(myBundle1_1.toString(),null);
            myEnum1_1 = myBundle1_1.getKeys();
            while (myEnum1_1.hasMoreElements()){
                temp = (String)myEnum1_1.nextElement();
                dispList.append("key: "+ temp +", res:
"+myBundle1_1.getString(temp),null);
            }
            dispList.append("-----",null);
            display.setCurrent(dispList);
        } else if (c == getKeysCommand){
            dispList.append(myBundle1_1.toString()+" ' keys",null);
            myEnum1_1 = myBundle1_1.getKeys();
            while (myEnum1_1.hasMoreElements()){
                dispList.append(": "+
myEnum1_1.nextElement(),null);
            }
            display.setCurrent(dispList);
        } else if (c == exitCommand){
            destroyApp(false);
            notifyDestroyed();
        } else if (c == backCommand){
            dispList.deleteAll();
            display.setCurrent(mainList);
        }
    }
}
}

```



3.2.6.7.3 Code Examples for PropertyResourceBundle

```
property1 = you are choosing property key 1  
property2 = you are choosing property key 2  
property3 = you are choosing property key 3  
property4 = you are choosing property key 4  
property5 = you are choosing property key 5
```

3.2.6.7.4 Code Examples for ListResourceBundle

```
import com.motorola.iden.resourcebundle.ListResourceBundle;  
import com.motorola.iden.resourcebundle.ResourceBundle;  
  
public class DemoListRB extends ListResourceBundle{  
    protected Object[][] getContents() {  
        Object [][] objects = {  
            {"Listkey 1","you are choosing list key 1"},  
            {"Listkey 2","you are choosing list key 2"},  
            {"Listkey 3","you are choosing list key 3"},  
            {"Listkey 4","you are choosing list key 4"},  
  
            {"Listkey 5","you are choosing list key 5"}  
  
        };  
        return objects;  
    }  
}
```




3.2.6.7.5 Code Examples for SystemResourceBundle

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import com.motorola.iden.resourcebundle.Locale;
import com.motorola.iden.resourcebundle.SystemResourceBundle;
import com.motorola.iden.resourcebundle.MissingResourceException;

public class DemoSysRB extends MIDlet
    implements CommandListener{
    private SystemResourceBundle srb;
    private Locale locales [];

    private Command dispCommand, exitCommand, backCommand,
    setLocaleCommand;
    private Display display;
    private List mainList, dispList;

    public DemoSysRB(){
        dispCommand = new Command("getString", Command.SCREEN, 1);
        exitCommand = new Command("Exit", Command.EXIT, 1);
        backCommand = new Command("Back", Command.BACK, 1);
        setLocaleCommand = new Command("setLocale", Command.SCREEN,
1);

        display = Display.getDisplay(this);
        mainList = new List("Sys Res Bndl", List.IMPLICIT);
        dispList = new List("Result", List.IMPLICIT);
        mainList.addCommand(dispCommand);
        mainList.addCommand(exitCommand);
        dispList.addCommand(backCommand);
        dispList.addCommand(setLocaleCommand);
        mainList.setCommandListener(this);
        dispList.setCommandListener(this);
        display.setCurrent(dispList);
    }
}
```



```
public void startApp(){
    srb = new SystemResourceBundle();
}

public void pauseApp(){
}

public void destroyApp(Boolean unconditional){
}

public void commandAction(Command c, Displayable s){
    if (c == dispCommand){
        srb.setLocale(locales[mainList.getSelectedIndex()]);
        dispList.append("Current locale is: " +
srb.getLocale().toString(),null);
        dispList.append("-----",null);
        dispList.append("resID: STRING_COMM_YES" + "
resContent: "
+srb.getString(SystemResourceBundle.STRING_COMM_YES),null);
        display.setCurrent(dispList);
    } else if (c == exitCommand){
        destroyApp(false);
        notifyDestroyed();
    } else if (c == backCommand){
        dispList.deleteAll();
        display.setCurrent(mainList);
    } else if (c == setLocaleCommand){
        dispList.deleteAll();
        srb.setLocale(new Locale("fr"));
        dispList.append("The locale of setLocale \"fr\" is: "+
srb.getLocale().toString(),null);
    }
}
}
```



3.2.7 LicenseInfo API

3.2.7.1 Overview



This API is only available on this handset.

The LicenseInfo API lets J2ME™ MIDlets access license file information. This data may be used to track when the license file is going to cause the MIDlet to expire and hence gives MIDlets the flexibility to pop up any UI specific information. Other data that can be retrieved are type of license, license file version information, license file issuer information, license file target IMEI information, license file issue information, license file name information, license file vendor information, license file version information, license file upgrade TEXT information, license file upgrade URL information, license file upgrade parameter information, license file status information, license file metric information, license file gifting URL information, and license file tell-a-friend URL information.

3.2.7.2 Class Description

The LicenseInfo API is located in package `com.motorola.iden.licenseinfo`

```
java.lang.Object
|
+ -- com.motorola.iden.licenseinfo.LicenseInfo
```

3.2.7.3 Method Descriptions

3.2.7.3.1 LicenseInfo Methods

3.2.7.3.1.1 getLicenseType

```
public final static int getLicenseType()
```

Returns an integer that corresponds to the type of the license file.

Returned integer value is one of the values in the following table:

Returned Int value	Corresponding License Type
0	TYPE_DEMO
1	TYPE_PREPAID
2	TYPE_LIMITED
3	TYPE_UNLIMITED
4	TYPE_DURATION
5	TYPE_METERED

If there is no license file associated with the MIDlet suite, then the return value will be `TYPE_UNLIMITED`



3.2.7.3.1.2 **getLicenseMetric**

```
public final static long getLicenseMetric()
```

Returns a long. The interpretation of this value depends on the license type which can be determined using `getLicenseType()` API mentioned above. If the license type is demo (value of 0) it returns the total continuous execution time remaining for that particular MIDlet to expire. If the license type is prepaid (value of 1) it returns the total number of executions remaining. If the license type is limited (value of 2) it returns a date and time remaining. It returns a metric value in long that could be interpreted as date and time remaining for that particular MIDlet to expire. If the license type is unlimited (value of 3) it returns a zero. If the license type is duration (value of 4) it returns duration of time remaining for that particular MIDlet to expire. If the license type is metered (value of 5) it returns the total amount of time remaining for the app to expire. It is important to note that for types metered and demo time will be decreasing while the MIDlet is running, and for types duration, limited, time will be decreasing upon the installation of the MIDlet. If the type is prepaid the counter be reduced by one every time the MIDlet is exited.

3.2.7.3.1.3 **getField**

```
public final static String getField(int field)
```

```
throws IllegalArgumentException
```

Returns license file information such as the version number of the MIDlet, issuer name, target IMEI, information of time when the license was issued, name of the MIDlet, name of the vendor, MIDlet version number, upgrade text, upgrade URL, upgrade parameter, status text, license metric body of the app., gifting URL, and tell-a-friend URL.



field must be one of the values in this table:

fieldID	Example of Data
INFO_VERSION	"01.00.00"
INFO_ISSUER	"Motorola, Inc."
INFO_TARGETIMEI	"010101010101010"
INFO_ISSUED	"2001-12-17T09:30:47-05:00"
J2ME_NAME	"LicenseMIDlet"
J2ME_VENDOR	"John Smith"
J2ME_VERSION	"01.00.00"
J2ME_UPGRADE_TEXT	"Would you like to upgrade license for LicenseMIDlet?"
J2ME_UPGRADE_URL	"licenseupgradeurl.jsp"
J2ME_UPGRADE_PARAMETER	"promocode=2dfer23fsef"
J2ME_STATUS_TEXT	"LicenseMIDlet is currently enabled."
J2ME_METRIC	<pre> <credits></credits> <continuousexecutiontime> P0Y0M0DT00H10M </continuousexecutiontime> <expiration></expiration> <durationperiod>P0Y0M0DT00H2M</durationperiod> <meteredexecutiontime></meteredexecutiontime> <status>enabled</status> <statustext> LicenseMIDlet is currently enabled. </statustext> </pre>
J2ME_GIFTING_URL	"licensegiftingurl.jsp"
J2ME_TELLAFRIEND_URL	"licensetellafriendurl.jsp"

If the field value is out of the provided range of field values then an `IllegalArgumentException` will be thrown.



3.2.7.4 Code Examples

```
public void test(){

    // Get the license type method
    // 1) license type

    if(LicenseInfo.getLicenseType() ==
        LicenseInfo.TYPE_DEMO)
        screen.append("TYPE_DEMO->" + LicenseInfo.TYPE_DEMO);

    if(LicenseInfo.getLicenseType() ==
        LicenseInfo.TYPE_PREPAID)
        screen.append("TYPE_PREPAID->" +
LicenseInfo.TYPE_PREPAID);

    if(LicenseInfo.getLicenseType() ==
        LicenseInfo.TYPE_LIMITED)
        screen.append("TYPE_LIMITED->" +
        LicenseInfo.TYPE_LIMITED);

    if(LicenseInfo.getLicenseType() ==
        LicenseInfo.TYPE_UNLIMITED)
        screen.append("TYPE_UNLIMITED->" +
        LicenseInfo.TYPE_UNLIMITED);

    if(LicenseInfo.getLicenseType() ==
        LicenseInfo.TYPE_DURATION)
        screen.append("TYPE_DURATION->" +
        LicenseInfo.TYPE_DURATION);

    if(LicenseInfo.getLicenseType() ==
        LicenseInfo.TYPE_METERED)
        screen.append("TYPE_METERED->" +
        LicenseInfo.TYPE_METERED);

    // Get the metric value of the license type
    // 2) get license metric

        screen.append("METRIC_VALUE->" +
            LicenseInfo.getLicenseMetric());

    // Get different fields from license file
    // 3) version info.
    // 4) issuer info.
    // 5) target imei info.
    // 6) issued info.
    // 7) j2me name info.
    // 8) j2me vendor info.
```



```
// 9) j2me version info.
//10) j2me upgrade text info.
//11) j2me upgrade url info.
//12) j2me upgrade parameter
//12) j2me status text info.
//13) j2me metric info.
//14) j2me gifting url info.
//15) j2me tellafriend url info.

try
{
    screen.append("INFO_VERSION->" +
LicenseInfo.getField(LicenseInfo.INFO_VERSION));

    screen.append("INFO_ISSUER->" +
LicenseInfo.getField(LicenseInfo.INFO_ISSUER));

    screen.append("INFO_TARGETTIMEI->" +
LicenseInfo.getField(LicenseInfo.INFO_TARGETTIMEI));

    screen.append("INFO_ISSUED->" +
LicenseInfo.getField(LicenseInfo.INFO_ISSUED));

    screen.append("J2ME_NAME->" +
LicenseInfo.getField(LicenseInfo.J2ME_NAME));

    screen.append("J2ME_VENDOR->" +
LicenseInfo.getField(LicenseInfo.J2ME_VENDOR));

    screen.append("J2ME_VERSION->" +
LicenseInfo.getField(LicenseInfo.J2ME_VERSION));

    screen.append("J2ME_UPGRADE_TEXT->" +
LicenseInfo.getField(LicenseInfo.J2ME_UPGRADE_TEXT));

    screen.append("J@ME_UPGRADE_URL->" +
LicenseInfo.getField(LicenseInfo.J2ME_UPGRADE_URL));

    screen.append("J2ME_UPGRADE_PARAMETER->" +
LicenseInfo.getField(LicenseInfo.J2ME_UPGRADE_PARAMETER));

    screen.append("J2ME_STATUS_TEXT->" +
LicenseInfo.getField(LicenseInfo.J2ME_STATUS_TEXT));

    screen.append("J2ME_METRIC->" +
LicenseInfo.getField(LicenseInfo.J2ME_METRIC));

    screen.append("J2ME_GIFTING_URL->" +
LicenseInfo.getField(LicenseInfo.J2ME_GIFTING_URL));

    screen.append("J2ME_TELLAFRIEND_URL->" +
LicenseInfo.getField(LicenseInfo.J2ME_TELLAFRIEND_URL));
```



```
}catch(IllegalArgumentException e)
{
    System.out.println("Something went wrong!"
        + e.toString());
}
}
```




3.3 MIDlet Suite and MIDlet Icon Support

iDEN handsets support MIDlet icons at the suite and individual MIDlet level. Icons are specified by MIDlets through the standard MIDP JAD tags of MIDlet-Icon and the icon field of the tag MIDlet-*n* will be displayed. Because different handsets vary in screen size and available display mode resolutions iDEN handsets allow developers to specify multiple versions of their icons.

To ensure that a MIDlet's icons are always displayed regardless of display mode resolution, four derivations of the same graphic must be available in varying sizes, all in PNG format. The specific icon displayed to the end-user corresponds to the display mode resolution of the handset. Users can set the display mode via the handset's settings.




On some handsets strict restrictions are placed on the icon dimensions. If the width and height of a specified icon deviate from the requirements for a particular display mode it will not be rendered.




Other handsets are able to dynamically resize the specified icons to fit the current display mode. However, the icon may not appear as intended after it has been resized. These handsets still allow developers to specify icons tailor made to a specific resolution.

The following tables summarize the resolutions and the corresponding usage scenario:

Handsets	Display Mode	Icon Resolution	Notes
	Compressed	11x11	List mode with compressed font
	Standard	13x13	List mode with standard font
	Zoom	15x15	List mode with zoom font
	Iconic	18x18	Iconic menu mode



Handsets	Display Mode	Icon Resolution	Notes
	Compressed	13x13	List mode with compressed font
	Standard	15x15	List mode with standard font
	Zoom	17x17	List mode with zoom font
	Iconic	32x32	Iconic menu mode

Since the JAD tags specified by MIDP do not include a provision for icons of varying resolutions, an iDEN specific schema was created. The iDEN specific schema is designed to work in conjunction with the standards based approach while providing enough flexibility and simplicity for multi-resolution support. This schema is file name based requiring developers to simply prepend a size specific string to the beginning of the icon's file name. The default display mode is iconic. The icon specified by the MIDlet-Icon JAD tag or the icon field of the MIDlet-*n* JAD tag is used when the display mode is set to iconic. Icons for the other display modes can be provided using files with the same base name prepended by an additional 2 character specifier. The following table lists the display modes and their corresponding icon file name specifier strings.



Icon Mode	Naming Convention
Iconic	[name of icon].png
Compressed	c-[name of icon].png
Standard	s-[name of icon].png
Zoom	z-[name of icon].png
Notes – Total file name length is 32 ASCII characters, including the pre- and post-fix. The prefix is case sensitive.	

The following example demonstrates the proper formatting of the image names under both MIDlet suite and MIDlet use cases.

Icon Mode	Example 1	Example 2
Iconic	Games.png	Bounce.png
Compressed	c-Games.png	c-Bounce.png
Standard	s-Games.png	s-Bounce.png
Zoom	z-Games.png	z-Bounce.png



The placement of the various icons must be in the root directory of the JAR file. If any of the icons are missing or misplaced within the JAR, a default system icon will be utilized as a placeholder. Additional restrictions on the type of PNG file supported should be followed. The table below summarizes PNG support for MIDlet icons.

Handsets	4-bit PNG	8-bit PNG Without Transparency	8-bit PNG With Transparency	24-bit PNG Without Transparency	24-bit PNG Without Transparency
	Yes	White pixels become transparent.	Not supported. The default MIDlet icon provided by the handset will be rendered.	Not supported. The default MIDlet icon provided by the handset will be rendered.	Not supported. The default MIDlet icon provided by the handset will be rendered.
	Yes	White pixels become transparent.	Transparency is not honored. The transparent color will be rendered as black. Other icon corruption may occur.	White pixels become transparent	Transparent color is honored.

3.3.1 Tips

- When using white pixels for transparency, those pixels should be fully white (i.e. 0xFFFFFFFF).
- In the Java apps screen, the icon specified by the tag MIDlet-Icon and the name specified by MIDlet-name are displayed when there is only one MIDlet in the suite.
- Iconic display mode is not used when MIDlets in a MIDlet suite are being listed.



3.4 CLDC 1.1

3.4.1 Overview



CLDC 1.1 is only available on these handsets.

CLDC 1.1 is a revised version of the CLDC 1.0 specification including enhancements to existing features and new features such as floating point and weak reference support. CLDC Specification version 1.1 is an incremental release that is backwards compatible with CLDC Specification version 1.0.

The list below summarizes the main differences between CLDC Specification versions 1.1 (JSR 139) and 1.0 (JSR 30):

- Floating point support, including all J2SE floating point byte codes.
- Float and Double classes have been added.
- Various methods have been added to the other library classes to handle floating point values.
- Weak reference support (small subset of the J2SE weak reference classes).
- Classes Calendar, Date and TimeZone have been redesigned to be closer to J2SE.
- Error handling requirements have been clarified, as well as the addition of the `NoClassDefFoundError` class.
- Thread objects now have names similar to those in J2SE. The method `Thread.getName()` has been introduced, and the Thread class has a few new inherited constructors from J2SE.
- Clearer and detailed verifier specification.
- Various bug fixes and minor library changes, such as the addition of the following fields and methods:
 - `Boolean.TRUE` and `Boolean.FALSE`
 - `Date.toString()`
 - `Random.nextInt(int n)`
 - `String.intern()`
 - `String.equalsIgnoreCase()`
 - `Thread.interrupt()`

For detailed API information about the CLDC1.1 library, refer to <http://www.jcp.org/aboutJava/communityprocess/final/jsr139/index.html>.



4

Multimedia and Graphics

4.1 Overview

This section will cover the following multimedia and graphics topics:

- MIDP 2.0 LCDUI
- External Display
- Keycode Remapping
- Look and Feel (LnF)
- Smart Text
- Lightweight Windows Toolkit (LWT)
- Graphics Acceleration
- Micro3D
- Mobile 3D Graphics
- Multimedia
- Video Playback
- Lighting
- Vibrator
- Java Image Utility



4.2 MIDP 2.0 LCDUI

4.2.1 Overview

With the changes in the keypad layout and the new MIDP 2.0 UI specification, developers must consider a few implementation specifics that may affect the look and feel of the applications. Although every effort has been made to support the backward compatibility of the applications, the numerous hardware and software specification required some changes in convention from previous handsets. The next few sections outline the implementation specifics that affect application layout and usability.

4.2.2 Commands

4.2.2.1 Softkey Layout Priorities

In the LCDUI specification, applications that contain multiple soft keys have the option of specifying priority in layout. Soft key priority is as follows: left soft key, right soft key, and submenu. With only two dedicated keys, additional soft keys are added to a submenu, but are now accessed with the Menu key. If multiple commands contain the same priority level, the keys are assigned to the dedicated soft keys in the order they are added in left soft key, right soft key, sub-menu order.

4.2.2.2 Empty String Labels

When commands have non-empty strings an outline of the soft key area is rendered along with the text. This places hard boundaries to the areas rendered by the platform. If a command is created with an empty string (""), the command is not rendered but still occupies the soft key location. Additionally, if the soft key is activated by the user, the `commandAction()` method is still called. This is useful for placing commands in explicit locations. If a command is created with a short string (" "), it will be rendered on the display with no visible font.

4.2.2.3 Short and Long Label Usage

A major shortcoming of MIDP 1.0 was that it prevented commands from specifying varying lengths of the labels. This resulted in command text being truncated depending on where it was rendered, i.e. the soft key area or the command menu area. The MIDP 2.0 specification allows a command to have different length labels which alleviates this issue while remaining hardware independent and backwards-compatible.

The short strings are used if the command is placed on a dedicated soft key. In the command menu, long strings are rendered if they have been specified. If no long string is specified, the short string is used instead.

Note – Even with the new short/long string feature, instances where truncation is necessary will arise. The handset will only display full characters (the trailing characters are truncated) with labels justified according to the language.



4.2.3 Canvas

4.2.3.1 Size Changes

A major shortcoming of the MIDP 1.0 Canvas specification was the inconsistency in which command/canvas interactions were treated. Most implementations either reserved space for the commands or gave the applications full screen access. This inconsistency resulted in commands overwriting application screen real estate. With MIDP 2.0, Canvas-based applications are now able to accommodate for screen size changes, regardless of the implementation.

Any platform components that occupy real estate and are added to the canvas results in a call to the `sizeChanged()` method, followed by a paint. Beyond adding and removing commands, addition or removal of tickers or titles invokes a call to `sizeChanged()`. Within the `sizeChanged()` callback, applications should query the new canvas size and adjust rendering accordingly. The following table summarized the conditions that trigger the callback:

Conditions	Notes
Adding/Removing Commands to Canvas	Only first command added triggers <code>sizeChanged()</code> . Only last command removed triggers <code>sizeChanged()</code> .
Adding/Removing Ticker to Canvas	
Adding/Removing Title to Canvas	
Call to <code>setFullScreenMode()</code>	
Change in font size*	Changes in ergo settings menu applies if <code>com.motorola.iden.lnf</code> package is utilized to <code>getDefaultFont()</code> .



4.2.4 List

4.2.4.1 OK Key



The physical OK key is mapped to the default `SELECT_COMMAND` in `javax.microedition.lcdui.List`. Applications utilizing `List` should listen for the command in `commandAction()`. For `javax.microedition.lcdui.ChoiceGroup`, the physical OK key operates as the select key for `EXCLUSIVE`, `MULTIPLE`, and `POPUP` types.

On some handsets, there is no physical OK key. On these handsets the `SEND` key is mapped to the default `SELECT_COMMAND` in `javax.microedition.lcdui.List`. For `javax.microedition.lcdui.ChoiceGroup`, the `Send` key operates as the select key for `EXCLUSIVE`, `MULTIPLE`, and `POPUP` types.

4.2.4.2 Fit Policy

In MIDP 2.0, the application is given an option to specify `Choice` fit policy within a `List`. The functionality is optional per spec and is present only to provide hints to the platform as to the desired layout. The application cannot rely on the availability of the fit policy. iDEN handsets do not support this functionality, but the API does exist for compatibility's sake.

Note – The default fit policy is wrapping.

4.2.5 Forms

4.2.5.1 Item Layout

Considerable layout directives changes for `Items` have been incorporated in MIDP 2.0. In MIDP 1.0, layout directives applied strictly to `ImageItem` within `Forms`. Examples of such directives include `LAYOUT_NEWLINE_BEFORE`, `LAYOUT_CENTER`, and `LAYOUT_DEFAULT`. These directives provide layout hints to the platform as to how `ImageItem` are arranged within a `Form`. For MIDP 2.0, the scope of layout directives has been broadened to incorporate `StringItem`, `CustomItem`, and `Spacer` as well as `ImageItem`. In addition to expanding the reach of this functionality, additional layout directives have been added to increase the flexibility and usefulness.

To this end, layout directives have been moved from `ImageItem`, up to the superclass `Item`. Any subclass of `Item` is now capable of storing its own layout directive. In addition to broadening the reach, additional directives such as `LAYOUT_LEFT`, `LAYOUT_RIGHT`, and `LAYOUT_TOP` are included. By default, if no directive is specified, new `Items` added to the `Form` inherit the layout directives of the previous `Item`. By default, if `Items` do not specify a layout directive, they are added row by row, from left to right.



Note - StringItems are also appended in this same manner, differing from MIDP 1.0. For more information regarding layout changes, please refer to JSR-118. The following table outlines the behavior of Items that do not follow the standard layout directives:

Item	Default Behavior
Gauge	Center Justified.
ChoiceGroup	Language Dependent – default left justified.
DateField	Language Dependent – default left justified.
TextField	Screen width – default caret position is language dependent.

4.2.6 Item Commands

Utilizing a platform that resides on devices with very limited input and output mechanisms poses many challenges for application developers. In providing a myriad of options to the user, applications must present the user with a simple UI structure while still providing all the functionalities advanced users require. With the previous implementation of LCDUI, commands are constrained to Forms exclusively. That is, commands can only be added to Forms. While this is sufficient for simple applications, the sophistication and complexity of today's applications are quickly outgrowing this model. For MIDP 2.0, commands reside not only in Forms, but may also be added to individual Items.

Commands may be added to individual Items instead of exclusively to the Displayable. The availability of these commands is conveyed to the user when the Item is highlighted. If a command is available to a highlighted item, the Menu icon is displayed. Pressing the Menu key brings the command submenu to the foreground. If no commands are available to a highlighted item, the Menu icon may not be shown (see following table). If, however, the Form contains more than 2 commands, the Menu icon will always be highlighted. The following table summarizes the characteristics:

Condition	Menu Icon	Soft keys
0 Displayable Commands 1 Item Commands	Yes	None
2 Displayable Commands 0 Item Commands	No	Displayable Commands
2 Displayable Commands 1 Item Command	Yes	Displayable Commands
Note – If the Displayable contains more than 2 commands, the MENU icon will always be displayed. There is no means for the end-user to determine if Items have commands. Avoid adding more than 2 commands to the Displayable.		

4.2.7 TextBox/TextField

4.2.7.1 Constraints and Initial Input Modes

Text entry using the standard ITU keypad is not only cumbersome, but error prone. To alleviate this, many manufacturers have incorporated predictive text entry of one kind or another to ease the burden. The group for JSR 118 has taken this into consideration and incorporated multiple input modes within the high-level LCDUI components. First and foremost, the value constraints from MIDP 1.0 remain, including `ANY`, `PHONENUMBER`, and `URL`. In addition, new modifier flag constraints have been added to let the developer control the smart text engine. iDEN handsets honor all of the required and optional modifier flag



constraints: PASSWORD, UNEDITABLE, SENSITIVE, NON_PREDICTIVE, INITIAL_CAPS_WORD, and INITIAL_CAPS_SENTENCE.

MIDP 2.0 also introduces the concept of input modes for even finer control of text components. The input mode is simply a request for a specific set of characters to be entered more conveniently. Since not all devices and platforms will support all modes, no specific input modes are required by the MIDP 2.0 specification. iDEN handsets support several of the suggested input modes as well as some platform specific additions. The following table lists the supported input modes:

Constraint	Description
MIDP_UPPERCASE_LATIN	Defined by MIDP, this input mode turns on caps lock and switches the text component to English if the text component is currently in a non-Latin language.
MIDP_LOWERCASE_LATIN	Defined by MIDP, this input mode turns off caps lock or character shifting and switches the text component to English if the text component is currently in a non-Latin language.
IS_LATIN_DIGITS	Defined by J2SE™, this input mode switches the text component to NUMERIC mode if necessary.
UCB_BASIC_LATIN	Defined by J2SE™, this Unicode character block subset input mode switches the text component to English if the text component is currently in a non-Latin language. This input mode is equivalent to UCB_LATIN-1 SUPPLEMENT.
UCB_LATIN-1_SUPPLEMENT	Defined by J2SE™, this Unicode character block subset input mode switches the text component to English if the text component is currently in a non-Latin language. This input mode is equivalent to UCB_BASIC_LATIN.
UCB_HEBREW	Defined by J2SE™, this Unicode character block subset input mode switches the text component to Hebrew language mode if necessary and if the phone is configured for Hebrew support.
UCB_HANGUL_SYLLABLES	Defined by J2SE™, this Unicode character block subset input mode switches the text component to Korean language mode if necessary and if the phone is configured for Korean support.
X_MOTOROLA_IDEN_ENGLISH	Defined specifically for iDEN handsets, this input mode ensures that the text component is in English language mode.
X_MOTOROLA_IDEN_SPANISH	Defined specifically for iDEN handsets, this input mode ensures that the text component is in Spanish language mode.
X_MOTOROLA_IDEN_FRENCH	Defined specifically for iDEN handsets, this input mode ensures that the text component is in French language mode.
X_MOTOROLA_IDEN_PORTUGUESE	Defined specifically for iDEN handsets, this input mode ensures that the text component is in Portuguese language mode.

While the Motorola-defined input modes do allow developers to change the language setting for a particular text component, it is important to note that text component language is



automatically selected depending on the phone's language setting. Manually forcing a specific language should be used with caution as it can create a bad user experience. In addition, well-written applications should store and reset user specified input modes and constraints between sessions.

The Command changes listed above, along with the native UI have also changed the way text entry notification is displayed for MIDlets. When a MIDlet's screen is focused on a TextField or a TextBox, the current entry mode (Alpha, Word, or Numeric) is displayed in the bottom middle of the screen (between the Command labels, if present). This icon not only indicates entry mode but shift state. However, if the screen has more than two Commands associated with it, or if the TextField has an ItemCommand, the entry mode icon will be replaced by the Menu icon. In this case a user will not be able to easily tell what editing mode or shift state they are in and will be required to cycle through Command menus before being able to change the entry mode. If possible, developers are encouraged to avoid using ItemCommand with TextField and to keep TextFields on screens with two or less Commands.



4.3 External Display

4.3.1 Overview



i730 only

The ExternalDisplay API lets a MIDlet render to the external display of a handset with a flip. The API allows for free-formed rendering using a subclass of Canvas. There are 3 ways an application can gain focus to the external display.

Firstly, the application can explicitly make a request for the external display with `ExternalDisplay.requestDisplay()`. This request is granted if the flip is closed and the phone is at default ready screen (idle mode). If these conditions are true, the Application Management System (AMS) calls the MIDlet's `showNotify()` method. Otherwise, the AMS does nothing.

Secondly, the application can generate an alert to display on the external display by calling `ExternalDisplay.setCurrent()` with an Alert and the Displayable to show if the user chooses to view the content. The user can agree to see the content by pressing the Hi-Low Audio key or ignore it by pressing the Smart key. If the user agrees, the AMS calls the MIDlet's `showNotify()` method. Otherwise, the AMS does nothing.

Thirdly, the user can explicitly bring an application that supports the external display support onto the external display by scanning through a list of currently running applications. To do this, the user repeatedly presses the Smart key. The most recent call is displayed followed the name of the first application that supports the external display. If the user hits the Hi-Low Audio key then the application's `showNotify()` method is called and it can begin rendering to the Display. If the user hits the Smart key again, then the next application that supports the external display is shown. After the user reaches the end of the list, an idle screen is displayed.

When an application can display on the external display, its `showNotify()` method is called. When an application can no longer display on the external display, its `hideNotify()` method is called. An application displaying on the external display can receive key presses from the following keys: Volume Up, Volume Down, PTT and the Speaker key. If the user presses the smart key while an application is on the external display, the application can no longer display there and its `hideNotify()` method is called.

When an application is rendering to the External Display, opening the flip will cause the application to Auto Resume onto the Internal Display.

4.3.2 Class Description

4.3.2.1 ExternalDisplay Description

The API for the ExternalDisplay is located in package `com.motorola.iden.lcdui`

```
java.lang.Object
|
+ - com.motorola.iden.lcdui.ExternalDisplay
```



4.3.2.2 ExternalDisplayCanvas Description

The API for the ExternalDisplayCanvas is located in package `com.motorola.iden.lcdui`

```
java.lang.Object
|
+--Canvas
|
+ - com.motorola.iden.lcdui.ExternalDisplayCanvas
```

The ExternalDisplayCanvas represents a Canvas that can be rendered onto the External Display. The ExternalDisplayCanvas is identical to a normal LCDUI Canvas with a few exceptions:

- ExternalDisplayCanvas does not support adding commands to the Canvas.
- When the Canvas is first created, its width and height are the same as any other Canvas on the phone. The height and width are changed to the sizes for the external display when its `sizeChanged()` method is called, which occurs before the canvas is rendered onto the external display.

4.3.3 Method Descriptions

4.3.3.1 ExternalDisplay Methods

4.3.3.1.1 getCurrent

Returns the Displayable from this MIDlet that will be rendered on the external display the next time this MIDlet is allowed to render on that display.

```
public Displayable getCurrent()
```

4.3.3.1.2 getDisplay

Returns an ExternalDisplay object for this MIDlet.

```
public static ExternalDisplay getDisplay(MIDlet m)
```

Successive calls to this method return the same ExternalDisplay object. An `IllegalStateException` shall be thrown if a request to `getDisplay` is made during the execution of the MIDlet class's constructor.

4.3.3.1.3 getFlipState

Returns true if the phone's flip is closed; returns false otherwise.

```
public Boolean getFlipState()
```

4.3.3.1.4 getHeight

Returns the height of the ExternalDisplay in pixels.

```
public int getHeight()
```



4.3.3.1.5 getWidth

Returns the width of the ExternalDisplay in pixels.

```
public int getWidth()
```

4.3.3.1.6 isColor

Returns true if the ExternalDisplay supports color; returns false otherwise.

```
public Boolean isColor()
```

4.3.3.1.7 numColors

Returns the total number of colors supported by the ExternalDisplay.

```
public int numColors()
```

4.3.3.1.8 releaseDisplay

Tells the AMS that this MIDlet no longer wishes to render anything on the ExternalDisplay.

```
public void releaseDisplay()
```

4.3.3.1.9 requestDisplay

Tells the AMS that the MIDlet wants to render on the ExternalDisplay.

```
public void requestDisplay()
```

The AMS lets the MIDlet render on the display if the phone's flip cover is closed and the MIDlet is in the paused state. A MIDlet knows it can render on the external display when `ExternalDisplayCanvas.showNotify()` is called.

4.3.3.1.10 setCurrent

Sets the Displayable that will be rendered on the ExternalDisplay the next time this MIDlet is allowed to render on that display.

```
public void setCurrent(Alert alert, Displayable nextDisplayable)
```

This method generates an Alert to notify the user and sets `nextDisplayable` to be the Displayable that is rendered on the ExternalDisplay the next time this MIDlet is allowed to render on that display.

```
public void setCurrent(Displayable nextDisplayable)
```

This method sets `nextDisplayable` to be the Displayable that's rendered on the ExternalDisplay the next time this MIDlet is allowed to render on that display.

4.3.3.1.11 callSerially



Synchronizes an action with other event calls.

```
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. 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()`.



4.3.4 Code Examples

```

public class ExtDispTest extends javax.microedition.midlet.MIDlet {
    ExternalDisplay ed;
    Display d;

    myCanvas dispCanvas;
    myCanvas extDispCanvas;
    String str1 = "String 1";
    String str2 = "String 1";
    boolean firstTime = true;

    public void startApp() {
        try {
            if(firstTime)
            {
                d = Display.getDisplay(this);
                ed = ExternalDisplay.getDisplay(this);
                dispCanvas = new myCanvas();
                d.setCurrent(dispCanvas);
                extDispCanvas = dispCanvas;
                ed.setCurrent(extDispCanvas);
                firstTime = false;
            }
        }
        catch(Throwable e)
        {
        }
    }

    public void pauseApp() {
        if(ed.getFlipState())
        {
            ed.requestDisplay();
            System.out.println("The Flip is closed!");
        }
        else
        {
            (new Thread(extDispCanvas)).start();
            System.out.println("The Flip is open!");
        }
    }

    public void destroyApp(boolean unconditional) {
    }
}

class myCanvas extends ExternalDisplayCanvas

```



```
{  
  
    public myCanvas()  
    {  
    }  
  
    protected void paint(Graphics g) {  
        g.setColor(0xffffffff);  
        g.fillRect(0,0,getWidth(),getHeight());  
        g.setColor(0x000000);  
        g.drawString("Hello World",0,0,g.TOP|g.LEFT);  
    }  
}
```

4.3.5 Tips /

- If you wish to have a text ticker on the external display then simply add a ticker onto the ExternalDisplayCanvas.
- Even though the Java application has entered the paused state while it is rendering to the external display, the application can play media items like MIDIs and WAVs.



4.4 Keycode Remapping



This feature is only available on these handsets.

MIDlets that need to use low-level key codes may not run on all handsets due to differing keycode values. Keycode remapping allows a MIDlet to specify what keycode values are desired for the actual keys on the handset as well as customization of game actions.

The 0 through 9 keys and the pound and star keys cannot be remapped since they have logical ASCII value mappings which are portable across most handsets. The end and power keys cannot be remapped as they send no key events to MIDlets.

All remapping is done through JAD tags unique to iDEN handsets. The following JAD tags allow key code values to be specified for the corresponding keys:

iDEN-MIDP-KEY-SELECT
iDEN-MIDP-KEY-SOFT-LEFT
iDEN-MIDP-KEY-MENU
iDEN-MIDP-KEY-SOFT-RIGHT
iDEN-MIDP-KEY-LEFT
iDEN-MIDP-KEY-RIGHT
iDEN-MIDP-KEY-UP
iDEN-MIDP-KEY-DOWN
iDEN-MIDP-KEY-AUDIO
iDEN-MIDP-KEY-VOL-DOWN
iDEN-MIDP-KEY-VOL-UP
iDEN-MIDP-KEY-OK
iDEN-MIDP-KEY-SMART
iDEN-MIDP-KEY-PTT

Additionally, developers can customize which keys correspond to the MIDP defined game actions. The following JAD tags allow game actions to be remapped:

iDEN-MIDP-KEY-GAME-A
iDEN-MIDP-KEY-GAME-B
iDEN-MIDP-KEY-GAME-C
iDEN-MIDP-KEY-GAME-D
iDEN-MIDP-KEY-GAME-UP
iDEN-MIDP-KEY-GAME-DOWN
iDEN-MIDP-KEY-GAME-LEFT
iDEN-MIDP-KEY-GAME-RIGHT
iDEN-MIDP-KEY-GAME-FIRE

Using the first set of JAD tags, any numeric value can be assigned to any one of the keys listed. In other words, the key code returned to the MIDlet for the key specified in the JAD file tag will be the one specified by the developer. For example if a MIDlet's JAD file has the line below:



```
iDEN-MIDP-KEY-PTT: 12
```

Then calls to the `keyPressed`, `keyReleased` and `KeyRepeated` methods would be made with 12 passed as the `keycode` parameter value instead of the default of -50.

Furthermore, if a key has its `keycode` remapped it will still retain its game action, if any. For example, if the up arrow's `keycode` is reassigned to 2 a call to `getGameAction(2)` would still return the game action `Canvas.UP`.

Using the second set of JAD tags, any numeric value can be associated with any of the game actions listed. In other words, the value associated with a game action will cause calls to `Canvas.getGameAction(int keycode)` to return the associated game action when the specified value is passed as a parameter. For example if a MIDlet's JAD file has the line below:

```
iDEN-MIDP-KEY-GAME-A: -50
```

Then calls to `Canvas.getGameAction(-50)` will return `Canvas.UP` instead of 0 by default.

Note that it is possible to make game actions unreturnable by associating them with `keycodes` that will never be returned.

The following JAD snippet contains mappings to non-iDEN Motorola handsets:

```
iDEN-MIDP-KEY-SELECT: -10
iDEN-MIDP-KEY-SOFT-LEFT: 21
iDEN-MIDP-KEY-MENU: 23
iDEN-MIDP-KEY-SOFT-RIGHT: 22
iDEN-MIDP-KEY-LEFT: 2
iDEN-MIDP-KEY-RIGHT: 5
iDEN-MIDP-KEY-UP: 1
iDEN-MIDP-KEY-DOWN: 6
iDEN-MIDP-KEY-AUDIO: -53
iDEN-MIDP-KEY-VOL-DOWN: 302276412
iDEN-MIDP-KEY-VOL-UP: 302276412
iDEN-MIDP-KEY-OK: 20
iDEN-MIDP-KEY-SMART: -54
iDEN-MIDP-KEY-GAME-FIRE: 20
```

The following snippet contains mappings to some non-Motorola handsets:

```
iDEN-MIDP-KEY-SELECT: -10
iDEN-MIDP-KEY-SOFT-LEFT: -6
iDEN-MIDP-KEY-MENU: 23
iDEN-MIDP-KEY-SOFT-RIGHT: -7
iDEN-MIDP-KEY-LEFT: -3
iDEN-MIDP-KEY-RIGHT: -4
iDEN-MIDP-KEY-UP: -1
```

iDEN J2ME™ Developer's Guide



iDEN-MIDP-KEY-DOWN: -2

iDEN-MIDP-KEY-OK: -10

iDEN-MIDP-KEY-GAME-FIRE: -10



4.5 Look and Feel (LnF)

4.5.1 Overview

The main purpose of the LnF API is to provide facilities to modify the graphical user interface and interface related behavior of J2ME™ components without breaking backward compatibility and without modifying the standardized J2ME™ APIs. The LnF allows developers to modify how the standard user interface (UI) components available in J2ME™ (javax.microedition.lcdui and com.motorola.lwt) look and respond to certain user interactions without modifying the base code of every component. The LnF also provides an API to allow developers to plug in different LnFs (styles) without compromising the standardized API provided by MIDP 1.0+ and LWT 1.0+. The LnF allows developers to modify font settings, color settings, border settings, and icon sets of UI components.

The LnF API allows specifying the style used by a particular family of UI components. Style refers to the border that surrounds the component, the color scheme and the font settings used by the component; in other words, the look-and-feel. A family of UI components refers to all components of the same Java class, including any that may have been instantiated already.

In addition to styles, LnF 2.0 provides a much more flexible and richer functionality. It allows developers to modify the geometry of the UI components by allowing them to overwrite the paint functionality used by the UI components. That means that the developer is now capable of implementing the low-level graphics code (refer to MIDP Canvas and Graphics) that will be invoked by the component when it requires the rendering of its contents. In order to achieve this capability, the developer must comply fully with the framework. That means that the developer must correctly overwrite all methods required by the LnF framework.

The main two functionalities that a developer must overwrite are the following: the low-level paint functionality and the preferred dimension request functionality. The LCDUI/LnF framework will first query the overwritten version of the LnF what the preferred dimensions of a particular component will be. Every UI component should have the appropriate rectangular space to render fully its own contents. In order for developers to calculate these preferred dimensions accurately, the framework passes references to a component's content. Once the preferred dimensions are obtained, the framework will proceed with the layout calculation. The layout calculation (refer to MIDP 2.0 Form/Item section and iDEN MIDP 2.0 layout section) will use the preferred dimensions to allocate the correct space for every UI component and determine its location within the screen space. Once the calculation is completed, the paint cycle of the Screen will invoke the overwritten paint method of every component while passing all the applicable information regarding the state and contents of the component in turn.

The LCDUI/LnF framework is a composition of UI components and their corresponding LnF classes. The UI component classes (i.e. any Item subclass such as ImageItem, Gauge, etc.) communicate through a defined API with their corresponding LnF class. The API uses 2 basic parameters: an array of Objects (`Object[]`) to pass in the contents of the Item (i.e. the Image and label of an ImageItem) and a single integer to specify the mode. The mode is just an arrangement of bits, which specify the state of the component such as highlighted (focused), and its mode (radio-button, checkbox, etc.).

Depending on the nature of the Item, the array of Objects will contain different Objects. The implementations of LnF classes must typecast every Object within the array to its appropriate



class in order to use it. Also, depending on the nature of the Item, the mode will have to be “decoded” using the appropriate masks to determine the state: highlighted, selected, etc.

4.5.2 Class Description

The API for the LnF is located in package `com.motorola.iden.lnf`.

```
com.motorola.iden.lnf.LookAndFeel
|
+- com.motorola.iden.lnf.DisplayableLookAndFeel
| |
| +- com.motorola.iden.lnf.SystemLookAndFeel
|
+- com.motorola.iden.lnf.WidgetLookAndFeel
|
+- com.motorola.iden.lnf.ItemLookAndFeel
| |
| +- com.motorola.iden.lnf.CheckboxItemLookAndFeel
| +- com.motorola.iden.lnf.ChoiceGroupItemLookAndFeel
| +- com.motorola.iden.lnf.DateFieldItemLookAndFeel
| +- com.motorola.iden.lnf.GaugeItemLookAndFeel
| +- com.motorola.iden.lnf.ImageItemLookAndFeel
| +- com.motorola.iden.lnf.StringItemLookAndFeel
| +- com.motorola.iden.lnf.TextFieldItemLookAndFeel
|
+- com.motorola.iden.lnf.CommandAreaLookAndFeel
+- com.motorola.iden.lnf.ScrollbarLookAndFeel
+- com.motorola.iden.lnf.TickerLookAndFeel
+- com.motorola.iden.lnf.TitleLookAndFeel

com.motorola.iden.lnf.LookAndFeelEngine
```

The most important class is the `LookAndFeelEngine`. The `LookAndFeelEngine` allows developers to control the LnF by allowing developers to replace existing `LookAndFeel` implementations for customized ones. The `LookAndFeelEngine` also provides an API to get references to the existent LnF classes so its inner attributes such as `Border`, `Font`, `ColorPalette`, and `JustificationStyle` can be modified as well.

4.5.3 Code Examples

The following is a set of examples to modify the LnF. These examples are organized as follows:

Replacing an existing `LookAndFeel` with a different customized version.

Creating a new `CheckboxItemLookAndFeel`.

1. Example 1 describes how to replace an existing `LookAndFeel` with a different one. Developers must create a valid LnF class that extends the original one used by the framework and overwrite the methods according to his/her needs.
2. Example 2 illustrates the creation of a new `LookAndFeel` that inherits the functionality of the original one used by the LnF framework.



4.5.3.1 Example 1.

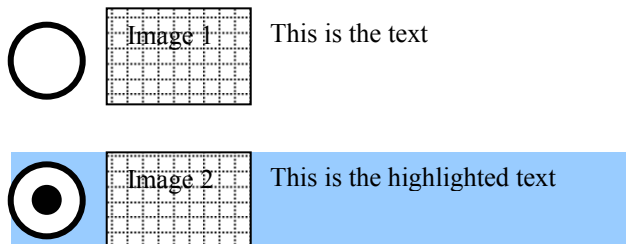
```
try
{
    // instantiate the new LookAndFeel to be used by all
    // List and ChoiceGroup options
    MyCheckboxLnF cbLnF = new MyCheckboxLnF();

    // replace the existing LnF with the new one
    LookAndFeelEngine.set(LookAndFeelEngine.LNF_ID_LCDUI_CHECKBOXITEM,
        cbLnF);

    // since we just modified the LnF used by *ALL* Lists and
    // ChoiceGroups, we must make all the Screens of these types
    // invalid so layout calculation happens and the new LnF gets used
    LookAndFeelEngine.markAsValid(myList, false);
    LookAndFeelEngine.markAsValid(myForm, false);
}
catch (LookAndFeelException lnfe)
{
    // a problem was encountered
}
```

4.5.3.2 Example 2.

The following example modifies the geometry of the options of a ChoiceGroup or List. The standard implementation produces the following radio-buttons:



This example will produce the following radio-buttons:

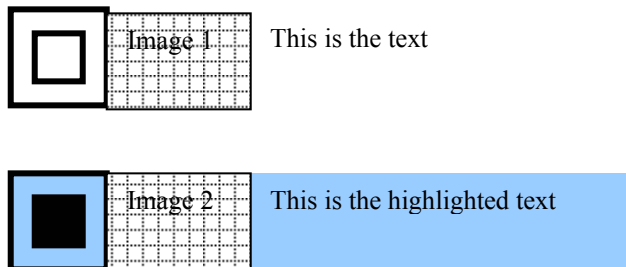


Figure 4.1



```
public class MyCheckboxLnF extends CheckboxItemLookAndFeel
{
    // the default constructor
    public MyCheckboxLnF() throws LookAndFeelException
    {
    }

    // returns the width occupied by the image and
    // the radio-button. The width of the TextView
    // will be set by the framework once this method
    // returns
    public int getPreferredWidth(Object[] params,
                                int mode)
    {
        int w = 0;

        // retrieve the parameters
        TextView tv = (TextView)params[0];
        Image img = (Image)params[1];
        Font font = (Font)params[2];

        // let's use the font height as the dimension of the radio-button
        w += font.getHeight();

        // consider the width of the image
        w += img.getWidth();

        return w;
    }

    // returns the height
    public int getPreferredHeight(Object[] params,
                                  int mode)
    {
        int h;

        // retrieve the parameters
        TextView tv = (TextView)params[0];
        Image img = (Image)params[1];
        Font font = (Font)params[2];

        // the TextView should have been reformatted by the framework
        // already, so its height is accurate
        int tvh = tv.getHeight();

        // let's use the font height as the dimension of the radio-button
        int fh = font.getHeight();

        // consider which is higher: radio-button icon or text
        if (fh > tvh)
        {
            h = fh;
        }
    }
}
```



```
else
{
    h = tvh;
}

// consider the height of the image
int ih = img.getHeight();
if (ih > h)
{
    h = ih;
}

return h;
}

// paint the radio-button/checkbox, etc.
public void paint(Graphics g, Object[] params,
                  int width, int height, int mode)
{
    // don't need to clear the background, it was cleared by
    // the Displayable that contains this ChoiceGroup option

    // determine if the item is selected
    boolean selected = (mode & ITEM_LNF_STATE_SELECTED != 0);

    // determine if the item is highlighted (has focus)
    boolean focused = (mode & ITEM_LNF_STATE_HIGHLIGHTED != 0);

    // render the focused rectangle/highlighting
    if (focused)
    {
        // use the highlighting color
        g.setColor(getColor(ColorPalette.HIGHLIGHTED_FILL_COLOR));
        g.fillRect(0, 0, width, height);
    }

    // set the default foreground color
    g.setColor(getColor(ColorPalette.FOREGROUND_COLOR));

    // render the radio-button or checkbox (if applicable)
    int size = font.getHeight();
    switch (mode & ITEM_LNF_ASPECT_MASK)
    {
        case ITEM_LNF_ASPECT_RADIOBUTTON:
            // outer rectangle
            g.drawRect(0, 0, size-1, size-1);

            // inner rectangle, if selected
            g.drawRect(1, 1, size-3, size-3);
            if (selected)
            {
                g.fillRect(1, 1, size-2, size-2);
            }
        }
    }
}
```




```
    }
    // move the origin of coords so the image
    // or TextView is next to this icon
    g.translate(size, 0);
    break;

case ITEM_LNF_ASPECT_CHECKBOX:
    // outer rectangle
    g.drawRect(0, 0, size-1, size-1);

    // Inner mark, if selected
    // (the inner mark is an x)
    g.drawRect(1, 1, size-3, size-3);
    if (selected)
    {
        g.drawLine(0, 0, size-2, size-2);
        g.drawLine(0, size-2, size-2, 0);
    }
    // move the origin of coords so the image
    // or TextView is next to this icon
    g.translate(size, 0);
    break;
}

// render the image (if any)
if (img != null)
{
    g.drawImage(0, 0, img, 0);

    // move the origin of coords so the TextView
    // is next to this image
    g.translate(img.getWidth(), 0);
}

// render the TextView
tv.paint(g);
}
```



4.6 Smart Text Entry

4.6.1 Overview

Text components on iDEN handsets are enabled with T9 smart text entry capability. Smart text entry allows users to enter text faster by removing the need for multiple key presses in order to input certain letters or symbols. The T9 engine recognizes given key sequences and matches them with the words contained in its database. Because multiple words may be entered with a single key sequence, users can access other matches in the word database, or explicitly enter any word character by character.

LCDUI based MIDlets that conform to the MIDP specification and use TextField or TextBox objects and LWT based MIDlets that use TextField or TextArea objects automatically benefit without any additional coding effort.

The rest of this section explains various T9 features and how users interact with T9 in a MIDlet.

4.6.2 T9 Features

The T9 engine allows users to enter text in one of four different ways: Word entry mode, Alpha entry mode, Numeric entry mode, and Symbol entry mode. Additionally, the T9 engine supports multiple languages, which can be combined with Word entry mode to enter text in English, Spanish, French, Portuguese, Hebrew, or Korean.

Without some form of smart text entry a handset must rely on multi-tap text entry in text components. With multi-tap entry, a user is required to press a key multiple times to access one of the multiple alphabetic characters mapped to that key. For example, to enter the character 'c', the '2' key must be pressed 3 times.

Text components can still use this method of entry, which is also called Alpha mode. While Alpha mode is not the most efficient way to enter characters on the phone, it is sometimes necessary in order to enter words that are not in the T9 word database.

For words that are in the database, T9's Word mode is much more efficient. With Word mode, users are not required to explicitly enter each character for a word. For example, without Word mode the word "back" requires 8 key presses to enter- double the amount of letters. With Word mode, a user needs to press only 4 keys. The T9 engine recognizes the key sequence as the word "back" since that is the only possible match for the sequence. In cases where there is more than one match, the user can press the "0" key, also labeled "next", to match to the next word. If a desired word is not in T9's word database, users can switch to Alpha mode and enter the word explicitly.

In cases where only numeric characters need to be entered, T9's "Numeric" mode is used. This mode simply maps the keys pressed to their labeled numeric value.

While users can access symbols like punctuation through Alpha and Word modes, T9 also has a "Symbol" mode. The Symbol mode operates slightly differently than the other modes because symbol entry is done on a separate screen. When the user is done choosing all the symbols needed, those symbols are inserted into the text component at the current cursor position.



4.6.3 The T9 UI

A MIDlet's UI is automatically integrated with the T9 UI when a text component is present and has focus. While entering text into a text component, the T9 UI consists only of the icon representing the current entry mode, and any capitalization taking place and word highlighting.

In order to maximize the amount of screen space on the device, the space for the entry mode icon is now shared with the menu icon. If the MIDlet has more than two Screen Commands or if the TextField has an Item Command, the menu icon is displayed in place of the entry mode icon.



The screenshot on the left displays a typical email MIDlet. The focused text component is the Message box.

The text within the Message box is being edited in Word mode, indicated by the icon at the bottom center of the screen and the word "application" is the returned word based on the key sequence entered. The user has pressed the '#' key enabling character shift, indicated by the up arrow in the icon. The next character entered by the user will be returned as a capital letter.

Figure 4.2 A T9-Enhanced MIDlet

4.6.4 Changing T9 Entry Mode

Smart text entry in a MIDlet is nearly seamless, giving the MIDlet more functionality without complicating it. However, users can interact in a more direct manner with the T9 engine to switch entry modes and change language. Although a user's interaction with the T9 engine is completely transparent to the MIDlet, users are required to go through a MIDlet's UI in order to access the T9 UI.

In both LWT and LCDUI based MIDlets the T9 entry mode screen is accessed by pressing the Menu key. If a MIDlet has more than two Screen Commands or the TextField has an Item Command, the first Menu key press accesses the Command menu screen per the MIDP specification. If the MIDlet was previously focused on a TextField, the second Menu key press accesses the T9 "Entry Method" menu. For MIDlets with two or fewer Screen Commands and no Item Command to access, the Menu key automatically accesses the Entry Method menu when focused on a TextField.

The image to the right shows how the Entry Method menu appears in an LCDUI-based MIDlet. This menu allows the user to select the desired mode of text entry. Selecting the Alpha, Word, or Numeric modes returns the user to the last MIDlet screen and changes the entry mode of the focused text component. Selecting Language or Symbol brings the user to a different screen where they can select the desired language or symbols.

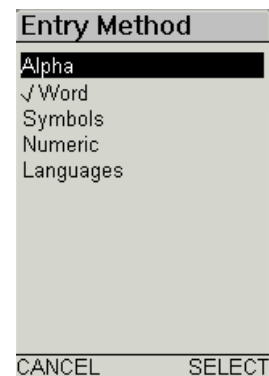


Figure 4.3 The Entry Methods



4.6.5 Influencing T9

Although there is no code change required to take advantage of T9 in an LCDUI or LWT based MIDlet, there are ways to influence T9 behavior.

The MIDP 2.0 specification has been updated to include APIs that allow developers to directly control the smart text engine for a `TextField`. These APIs are capable of controlling entry mode, capitalization, and language among other features. Refer to the MIDP 2.0 specification for more information on those APIs.

In addition to receiving direction with the MIDP 2.0 APIs, the smart text engine is affected by implicit settings on the phone. Just as a MIDlet can take advantage of the internationalization features of the phone by displaying the MIDlet name in different languages, the T9 engine detects the default language of the user and starts itself in the appropriate language setting.

The initial entry mode of the T9 engine can also be affected by the constraints of the text component that it services. For example, a `TextField` created with the `TextField.PHONENUMBER` constraint will create a `TextField` with a T9 engine set initially to Numeric entry mode (with some modifications that allow users to enter special dialing characters). In addition, the user will not be able to access the Entry Method menu to change the input mode. Similarly, a `TextField` created with the `TextField.NUMERIC` constraint will initially be set to Numeric entry mode, but will also allow for negative numbers. Text components whose constraints are set to `NUMERIC` or `PHONENUMBER` after instantiation will change their entry mode accordingly, but relaxing constraints will not change the entry mode of a text component.

4.6.6 T9 Engine Lifecycle

Text components each have their own instance of the T9 engine, allowing MIDlets to contain multiple text components at once, each performing different smart text functions and using different entry modes.

MIDlets that reuse text components on different screens typically clear out any old text before redisplaying that component to a user. While the MIDP API provides for such functionality, it does not provide for any functionality to reset the T9 engine of a text component. MIDlets that reuse text components will also carry the old T9 state of that text component when it is redisplayed. For example, if a user last left a `TextField` in Alpha entry mode and that `TextField` was removed from one `Screen` object and added to another, it will look as if it was initialized in Alpha mode in the new `Screen`. To avoid this behavior, text components should not be recycled.



4.7 Lightweight Window Toolkit (LWT)

4.7.1 Overview

The Lightweight Window Toolkit is an OEM extension that was designed to address the limitations of LCDUI imposed by the MIDP 1.0 specification. Since the release of MIDP 2.0, most of these limitations have been addressed. For example, LCDUI now provides developers with control over screen layouts and custom components.

While LWT was a key enabler for the development of full-featured applications on legacy devices, its functionality is superseded on MIDP 2.0 compatible devices. It is included on some handsets for backwards compatibility with pre-existing applications. The LWT package has not been updated to take advantage of new MIDP 2.0 capabilities. As such, features like TextField's ability to specify enhanced text entry modes are not available through LWT. In addition, some backwards compatibility issues with the MIDP 2.0 specification can cause undesirable behavior in LWT.

Developers are strongly encouraged to use standard MIDP 2.0 LCDUI classes for UI development in order to ensure compatibility across most devices; however we have provided legacy documentation on the LWT package for your reference.

4.7.2 Example: Hello LWT World

The following example illustrates the creation of the classical "Hello World" program using the LWT package.

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import com.motorola.lwt.*;
import java.io.IOException;

public class HelloLWTWorld extends MIDlet {
    Display display;
    ComponentScreen scr;
    ImageLabel label;

    public HelloLWTWorld() {
        // get display
        display = Display.getDisplay(this);

        // create the ComponentScreen
        scr = new ComponentScreen();

        // create ImageLabel
        label = new ImageLabel(null, null, "Hello LWT World!");

        // place the ImageLabel in the center of the screen
        label.setLeftEdge(Component.SCREEN_HCENTER,
            -label.getPreferredWidth()/2);
        label.setTopEdge(Component.SCREEN_TOP,
            (scr.getHeight()-label.getPreferredHeight())/2);
    }
}
```



```

        // add ImageLabel to the ComponentScreen
        scr.add(label);
    }

    public void startApp() {
        display.setCurrent(scr);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean b) {
    }
}

```

The main concept illustrated in the example is the creation of a container, the `ComponentScreen`, and the addition of a Component, the `ImageLabel`. This example will be revisited later to cover the details regarding the location and dimension of the Component.

4.7.3 Class Hierarchy and Overview

The following diagram shows the class hierarchy of LWT.

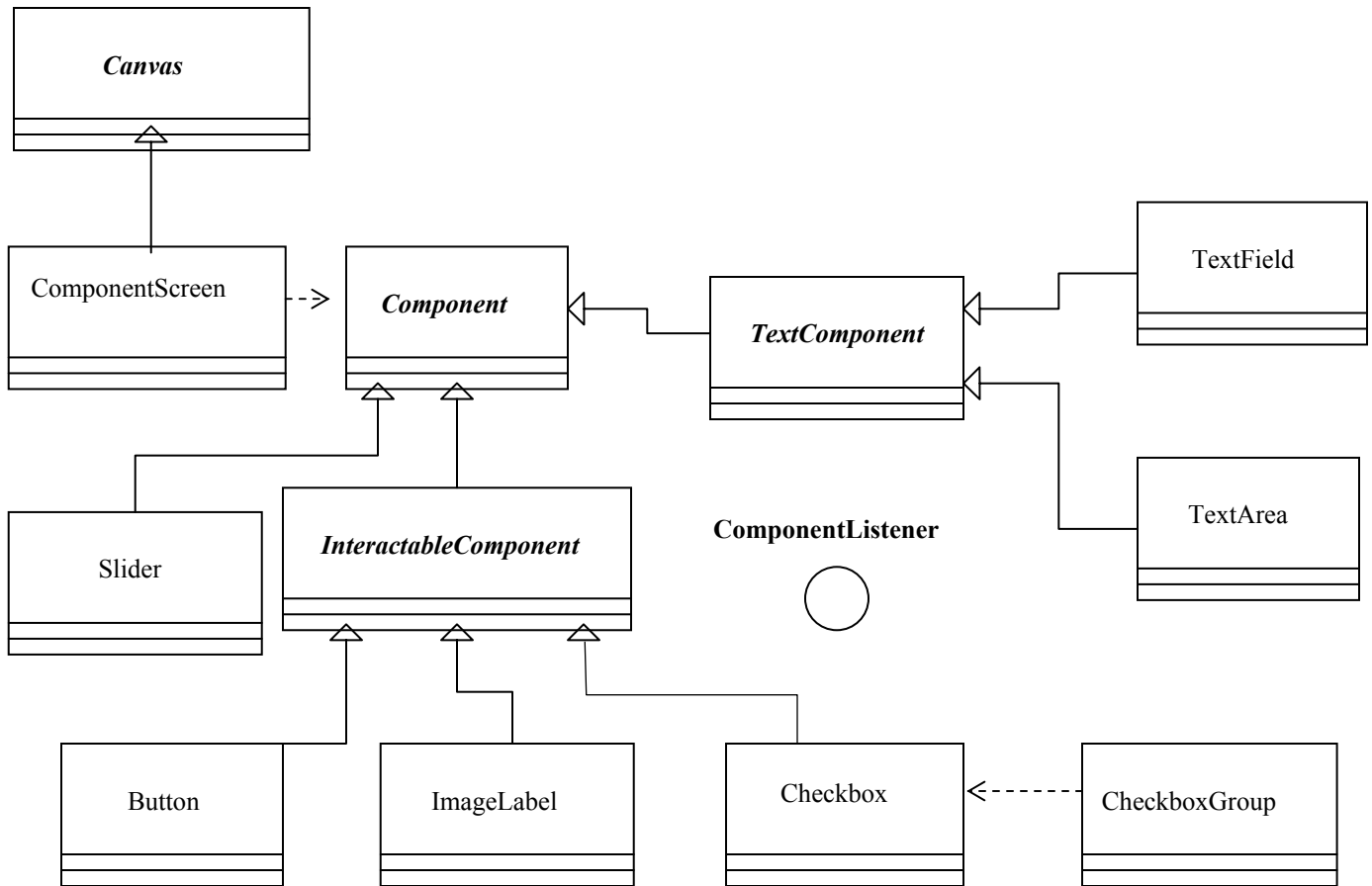


Figure 4.4 LWT Class Hierarchy



4.7.4 ComponentScreen

The ComponentScreen class provides the basic functionality to create application screens to interact with the user. It is the top-level container in an LWT user interface. ComponentScreen provides a variety of means to manage Components, layout, focus traversing, and scrolling.

4.7.5 Component

Component is the abstract base class of all LWT user interface entities that can be added to a ComponentScreen. It provides core functionality to allow rendering, layout management, input event handling and more.

4.7.6 ComponentListener

ComponentListener is an interface that can be implemented by any class that wishes to receive events from a Component. The events vary depending on the object that originates the event, but can include events such as button actuation, checkbox selection, etc..

4.7.7 InteractableComponent

The InteractableComponent is a subclass class of Component that can be actuated by the user; that is, it can be visually 'pressed' and 'released' by the user. The following are examples of InteractableComponents:

- Button
- Checkbox
- ImageLabel – A Component that can display an image and/or a text label.



Figure 4.5 LWT Sample Screen



4.7.8 The ComponentScreen Class

This section describes the fundamental behaviors exhibited by ComponentScreen. The ComponentScreen class provides the basic functionality to create application screens to interact with the user. It is the top-level container in an LWT user interface. As a subclass of LCDUI's Canvas, it can be interchanged with other LCDUI screens such as Canvas, Form, and Alert.

ComponentScreen inherits several methods from Canvas that provide the mechanisms for handling input events and repainting; thus, the interface to LCDUI is accomplished using the published APIs, and LWT can be integrated with any MIDP-compliant implementation.

This class provides a variety of means to manage Components, layout, focus traversing, and scrolling. There are several compelling reasons for describing these behaviors and their mechanisms in detail. First, it allows a developer to fully exploit the APIs and minimize redundant code. Second, it ensures that clean-room implementations of LWT are fully compatible and behave consistently. Finally, it enables developers to customize behavior without the risk of side effects.

4.7.8.1 Component Management

The basic operation that can be performed on a ComponentScreen is to populate it with Components.

Components can be added and removed from a ComponentScreen. A ComponentScreen cannot be added to another ComponentScreen, and Components cannot be added to another Component.

A Component can have only one parent ComponentScreen at a time, and it can be added to a given ComponentScreen only once. Whenever a Component is added to a ComponentScreen, it is first removed from the current parent if one exists, thereby ensuring that these two rules are enforced.

A ComponentScreen maintains an ordered list of its child Components and assigns each one a unique index. The index of a Component indicates its position in the list where 0 is the first Component, and the highest index is the last Component. Indices are always consecutive, so the index for a given Component may change if other Components are added or removed from the same ComponentScreen. For example, if a Component is added in the middle of the list, the indices of the subsequent Components will be incremented to account for the inserted Component.



Figure 4.6 LWT Component Z-Ordering

A Component may be inserted at a specific valid index, or it may be simply appended at the end of the list and automatically assigned the next index. A Component's index is significant since it implies Z-order and dictates the order in which layout and focus traversal are performed.

The Component with the highest index is considered to be closest to the user, as shown.



4.7.8.2 Component Management Methods

`add(Component w)`

Appends the specified Component to the screen. If the Component is currently added to a screen (including this one), it is automatically removed first.

`insert(Component w, int index)`

Inserts a Component to the screen at the specified index. If the Component is currently added to a screen (including this one), it is automatically removed first. If the index is equal or greater than the number of Components in the screen, the Component will be appended at the end. If the index is less than or equal to 0, the Component will be inserted at the start. Otherwise, the Component is inserted such that its index is equal to the one specified; the indices of subsequent Components in the screen will be incremented to account for the inserted Component.

`getComponent(int index)`

Gets the Component at the specified index.

`getComponentCount()`

Gets the number of Components currently added to this screen.

`remove(Component w)`

Removes the specified Component from the screen. This method does nothing if the specified Component has not been added to this screen.

`remove(int index)`

Removes the Component at the specified index from the screen.

`removeAll()`

Removes all Components from this screen.

4.7.8.3 Rendering

The ComponentScreen is rendered by a call to its `paint()` method. By default, this method first clears the background (i.e., fills it with white pixels) and then renders its Components.

A ComponentScreen subclass may override the default paint method to implement special backgrounds or to render other artifacts on the screen.

The `paintComponents()` method renders the Components in ascending index order. If the Components overlap, the Component with the highest index is rendered last and appears to be closest to the user, thereby implementing the correct Z-order. Invisible Components are not rendered.

4.7.8.4 Rendering Methods

`paint(Graphics g)`

Renders the screen. The repaint region is first cleared (i.e. filled with the appropriate background color), then the Components are rendered by calling `paintComponents()`. Subclasses may override this method to provide special backgrounds, game graphics, etc.

`paintComponents(Graphics g)`



Renders the Components. This method is normally called by the `paint()` method to render the Components on top of the background. The Components are rendered in ascending index order to provide the correct z-order.

4.7.8.5 Layout Management and Validation Cycle

The Layout management refers to the ability of a `ComponentScreen` to determine the location of every Component regarding the layout directives provided by every Component (see “`ComponentScreen` class” on page 91). It also takes into consideration the dimensions of every Component present in the `ComponentScreen`.

To minimize redundant layout computations, the `ComponentScreen` tracks the state of its layout and computes its layout only when necessary. This mechanism effectively consolidates requests for layout computation and defers the layout process until updated Component bounds are actually needed.

For some Components, the preferred size is dependent on Component-specific attributes such as label width, image size, font, etc.. In such cases, a change to one of these attributes may result in a change to the preferred width or height. When such a change occurs, the Component must call `preferredWidthChanged()` or `preferredHeightChanged()`, respectively. These methods invalidate the parent if the preferred dimension is currently being used for the Component's layout; otherwise the change is irrelevant and invalidation is not required.

The `doLayout()` method computes the location of left, right, top, and bottom edges for each Component based on their schemes and accompanying values. The Components are processed in ascending index order. The layout process ignores invisible Components; their edges are not computed and they never become the previous Component.

A `ComponentScreen` becomes invalid when a change is made that could potentially alter the layout of its Components. Such changes include adding or removing Components and changing the edge specifications or visibility of a child Component. When such a change is made, the affected `ComponentScreen` automatically becomes invalid. A `ComponentScreen` can be programmatically made invalid by calling `invalidate()`.

An invalid `ComponentScreen` becomes valid by ensuring that the layout of its children is up to date. The process of validation involves checking whether or not the `ComponentScreen` is invalid; if so, the `doLayout()` method is called and the `ComponentScreen` then becomes valid.

Validation automatically occurs prior to any operation that relies on accurate Component layout information, specifically rendering and pointer event dispatching. A developer can programmatically force validation to occur by calling `validate()`.



4.7.8.6 Layout Management Methods

`invalidate()`

Invalidates this `ComponentScreen`, indicating that its `Components` need to be laid out. The screen is automatically invalidated when a change is made to its `Components` that may require layout to be performed again. These changes include adding or removing `Components`, changing the layout edge specification of a `Component`, or changing the dimensions or visibility of a `Component`.

`validate()`

Validates this `ComponentScreen`. Calling this method will lay out the `Components` if the screen is invalid, otherwise it does nothing.

`doLayout()`

Recomputes the layout of the `Components` according to edge specifications for each `Component`. Subclasses may override this method to implement special layout algorithms if desired.

4.7.8.7 Focus Management

Focus management refers to the ability of a `ComponentScreen` to redirect the input events it receives to one of its `Components` capable of handling input events. In general, none or only one `Component` can have focus at a given time.

Each `ComponentScreen` instance keeps track of its current focus owner. The focus owner is the `Component` within the `ComponentScreen` that receives key events. By default, the focus owner is null, indicating that no `Component` is currently receiving key events; in this case, the `ComponentScreen` continues to receive key events but does not dispatch them to a `Component`. The focus owner may also become null if the current focus owner is removed or it is no longer eligible to maintain focus.

Generally, `ComponentScreen` is responsible for switching the currently focused `Component`. It allows the currently focused `Component` to handle the input events first. If the input event is not consumed by the `Component`, it then attempts to determine if the input event should be translated into a focus change.

The user may traverse focus by the appropriate keys on the device. Focus traversal occurs in `Component` index order and skips any `Components` that are not eligible to receive focus. Focus traversal wraps from the last `Component` to the first `Component` and vice versa.

4.7.8.8 Focus Management Methods

`setFocusNext()`

Moves key event focus to the next `Component` that accepts focus. The first `Component` receives focus if no `Component` currently has focus.

`setFocusPrevious()`

Moves key event focus to the previous `Component` that accepts focus. The first `Component` receives focus if no `Component` currently has focus.

`getFocusOwner()`

Returns the `Component` that currently has key event focus. To assign focus to a specific `Component`, call `requestFocus()` on the desired `Component`.



4.7.8.9 Scrolling

Scrolling is the capability of ComponentScreen to move its view port up or down to guarantee that a particular portion of the ComponentScreen will be within the visible area. The following figure depicts the concept of scrolling. The view port is typically the size of the device screen excluding the command area if commands are present.



Figure 4.7 LWT Scrolling

There are two basic ways of manipulating the scrolling of a ComponentScreen: scrolling based on a Component, or absolute scrolling.

- *Scrolling based on a Component is moving the view port so a given Component becomes visible.*
- *Absolute scrolling is moving the view port an absolute distance measured in pixels.*

ComponentScreen supports vertical scrolling, but does not support horizontal scrolling.

Scrolling is automatically enabled by the native user interface if the bottom edge of the last Component extends past the bottom of the screen. In other words, scrolling support is provided when it is needed, and may be removed when it is not needed.

Whenever a Component receives key event focus, the screen is automatically scrolled when necessary to ensure that the Component is visible to the user.

It is the responsibility of the implementation and native user interface to provide the user with the ability to control the scroll position.

A MIDlet can query and set the scroll position programmatically; however, a MIDlet is not permitted to explicitly enable or disable scrolling since the device implicitly provides this functionality.



4.7.8.10 Scrolling APIs Descriptions

```
scrollTo(Component c)
```

Scrolls to the specified Component. This method ensures that the screen's scroll position is adjusted to show as much of the specified Component as possible.

```
setScrollOffset(int offset)
```

Sets the vertical scroll offset. The scroll offset is automatically constrained such that the screen cannot be scrolled past the top of the screen or the bottom of the last Component. Scrolling is automatically provided by the platform. A scrollbar (or other similar mechanism) is provided by the native UI as needed so that the user can adjust the scroll offset.

```
getScrollOffset()
```

Gets the current vertical scroll offset. The offset indicates the offset into screen that corresponds to the top edge of the visible portion.

4.7.9 The Component Class

Component is the abstract base class of all LWT user interface entities that can be added to a ComponentScreen. The Component class collects core functionality required to create user interface Components capable of interacting with the user.

A Component must be capable of rendering itself as well as capable of responding to input events. A few other basic capabilities provided by the Component class are: location and dimension management, visibility, enabling, and focus.

4.7.9.1 Rendering

Rendering is one of the primary responsibilities of a Component. A Component must render itself when its `paint()` method is called. Under normal circumstances, ComponentScreen calls the `paint()` method of a Component. It is then the responsibility of the Component to render the rectangular area confined by the Component boundaries.

Since the ComponentScreen is responsible for rendering the background, Component does not need to clear the background prior to rendering. Rendering of the background by Component is redundant and reduces user interface performance.

A Component must render itself in a manner that conveys its current state to the user. All Components must render themselves to reflect the following mutually exclusive states:

- *Normal*: Normal appearance of a Component without focus
- *Disabled*: Should be grayed out or drawn with dotted lines instead of solid lines
- *Focus Owner*: Normal appearance with a thick border (a Component needs to support this state only if it accepts key focus)

Component subclasses may include additional states or attributes that affect their appearance. These should also be accounted for by the rendering code.

4.7.9.2 Rendering Methods

```
paint(Graphics g)
```

Renders the Component.

```
repaint()
```



Requests a repaint for the entire Component. This method in turn requests a repaint for the corresponding region of the parent screen.

```
repaint(int x, int y, int width, int height)
```

Requests a repaint for the specified portion of this Component. This method in turn requests a repaint for the corresponding region of the parent screen. The specified region is clipped so that it does not extend beyond the bounds of this Component.

4.7.9.3 Events Handling

There are two types of input events: key events and pointer events. Key events are those generated by the user when he/she actuates a key. Pointer events are those generated by the user when he or she actuates a pointing device on a particular location. Currently, no iDEN handsets include support for pointing devices, although pointer event handling methods are accessible in the LWT package.

Key events are dispatched to the current ComponentScreen through the three methods defined in LCDUI's Canvas. The default implementations of these methods in ComponentScreen check if there is a current focus owner and dispatch the event to that Component, if any. Subclasses may override these methods to implement custom key event handling.

Key events are dispatched to the Component through these three methods: `keyPressed()`, `keyRepeated()`, and `keyReleased()`. These methods return a Boolean to indicate if the Component consumed the key event, thereby allowing ComponentScreen subclasses to implement default behaviors for unconsumed key events.

4.7.9.4 Event Handling Methods

```
keyPressed(int keyCode)
```

Called when a key is pressed. The Component must have key input focus in order to receive key events.

```
keyReleased(int keyCode)
```

Called when a key is released. The Component must have key input focus in order to receive key events.

```
keyRepeated(int keyCode)
```

Called when a key is repeated (held down). The Component must have key input focus in order to receive key events.

4.7.9.5 Location, Dimension and Layout

The location of a Component refers to its absolute (x, y) location within the boundaries of the ComponentScreen in which it is contained.

The dimension of a Component refers to the width and height of the rectangular area that occupies within the ComponentScreen that contains it.

Each Component occupies a rectangular region of its parent ComponentScreen; this is called the Component region. A Component receives pointer events that occur within its rectangular region, and is responsible for rendering the pixels within its region. A Component may render itself as an ellipse, a triangle, a cloud, etc., but its bounding region is always rectangular.



4.7.9.6 Region Parameters

The region is fully described by the location of the upper-left corner of the Component and by the Component's width and height. The location of the upper-left corner is relative to the ComponentScreen's origin and is based on the MIDP coordinate system. Width and height are expressed in terms of pixels.

A developer can query the bounds of a Component by calling `getX()`, `getY()`, `getWidth()`, and `getHeight()` on the Component.

Tip: The values returned by these calls are not guaranteed to be correct if the ComponentScreen has not been validated (see "4.7.8.5 Layout Management and Validation Cycle" on page 94).



Figure 4.8 Region Parameters

4.7.9.7 Preferred Size

Each Component subclass must implement the methods `getPreferredWidth()` and `getPreferredHeight()`.

Together, these two methods specify the ideal dimensions of a given Component instance. Even for the same class, different instances may specify different preferred sizes to reflect the length of a text label, size of an image, etc..

The `preferredWidthChanged()` method must be called whenever the preferred width of the Component changes. Similarly, the `preferredHeightChanged()` method must be called whenever the preferred height of the Component changes. For standard LWT Components, these methods are automatically called when a relevant parameter is changed; for custom Components, it is a developer's responsibility to call these methods whenever a change is made that impacts the preferred width or height of the Component.

4.7.9.8 Layout

LWT's layout model has been designed to provide a developer with complete control over Component placement and size. Although this approach provides the greatest flexibility, it can result in fairly large applications, especially if the application must automatically adjust its layout to account for different display and Component sizes. Therefore, the LWT layout model also incorporates several features that enable the creation of adaptable complex layouts with very little code; furthermore, the execution of these layouts is inherently efficient.

A Component's region is specified in terms of its left, right, top, and bottom edges.



A developer can independently specify the location of each edge using one of several schemes. An accompanying value controls the location of the edge according to the scheme selected.

For all schemes that use an offset, the offset values extend down and to the right. That is, a horizontal offset extends to the right for positive values and to the left for negative values. Similarly, a vertical offset extends down for positive values and up for negative values.

The following schemes may be used for specifying the location of a Component's left edge:

`SCREEN_LEFT` (default) The accompanying `value` describes the edge's offset from the left edge of the screen.



Component Position Specified With `SCREEN_LEFT`

`SCREEN_HCENTER` The accompanying `value` describes the edge's offset from the center of the screen.



Component Position Specified With `SCREEN_HCENTER`



SCREEN_RIGHT The accompanying value describes the edge's offset from the right edge of the screen.



Component Position Specified With **SCREEN_RIGHT**

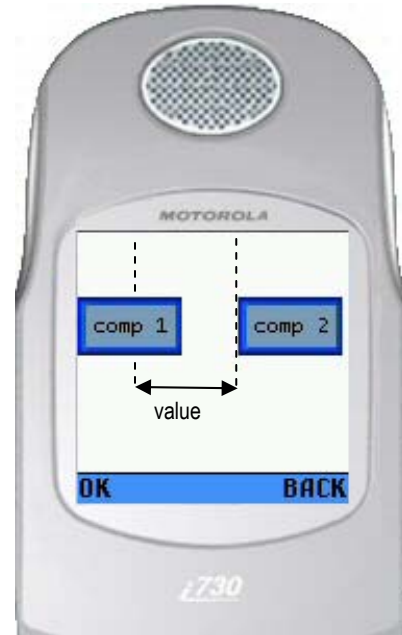
PREVIOUS_COMPONENT_LEFT The accompanying value describes the edge's offset from the left edge of the previous Component. Interpreted as **SCREEN_LEFT** if there is no previous Component.



Component Position Specified With **PREVIOUS_COMPONENT_LEFT**



PREVIOUS_COMPONENT_HCENTER The accompanying value describes the edge's offset from the center of the previous Component. Interpreted as **SCREEN_LEFT** if there is no previous Component.



Component Position Specified With **PREVIOUS_COMPONENT_HCENTER**

PREVIOUS_COMPONENT_RIGHT The accompanying value describes the edge's offset from the right edge of the previous Component. Interpreted as **SCREEN_LEFT** if there is no previous Component.



Component Position Specified With **PREVIOUS_COMPONENT_RIGHT**



The following schemes may be used for specifying the location of a Component's right edge. Developers are encouraged to use `PREFERRED_WIDTH` wherever feasible to maximize application portability across different devices.

`SCREEN_LEFT` The accompanying value describes the edge's offset from the left edge of the screen.



Component Position Specified With `SCREEN_LEFT`

`SCREEN_HCENTER` The accompanying value describes the edge's offset from the center of the screen.



Component Position Specified With `SCREEN_HCENTER`



`SCREEN_RIGHT` The accompanying value describes the edge's offset from the right edge of the screen.



Component Position Specified With `SCREEN_RIGHT`

`PREVIOUS_COMPONENT_LEFT` The accompanying value describes the edge's offset from the left edge of the previous Component. The Component is set to its preferred width if there is no previous Component.



Component Position Specified With `PREVIOUS_COMPONENT_LEFT`



PREVIOUS_COMPONENT_HCENTER The accompanying value describes the edge's offset from the center of the previous Component. The Component is set to its preferred width if there is no previous Component.



Component Position Specified With **PREVIOUS_COMPONENT_HCENTER**

PREVIOUS_COMPONENT_RIGHT The accompanying value describes the edge's offset from the right edge of the previous Component. The Component is set to its preferred width if there is no previous Component.



Component Position Specified With **PREVIOUS_COMPONENT_RIGHT**



WIDTH The right edge is located such that the Component's width is equal the accompanying value.



Component Position Specified With `WIDTH`

PREFERRED_WIDTH (default) The right edge is located such that the Component's width is equal to its preferred width plus the accompanying value.



Component Position Specified With `PREFERRED_WIDTH`



The following schemes may be used for specifying the location of a Component's top edge:

SCREEN_TOP The accompanying value describes the edge's offset from the top edge of the screen.



Component Position Specified With SCREEN_TOP

PREVIOUS_COMPONENT_TOP The accompanying value describes the edge's offset from the top edge of the previous Component. Interpreted as **SCREEN_TOP** if there is no previous Component.



Component Position Specified With PREVIOUS_COMPONENT_TOP



`PREVIOUS_COMPONENT_VCENTER` The accompanying value describes the edge's offset from the center of the previous Component. Interpreted as `SCREEN_TOP` if there is no previous Component.



Component Position Specified With `PREVIOUS_COMPONENT_VCENTER`

`PREVIOUS_COMPONENT_BOTTOM` (default) The accompanying value describes the edge's offset from the bottom edge of the previous Component. Interpreted as `SCREEN_TOP` if there is no previous Component.



Component Position Specified With `PREVIOUS_COMPONENT_BOTTOM`



The following schemes may be used for specifying the location of a Component's bottom edge. Developers are encouraged to use `PREFERRED_HEIGHT` wherever feasible to maximize application portability across different devices.

`SCREEN_TOP` The accompanying value describes the edge's offset from the top edge of the screen.



Component Position Specified With `SCREEN_TOP`

`PREVIOUS_COMPONENT_TOP` The accompanying value describes the edge's offset from the top edge of the previous Component. The Component is set to its preferred height if there is no previous Component.



Component Position Specified With `PREVIOUS_COMPONENT_TOP`



`PREVIOUS_COMPONENT_VCENTER` The accompanying value describes the edge's offset from the center of the previous Component. The Component is set to its preferred height if there is no previous Component.



Component Position Specified With `PREVIOUS_COMPONENT_VCENTER`

`PREVIOUS_COMPONENT_BOTTOM` The accompanying value describes the edge's offset from the bottom edge of the previous Component. The Component is set to its preferred height if there is no previous Component.



Component Position Specified With `PREVIOUS_COMPONENT_BOTTOM`



HEIGHT The bottom edge is located such that the Component's height is equal the accompanying value.



Component Position Specified With HEIGHT

PREFERRED_HEIGHT (default) The bottom edge is located such that the Component's height is equal to its preferred height plus the accompanying value.



Component Position Specified With PREFERRED_HEIGHT



4.7.9.9 Layout Methods

`getX()`

Gets the x coordinate of the Component's left edge within the parent.

`getY()`

Gets the y coordinate of the Component's top edge within the parent.

`getWidth()`

Gets the width of the Component, in pixels.

`getHeight()`

Gets the height of the Component, in pixels.

`getPreferredWidth()`

Gets the preferred width of this Component. Subclasses must implement this method and return the preferred width for this Component. This method may be called quite frequently so it must execute quickly and without creating garbage. The preferred width of a Component may depend on a number of factors including Component attributes, the device's font sizes, and the device's native look and feel. The preferred size of a Component is the minimum size that enables the Component to render itself fully without being clipped.

`getPreferredHeight()`

Gets the preferred height of this Component. Subclasses must implement this method and return the preferred height for this Component. This method may be called quite frequently so it must execute quickly and without creating garbage. The preferred height of a Component may depend on a number of factors including Component attributes, the device's font sizes, and the device's native look and feel. The preferred size of a Component is the minimum size that enables the Component to render itself fully without being clipped.

`preferredWidthChanged()`

Notifies the system that the preferred width of this Component has changed. This method checks if this Component is currently using preferred width for its layout; if so, it invalidates the parent screen so that the layout is recomputed using the new preferred width. If the preferred width is not being used for the Component's layout, the change will have no effect on layout and invalidation does not need to occur.

`preferredHeightChanged()`

Notifies the system that the preferred height of this Component has changed. This method checks if this Component is currently using preferred height for its layout; if so, it invalidates the parent screen so that the layout is recomputed using the new preferred height. If the preferred height is not used for the Component's layout, the change will have no effect on layout and invalidation does not need to occur.

`setLeftEdge(int schema, int value)`

Specifies the location of the Component's left edge.

`setRightEdge(int schema, int value)`

Specifies the location of the Component's right edge.

`setTopEdge(int schema, int value)`



Specifies the location of the Component's top edge.

```
setBottomEdge(int schema, int value)
```

Specifies the location of the Component's bottom edge.

4.7.10 Component Visibility, State & Focus

4.7.10.1 Visible & Invisible

A visible Component is shown to a user, whereas an invisible Component is not. Components may be hidden to conceal portions of the user interface that are not relevant, thereby simplifying the user interface. By default, all Components are initially visible.

4.7.10.2 Enabled & Disabled

Enabling indicates whether or not a Component is currently available to a user. Enabling and disabling is useful for conveying the availability of certain features that may be temporarily unavailable based on the current context. For example, a View button should be disabled if the corresponding list contains no items. By default, all Components are initially enabled.

4.7.10.3 Focused & Unfocused

The concept of focus in a Component refers to the capabilities of the Component to handle input events. ComponentScreen relies on every Component to determine if such Component is a focus candidate. ComponentScreen queries every Component when focus traversing is taking place.

A Component must implement certain API if it wants the ComponentScreen to manage the focus correctly.

A Component may indicate whether or not it is interested in ever becoming the focus owner by setting the Boolean field `acceptsKeyFocus` to the appropriate value. If this field is set to true, the Component may gain focus; if false, the Component will never gain focus.

In order to be eligible to gain focus, the Component must be visible, enabled, and have its `acceptsKeyFocus` field set to true.

A Component may programmatically request focus by calling `requestFocus()`.

Whenever a Component gains key focus, its `gainedFocus()` method is called. Similarly, its `lostFocus()` method is called whenever it loses focus. A Component can query whether or not it has focus by calling `hasFocus()`.

4.7.10.4 Component Visibility, State & Focus Methods

```
isVisible()
```

Checks if this Component is visible (can be seen by the user). Note that a Component is still considered to be visible even if it is scrolled off the screen or if the screen is not currently shown. Components are visible by default.

```
setVisible(boolean visible)
```

Shows or hides the Component. A Component is visible by default. This method automatically repaints the Component when its visibility changes.

```
isEnabled()
```



Checks if this Component is currently enabled (can be interacted with by the user).

```
setEnabled(boolean enable)
```

Enables or disables this Component. An enabled Component receives input events that it should receive; a disabled Component does not receive any input events. This method automatically repaints the Component when its enabled state changes. A Component is enabled by default.

```
acceptsFocus()
```

Checks if this Component currently accepts key focus. When the user traverses the screen, only those Components that accept focus will receive it; other Components will be skipped. In order for this method to return true, the `acceptsKeyFocus` field must be set to true (set to false by default), the Component must be visible, and the Component must be enabled.

```
hasFocus()
```

Checks if this Component currently has key focus (that is, it is receiving key events). For a given screen, no more than one Component can have key focus.

```
requestFocus()
```

Requests key focus for this Component. For a Component to receive focus, it must be added to a ComponentScreen, and `acceptsFocus()` must return true.

```
gainedFocus()
```

Called when this Component gains key focus. For a Component that accepts key focus, this method is called when it gains focus; it may use this method to change its appearance, etc. By default, this method requests a repaint.

```
lostFocus()
```

Called when this Component loses key focus. For a Component that accepts key focus, this method is called when it loses focus; it may use this method to change its appearance, etc. By default, this method requests a repaint.

4.7.11 The ComponentListener Interface

The ComponentListener interface is implemented by any class that receives events from a Component. The ComponentListener is notified of an event by calling its `processComponentEvent()` method with the source Component reference and an integer identifying the event type.

Every Component that wishes to notify a ComponentListener is responsible for defining the events that can generate, typically as static fields. The Component is also responsible to call the method `processComponentEvent()` on the ComponentListener, if there is one available, when any of the defined events take place.

The two parameters passed when the `processComponentEvent()` method is called are the following: (1) an Object reference to the source of the event (typically the Component itself), and (2) the event defined as an integer value (typically a static value defined by the subclass of Component). Refer to the example in the “4.7.13 The Button Class” section on page 116.

4.7.12 The InteractableComponent Class

InteractableComponent is the abstract base class of the Components that a user can ‘press’ and ‘release’. Such Components include buttons, checkboxes, and icons. This class serves to reduce



code size and complexity of its subclasses by providing the basic interaction functionality. An `InteractiveComponent` is typically actuated by tapping and releasing within its bounds, or by pressing and releasing the Send key when the Component has focus.

The `InteractiveComponent` class provides some useful services to enhance the usability of the Component with little or no code at all. These services can be grouped in three main categories: aspect, low-level event handling, and high-level event handling.

4.7.12.1. Aspect

Aspect refers to how the Component is rendered to convey its meaning and functionality.

`InteractiveComponent` has a `String` that is rendered to depict the meaning of the Component, its label. The location where the label is rendered regarding the boundaries of the Component depends on the implementation provided by subclasses.

`InteractiveComponent` has a reference to a `Font` object that is used to render the label.

`InteractiveComponent` also has a Boolean state that represents if the Component is 'pressed' or 'unpressed'. This concept can also be used to convey the logical state of the Component.

4.7.12.2 Aspect Methods

```
getLabel()
```

Gets the label for this Component.

```
setLabel(String l)
```

Sets the label for this `InteractiveComponent`.

```
getFont()
```

Gets the `Font` associated with this Label.

```
setFont(Font f)
```

Sets the `Font` object for rendering the label, if any.

```
isPressed()
```

Checks if this `InteractiveComponent` is currently pressed.

```
setPressed(boolean b)
```

Sets the pressed/unpressed state of this Component.

4.7.12.3 Event Handling

The low level event handling functionality basically overloads the input event handling API specified by Component.

`InteractiveComponent` adds the appropriate logic to alter the 'pressed' state depending on the user input event handling. For instance, an `InteractiveComponent` appears to remain pressed while the key that actuates the Component is being held.



4.7.12.4 Event Handling

Besides the low level event handling API, `InteractableComponent` provides a high level event handling which follows the listener paradigm commonly used in Java.

The listener paradigm allows any entity to be notified when the `InteractableComponent` has been actuated ('pressed' and 'released').

`InteractableComponent` is responsible for notifying its `ComponentListener` with the appropriate information when is actuated. The information provided to the `ComponentListener` is the `InteractableComponent` that was actuated and the event.

4.7.12.5 Event Handling Methods

`componentActuated()`

Called when the Component is actuated (tapped and released).

`dispatchComponentEvent(int event)`

Dispatches the specified event to this `InteractableComponent`'s listener, if any.

`setComponentListener(ComponentListener l)`

Sets this `InteractableComponent`'s listener.

4.7.13 The Button Class

As a subclass of `InteractableComponent`, the `Button` class mainly relies on the functionality implemented by `InteractableComponent`.

A `Button` is a basic button that a user can actuate. A button can display text to convey its meaning. The text font is the only customizable attribute of the `Button` class.

`Button` provides full advantage of the three basic services described in the `InteractableComponent` Class with minimal effort. The developer is also given flexibility to implement `Button` methods in different ways to achieve other results.

There are two different constructors to create a `Button`: a constructor with no arguments and a constructor with a `String` argument.

4.7.13.1 Example: Button

The following example creates two `Buttons` that can be actuated when added to a `ComponentScreen`. The first `Button` is clickable and changes the text of the second button when actuated. The second `Button` can be pressed but has no effect.

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import com.motorola.lwt.*;
import java.io.IOException;

public class HelloLWTWorld extends MIDlet implements ComponentListener
{
    Display display;
    ComponentScreen scr;
    String[] toggleStr = {
        "off",
        "on"
    }
}
```




```
};
Button bClick;
Button bToggle;
static final int TOGGLE_BUTTON_WIDTH = 40;

public HelloLWTWorld() {
    // get display
    display = Display.getDisplay(this);

    // create the ComponentScreen
    scr = new ComponentScreen();

    // create 2 buttons
    bClick = new Button("click");
    bToggle = new Button(toggleStr[0]);

    // make the toggling button 40 pixels wide
    bToggle.setRightEdge(Component.WIDTH, TOGGLE_BUTTON_WIDTH);

    // place the Buttons in the center of the screen
    bClick.setLeftEdge(Component.SCREEN_HCENTER,
        -bClick.getPreferredWidth()/2);
    bClick.setTopEdge(Component.SCREEN_TOP,
        bClick.getPreferredHeight()/2);
    bToggle.setLeftEdge(Component.SCREEN_HCENTER,
        -TOGGLE_BUTTON_WIDTH/2);
    bToggle.setTopEdge(Component.PREVIOUS_COMPONENT_BOTTOM,
        bToggle.getPreferredHeight()/2);

    // set the ComponentListener which will control the toggle
    // capability of the toggling Button when the
    // clicking button is actuated
    bClick.setComponentListener(this);

    // add both Buttons to the ComponentScreen
    scr.add(bClick);
    scr.add(bToggle);
}

public void processComponentEvent(Object source, int event) {
    if (source == bClick) {
        // toggle the current mode
        bToggle.setPressed(!bToggle.isPressed());

        // determine the mode
        int mode = bToggle.isPressed()? 1 : 0;

        // change the label according to the mode
        bToggle.setLabel(toggleStr[mode]);
    }
}

public void startApp() {
```



```

        display.setCurrent(scr);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean b) {
    }
}

```

4.7.14 The ImageLabel Class

The ImageLabel class heavily relies on the functionality implemented by InteractableComponent.

ImageLabel allows to take full advantage of the three basic services described in “4.7.12 The InteractableComponent Class” section on page 114 with almost no effort. However, a developer is allowed to uniquely implement any of the methods provided by ImageLabel to achieve different results.

An ImageLabel is a general-purpose Component that can display an image and/or a text label; it can be an interactive or a read-only Component.

4.7.14.1 Image Manipulation

If an image is displayed, a developer can use either a single image or multiple images to reflect a Component's different states (such as pressed, disabled, etc.).

The ‘normal’ image is used when the ImageLabel is rendered in its normal state ‘unpressed’.

The ‘disabled’ image is used when the ImageLabel is rendered in its disabled state. An ImageLabel is in its disabled state when its method `isEnabled()` returns false.

The ‘pressed’ image is used when the ImageLabel is rendered in its pressed state. An ImageLabel is pressed when its `isPressed()` method returns true.

4.7.14.2 Image Manipulation Methods

```
setNormalImage(Image n)
```

Sets the image for the normal state.

```
setDisabledImage(Image d)
```

Sets the image for the disabled state.

```
setPressedImage(Image p)
```

Sets the image for the pressed state.

4.7.14.3 Text Manipulation

If text is displayed, a developer can specify the text and its font. See the “4.7.12.1. Aspect” section on page 115 for the corresponding API. Also, there is a mechanism to provide the color scheme used by the ImageLabel. The two colors that can be manipulated are the background color and the foreground color.

The following figure shows how these two colors are used.



Figure 4.9 ImageLabel With Text

The color scheme also provides a useful transparent color that can be used to render portions of the ImageLabel as transparent.

4.7.14.4 Text Manipulation Methods

`getBackgroundColor()`

Gets the background color of the text.

`getForegroundColor()`

Gets the foreground color of the text.

`setBackgroundColor(int bgc)`

Sets the foreground color of the text.

`setForegroundColor(int bgc)`

Sets the background color of the text.

4.7.14.5 Label Location

If both text and an image are displayed, the location of the text relative to the image can be specified as above, below, to the left, to the right, and centered.



Figure 4-10 ImageLabel Text Positions

4.7.14.6 Label Location Methods

```
setLabelLocation(int location)
```

Sets the location of the label, if any, relative to the ImageLabel's image.

4.7.14.7 Alignment

Regardless of what is displayed (text, image, or both), the collective alignment of the image and/or text within the bounds of the ImageLabel may be specified as: North, South, East, West, and Centered.

4.7.14.8 Alignment Methods

```
setAlignment(int alignment)
```

Sets the desired alignment for this ImageLabel.



4.7.15 Checkboxes

4.7.15.1 The Checkbox Class

As a subclass of `InteractableComponent`, the `Checkbox` class heavily relies on the functionality implemented by `InteractableComponent`.

Used as is, `Checkbox` provides a single, independent Boolean choice. A `Checkbox` displays text to convey its meaning through its label.

`Checkbox` takes full advantage of the three basic services described in “4.7.12 The `InteractableComponent` Class” section on page 114 with almost no effort.

4.7.15.2 The `CheckboxGroup` Class

A `CheckboxGroup` is a non-UI object that manages one or more `Checkboxes`. It can be configured to enforce multiple selection or exclusive selection rules, and can be used to query the current values of the checkboxes.

When used in conjunction with a `CheckboxGroup`, `Checkboxes` can provide a list of exclusive choices in the form of radio buttons or a list.

The following figure shows the three different styles allowed in a `CheckboxGroup`. The type must be specified when the constructor is called.



Figure 4-12 Checkbox Examples

`Checkboxes` must still be added to the `ComponentScreen`; adding them to the `CheckboxGroup` only impacts their behavior.

`CheckboxGroup` provides a complete set of operations to manipulate `Checkboxes`. `Checkboxes` can be added to or removed from a `CheckboxGroup` as needed. Also, it provides mechanisms to query and modify the Boolean state of every `Checkbox`.



4.7.15.3 CheckboxGroup Management Methods

`add(Checkbox cb)`

Adds a checkbox to this group and returns the index assigned to it.

`insert(Checkbox cb, int index)`

Inserts a checkbox into this group at the specified index.

`remove(Checkbox cb)`

Removes the specified checkbox from this group.

`remove(int index)`

Removes the checkbox with the specified index from this group.

`getCheckboxCount()`

Gets the Checkbox with the specified index.

`getCheckbox(int index)`

Gets the Checkbox with the specified index.

`getSelectedIndex()`

Gets the index of the selected element.

`isSelected(int index)`

Gets the value of the Checkbox with the specified index.

`setSelectedFlags(boolean[] selectedArray)`

Sets the values of the group's Checkboxes to the values of the provided array.

`setSelectedIndex(int index, boolean value)`

Sets the selection for this CheckboxGroup.

4.7.16 The TextComponent Class

TextComponent is the abstract base class for Components that can display and edit text. It provides common functionality such as text manipulation, constraints, and input event handling.

4.7.16.1 Aspect & Render

The aspect of a TextComponent can be controlled by the mechanisms provided by Component.

Also, TextComponent provides three mechanisms to change the way that the text is rendered: (1) a method to change the Font used to render the text, (2) an echo character to be used when the real characters are not meant to be displayed (for example in a password field), and (3) justification.

TextComponent has the capability to render the text justified in three different ways: left, right, and centered.



4.7.16.2 Aspect & Render Methods

`getFont()`

Gets the font currently used by this text Component.

`setFont(Font f)`

Sets the current font of the text control.

`getEchoChar()`

Obtains the echo character used by this text Component.

`setEchoChar(char c)`

Sets the echo character to be displayed by this text Component.

`setJustification(int justification)`

Sets the justification of this text Component.

4.7.16.3 Text Manipulation

TextComponent provides a variety of mechanism to manipulate the contained text. Operations such as append, insert, set, and retrieve are available at the character level as well as at the String level.

4.7.16.4 Text Manipulation APIs Descriptions

`getText()`

Obtains the text contained in this text Component.

`appendText(String t)`

Appends the specified text at the end of the current text.

`insertText(String t, int index)`

Inserts the specified text at the specified index in the current text.

`setText(String t)`

Sets the contents of the text control to the specified String.

`appendChar(char c)`

Appends the specified character at the end of the current text.

`insertChar(char c, int index)`

Inserts the specified character at the specified index in the current text.

4.7.16.5 Input Constraints

Different constraints allow the application to request that the user's input be restricted in a variety of ways. For example, if the application requests the `NUMERIC` constraint, TextComponent will only allow numeric characters to be entered.

Another way of restricting the input is setting the maximum number of characters that can be entered.

TextComponent provides almost all of the input constraints available in MIDP 2.0 TextField: `NUMERIC`, `ANY`, `URL`, `EMAILADDR`, and `PHONENUMBER`. The `PASSWORD` constraint is available



in MIDP 2.0, but it is not supported. If password functionality is desired, the echo char mechanism must be used instead.

4.7.16.6 Input Constraints Methods

`getConstraints()`

Gets the text entry constraint for this `TextComponent`.

`setConstraints(int constraints)`

Sets the text entry constraint for the contents of this `TextComponent`.

`getLengthLimit()`

Gets the length limit.

`setLengthLimit(int maxChars)`

Sets the length limit.

4.7.16.7 High Level Input Event Handling

`TextComponent` exposes the standard low level input event handling mechanisms provided in `Component` (see “4.7.9.3 Events Handling” on page 98) but it also provides a higher-level input event handling.

The high level input event handling mechanism provided by `TextComponent` notifies a subclass of `TextComponent` that a range of characters has been altered. Typically, this notification takes place when characters in the text are changed, inserted or added. Also, the mechanism notifies if the change was originated by the user or it was generated programmatically.

4.7.17 The TextField Class

`TextField` is a single-line `TextComponent` designed to display and edit text. `TextField` supports horizontal scrolling only.

When a `TextField` is created, two parameters can be provided: the initial text and the number of columns.

4.7.18 The TextArea Class

`TextArea` is a multi-line `TextComponent` designed for displaying and editing text. `TextArea` supports vertical scrolling only.

When a `TextArea` is created, three parameters can be provided: the initial text, the number of rows, and the number of columns.

4.7.19 The Slider Class

The `Slider` is a gauge-type `Component` that provides a graphical representation of a numeric value. A `Slider` can be read-only or adjustable.

A read-only `Slider` might be used to indicate memory usage or battery level; an adjustable `Slider` might be used to adjust a volume level.



The current value of a Slider represents the current setting or level of the Slider, which can be between 0 and the maximum value, inclusive. The maximum value of a Slider may be set programmatically to any non-negative integer value.

A Slider does not include a label; a developer can add labels and icons to the ComponentScreen to indicate meanings, endpoint values, etc.

When a Slider is created, three parameters must be specified: (1) a Boolean value to specify whether the slider can be operated by the user, (2) the maximum value of the slider, and (3) the initial value.

The Slider class provides mechanisms to set and retrieve the current value and the maximum allowed value.

4.7.19.1 Slider Methods

```
Slider(boolean interactive, int maxValue, int value)
```

Creates a new Slider of the desired type with the specified maximum and initial values.



4.8 Graphics Acceleration

4.8.1 Overview



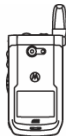
This feature is only available on these handsets.

Some handsets support graphics acceleration in hardware. Graphics acceleration allows an application to perform graphics routines faster because these routines are executed in parallel by a specialized processor: the hardware graphics accelerator. This relinquishes the main processor to execute other operations while the graphics accelerator takes care of the graphics operations.

Graphics acceleration is a feature that cannot be controlled programmatically. In other words there is no API to interact with the hardware graphics accelerator directly. However, by knowing the rules that the hardware accelerator uses to speed rendering, an application developer can take advantage of acceleration. All graphics routines are accelerated. In certain cases, the acceleration of a particular routine may cause the degradation of a subsequent routine due to physical limitations in the hardware. That is why it is very important that the developer knows the rules that are applied.

4.8.2 iDEN-Graphics-Acceleration: on | off | auto

The only explicit control available allows developers to turn on or off graphics acceleration. When on is selected, all graphics routines are executed by the specialized hardware. When off is selected, all graphics routines are executed by the main processor. There is a third mode available to control the hardware acceleration: auto. The auto mode directs the platform to measure performance, in frame rate, of the two modes of operation: on and off. The mode which gives the highest frame rate will be set as the mode of operation. Auto mode is not recommended as the default mode because the frame rate calculation does not know what screen is being rendered. For instance, in an application with introduction screens, the frame rate of such introduction screens will be taken into consideration and compared with a more graphics intensive screen; thus, leading to the incorrect determination of the mode. When the JAD tag is set to auto, the handset determines which mode is better each time the application is resumed. The frame rate of each mode will be measured and the mode with the highest frame rate will be set each time the application is resumed.



By default, hardware acceleration is off. This means that no application, by default, attempts to release the main processor from executing graphics routines and assigning the execution of such routines to the specialized hardware.



By default, hardware acceleration is on.



4.8.3 How it works

The graphics accelerator uses a specialized Video Random Access Memory (VRAM). Operations carried out by the graphics accelerator in VRAM can be done faster than the same operations carried out by the main processor in regular RAM.

When a Java application executes, all objects associated with that Java application live in the Java heap, which is system memory. A Graphics object (`javax.microedition.lcdui.Graphics`) as well as an Image object (`javax.microedition.lcdui.Image`) has control over system memory space that contains all its pixels; this is called the raster (see figure below). Rasters are also in system memory.

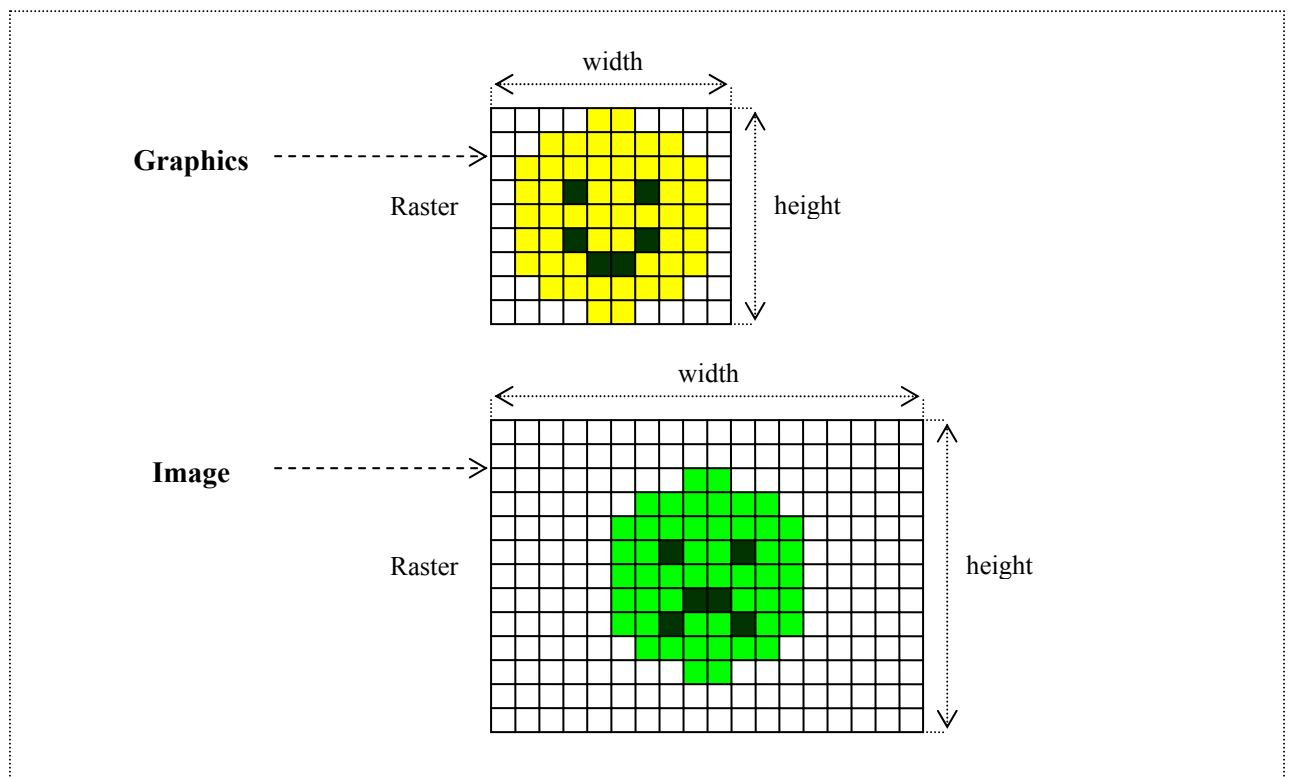


Figure 4.13

There are two kinds of rasters: mutable and immutable. A mutable raster typically can be used as the destination of a graphical operation. An immutable raster is typically derived from a decoded image and cannot be used as the destination of a graphical operation. Generally, an immutable raster is used as the source (see example).

```
Image img = Image.createImage(100, 100); // blank image => mutable
raster

// the g_raster           Graphics g = img.getGraphics();
// g_raster               g.drawLine(0, 0, 100, 100);
// g_raster used as a destination

Image pic = Image.createImage("pic.png");
```



```
// immutable img => immutable raster
// the pic_raster          g.drawLine(0, 0, 100, 100);
// pic_raster used as source and
// g_raster is used as destination
```

The graphics accelerator outperforms the main processor when the destination raster and source raster when applicable are in VRAM. For graphics acceleration to take place, rasters must be copied from system memory to VRAM. If a raster is mutable and it is used as the destination of a graphical operation, it is copied from system memory to VRAM; this is called caching. Similarly, an immutable raster is cached when it is used as the source of a graphical operation (i.e. `drawImage`) and the destination raster has already been cached.

Graphics acceleration does not necessarily improve the very first render. In fact, the very first render would be slower because it has the overhead of caching the destination raster (and source raster when applicable). The benefit is seen in subsequent render operations because the destination raster has been already cached. Let's analyze the following code snippet:

```
g.setColor(0xffffffff); // set color to white
g.fillRect(0,0,100,100); // (1)
g.drawLine(0,0,100,0); // (2)
g.drawLine(0,1,100,1); // (3)
```

When there is no hardware acceleration, the main processor must fill the rectangular area (0,0) through (100,100) in white and then render 2 lines of 100 pixels each. At a glance, that is executing the overhead code for filling a rectangle and writing 10000 pixels (100x100). Then, the overhead of drawing a line and writing 100 pixels; this has to be done twice, one for each line. The overhead to fill a rectangle or draw a line is fairly constant; it does not depend on the size and position of the rectangle or the position, length or slope of the line. The time consumed to draw a single pixel is constant so the total elapsed time grows depending on the number of pixels to write.

When there is hardware acceleration, the main processor must interact with the hardware accelerator via commands. But before that, the main processor determines if the destination raster has to be cached or if it has been cached before. If the raster has not yet been cached, then it caches it. Then, by issuing the appropriate commands to the hardware accelerator, the graphics accelerator writes the pixels. Thus, caching the destination raster and issuing the commands is considered overhead. And, because the hardware accelerator is the one writing the pixels in parallel from the main processor, the drawing time is considered constant and relatively small compared to the no hardware acceleration case. In other words, the elapsed time to render the pixels does not depend on the number of pixels; it is constant from the main processor point of view. However, the overhead time changes depending on the raster management and VRAM management the main processor has to do before issuing the commands to the hardware accelerator. In other words, the overhead varies depending on whether the rasters are or are not in VRAM.

Now, returning to the code snippet, the overhead of (1) will be the highest because the destination raster associated with Graphics object `g` (`g_raster`) has not been cached. The main processor must copy the contents of the destination raster to VRAM and issue the fill rectangle command. However, when operation (2) is executed, the overhead is smaller because the destination raster is already cached. So, the only overhead applicable is the one for issuing the draw line commands to the hardware accelerator. The same occurs with operation (3).

As you can see, as long as the destination raster stays cached, the overall elapsed time to complete the drawing operations will be smaller than the elapsed time of the no hardware acceleration case. This is because it does not grow based on the number of pixels being rendered.



However, there are physical limitations that the developer must be aware of. The hardware accelerator has a limited amount of VRAM. If a destination raster has not been cached, the platform will attempt to cache it. Caching of a raster only occurs when the raster is involved in the drawing operation: either as destination or as source. When there is not enough VRAM available to cache the raster, the least recently used rasters are demoted (decached). The contents of these rasters may be copied from VRAM into system memory to keep their integrity according to Java specifications. For instance, when an immutable raster is demoted, its contents are not copied to system memory. When a mutable raster is demoted, its contents must be copied to system memory. Remember that a mutable raster is maintained by the application and its contents are persistent.

Let's consider the following code snippet:

```
g.fillRect(0,0,100,100);           // (1)
g.drawImage(img50x50,25,25,0);    // (2) render the 50x50 image
                                   // at position (25,25).
g.drawImage(img20x20,60,60,0);    // (3) render the 20x20 image
                                   // at position (60,60).
```

When hardware acceleration is active, the main processor will cache the destination raster and then it will issue the fill rectangle command in order to complete operation (1). Then, the main processor will determine if the raster associated with `img50x50` has been cached. If it has not been cached, it will cache it. Then, it will issue the draw image command to the hardware accelerator to complete operation (2). Thereafter, the main processor will determine if the raster associated with `img20x20` has been cached. It will be cached because it has not been cached before. At this point, depending on the VRAM available, the demotion of the raster associated with `img50x50` may be triggered because there is not enough VRAM to accommodate all three rasters at the same time. Since the main processor knows that destination rasters are not to be demoted to cache source rasters, it will not demote the destination raster. Though, it will demote the least recently used raster; in this case, the one associated with `img50x50`. Then, it will cache the raster of `img20x20` and command the graphics accelerator to draw the source raster on top of the destination raster to complete operation (3). In the ideal case of unlimited VRAM, all rasters will be cached at all time so no demotion is needed. In reality, the limited amount of VRAM may cause excessive caching and demotion which cancels out the effect of the acceleration. Remember that acceleration is effective when source and destination are both in VRAM and the overhead is minimal. In other words, when source and destination are both cached and any subsequent render operation does not pay the overhead of caching.

Let's consider a more typical case, the following two game scenes. For simplification purposes, each raster is represented at the pixel level. That means that each square represents a pixel. The vertical and horizontal lines are used as guidelines to ease the visualization of each frame and tile.

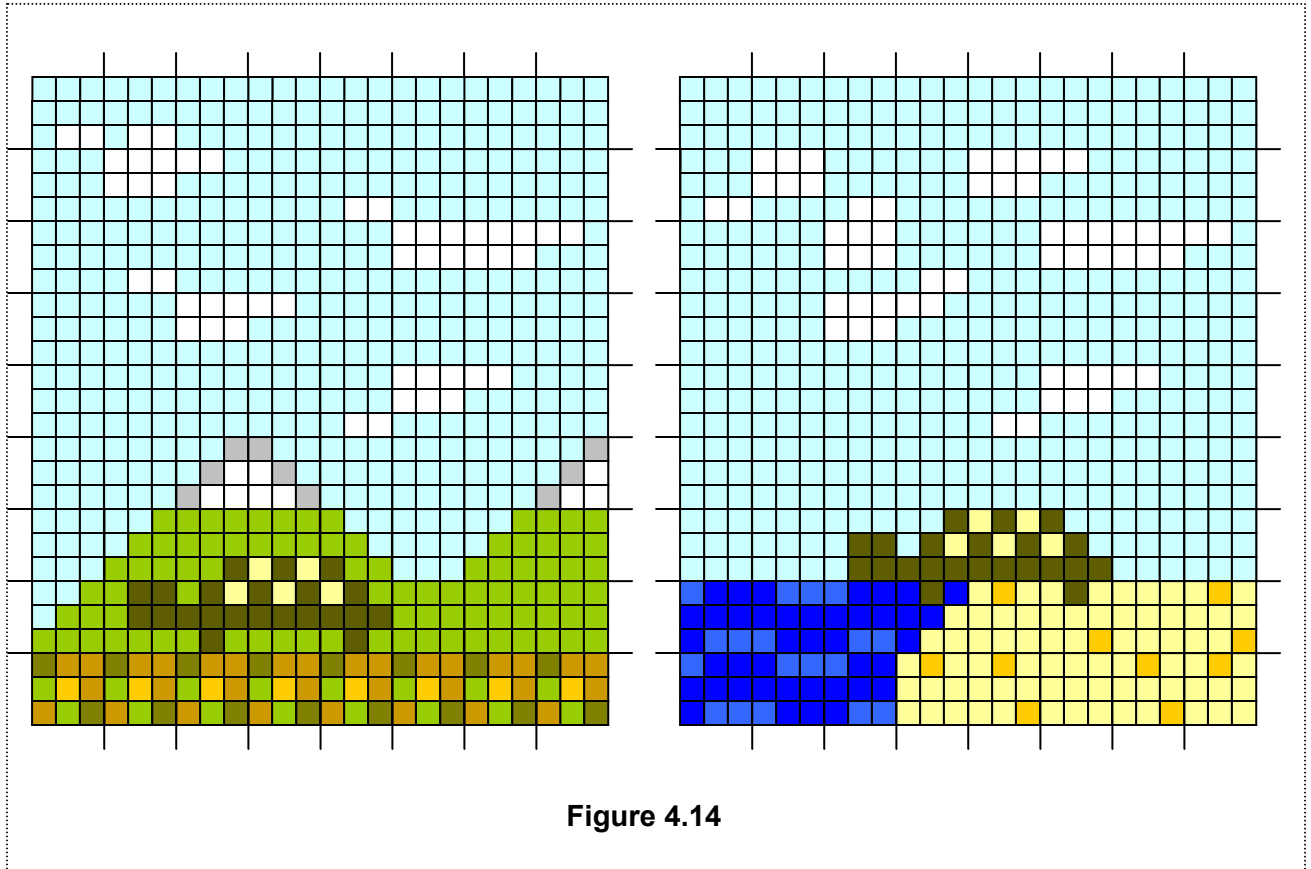


Figure 4.14

Each scene is composed by sprites. Typically, a single sprite is in fact a subsection (a frame) of a large image containing all frames. Please refer to JSR-118 for a more comprehensive explanation of Sprites, Tiles, and related concepts.

A developer has multiple means to manage the sprites. One possible way is to create a single image that contains all frames of all sprites or tiles (1). Another way is to create multiple images that contain all frames of a single sprite (2). There are many possible combinations. As a developer you will have to pick the one that works best for you. It all depends on the complexity of the scene and how the VRAM is used.

The next figures illustrate the 2 options we just described. Figure 4.15(a) contains all tiles of both scenes.

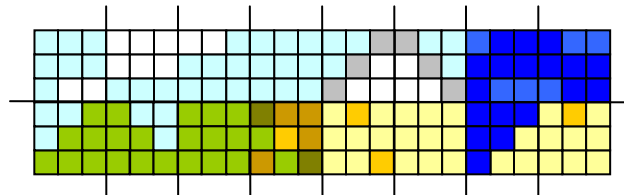


Figure 4.15(a) tiles.png



Figure 4.15(b) contains all tiles used in scene 1.

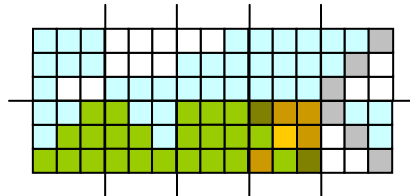


Figure 4.15(b) tiles0.png

Figure 4.15(c) contains the tiles used in scene 2.

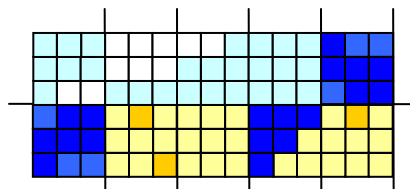


Figure 4.15(c) tiles1.png

Figure 4.15(d) contains the frames to create the sprite.

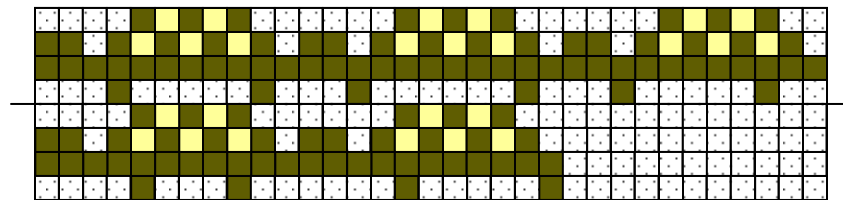


Figure 4.15(d) turtle.png

The following code snippet shows how to create the necessary sprites and layers to compose the scenes.

```
// create the sprite
Sprite turtle = new Sprite("/turtle.png", 11, 4); // the turtle is 11x4
pixels

// matrices to compose both scenes backgrounds
static int[][][] scenes =
{
    { /* the mountains */
        { 1, 1, 4, 4, 4, 4, 4, 4 },
        { 4, 2, 3, 4, 1, 4, 4, 4 },
    }
}
```



```

    { 4, 1, 4, 4, 4, 2, 2, 3 },
    { 4, 4, 2, 3, 4, 4, 4, 4 },
    { 4, 4, 4, 4, 1, 2, 3, 4 },
    { 4, 4, 5, 6, 4, 4, 4, 5 },
    { 4, 9, 11, 11, 10, 4, 9, 11 },
    { 9, 11, 11, 11, 11, 11, 11, 11 },
    { 12, 12, 12, 12, 12, 12, 12, 12 }
},
{ /* the beach */
    { 4, 4, 4, 4, 4, 4, 4, 4 },
    { 1, 2, 1, 4, 2, 3, 4, 4 },
    { 4, 4, 2, 1, 4, 2, 2, 3 },
    { 4, 4, 2, 3, 4, 4, 4, 4 },
    { 4, 4, 4, 4, 1, 2, 3, 4 },
    { 4, 4, 4, 4, 4, 4, 4, 4 },
    { 4, 4, 4, 4, 4, 4, 4, 4 },
    { 7, 8, 7, 15, 16, 13, 14, 13 },
    { 7, 8, 7, 16, 13, 14, 13, 16 }
},
};

// create the tiles used for the scenes. Each tile is 3x3 pixels
Image tiles = Image.createImage("/tiles.png");

// create scene (both scenes are same size so we can use same
TiledManager)
TiledManager scene = new TitledManager(8, 9, tiles, 3, 3);

...

// create scenes
void buildScene(int gamescene)
{
    for (int row=0; row<9; row++)
    {

```




```

    for (int col=0; col<8; col++)
    {
        scene.setCell(col, row, scene[gameScene][row][col]);
    }
}

// the paint method of the GameCanvas
void paint(Graphics g)
{
    scene.paint(g);
    turtle.paint(g);
}

```

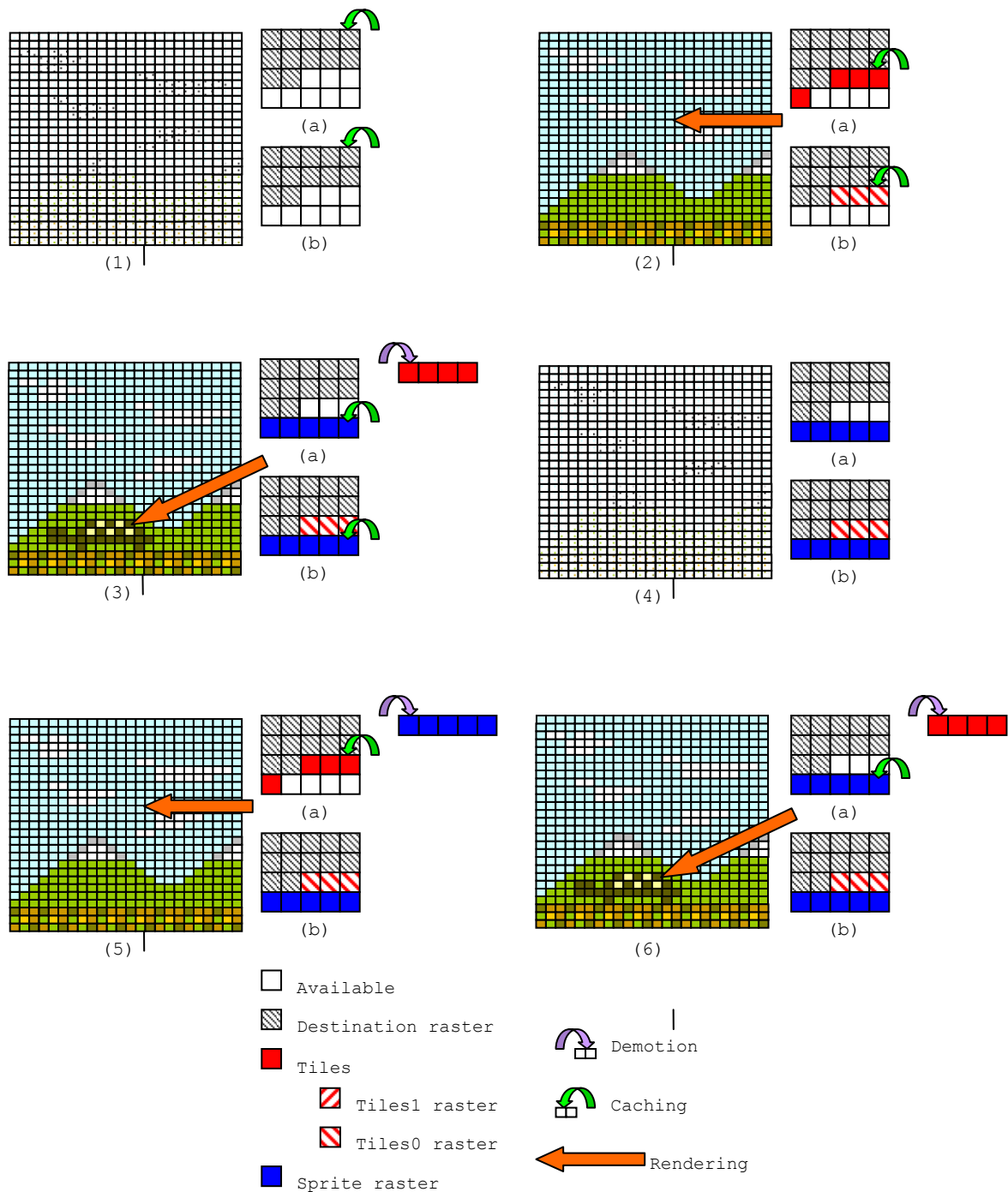
Let us discuss how the VRAM is used when the sprites are used. The sequence of figures below shows how the VRAM gets populated. See Figure 4.16.

```

// paint 1                                paint 2
// time -|----|----|----/\|----|----|----|---->
scene.paint(g); //      (1), (2),              (4), (5),
turtle.paint(g); //              (3),              (6)

```

The numbers between parentheses correlate with the state of the VRAM at every point in time. Each one represents a snapshot of the VRAM as if that VRAM were visible on the screen as a full frame.

**Figure 4.16**

Frame (1), (2) and (3) correspond to the first paint (first iteration) while frames (4), (5), (6) correspond to the second time paint is called (second iteration).



In frame (1a), the platform caches the destination raster. In frame (2a), the platform caches the raster that contains all tiles to construct the background and renders it. In frame (3a), the platform attempts to cache the raster that contains the character sprite and since there is not enough VRAM available, the raster with the background tiles gets demoted and the sprite raster gets cached. After the demotion and caching, the character is rendered.

During the second iteration of the paint method, the behavior is slightly different. In frame (4a), the destination raster won't be cached since it was cached during the first paint. Then, in frame (5a), the caching of the raster with the background tiles will force the demotion of the sprite with the character before the background is rendered. Finally, in frame (6a), the raster with the background tiles will be demoted to open space for the character sprite to be cached and rendered.

As you can observe, the platform did not take advantage of acceleration since each render required a cache operation. Acceleration is only efficient for drawing images when destination and source rasters have both already been cached into VRAM as a result of a previous render operation. If a source raster is cached and immediately demoted, the acceleration capability is defeated. In fact, performance may actually degrade in this situation.

The following code example demonstrates subtle changes to the source code that could allow efficient graphics acceleration based on the assumption that VRAM is used more efficiently. The changes are in bold letters. Frame sequence with lower case letter (b) demonstrates how the VRAM is being used.

```
// create the sprite
Sprite turtle = new Sprite("/turtle.png", 11, 4); // the turtle is 11x4
pixels

// matrices to compose both scenes backgrounds
static int[][][] scenes =
{
    { /* the mountains */
        { 1, 1, 4, 4, 4, 4, 4, 4 },
        { 4, 2, 3, 4, 1, 4, 4, 4 },
        { 4, 1, 4, 4, 4, 2, 2, 3 },
        { 4, 4, 2, 3, 4, 4, 4, 4 },
        { 4, 4, 4, 4, 1, 2, 3, 4 },
        { 4, 4, 5, 10, 4, 4, 4, 5 },
        { 4, 6, 8, 8, 7, 4, 6, 8 },
        { 6, 8, 8, 8, 8, 8, 8, 8 },
        { 9, 9, 9, 9, 9, 9, 9, 9 }
    },
    { /* the beach */
        { 4, 4, 4, 4, 4, 4, 4, 4 },
        { 1, 2, 1, 4, 2, 3, 4, 4 },
    }
}
```



```

        { 4, 4, 2, 1, 4, 2, 2, 3 },
        { 4, 4, 2, 3, 4, 4, 4, 4 },
        { 4, 4, 4, 4, 1, 2, 3, 4 },
        { 4, 4, 4, 4, 4, 4, 4, 4 },
        { 4, 4, 4, 4, 4, 4, 4, 4 },
        { 5, 6, 5, 9, 10, 7, 8, 7 },
        { 5, 6, 5, 10, 7, 8, 7, 10 }

    },
};

...

// create scenes
void buildScene(int gamescene)
{
    // create the tiles used for the scenes. Each tile is 3x3 pixels
    // there are 2 imgs: tiles0.png and tiles1.png. The first, tiles0.png
    // contains the tiles of scene 1 while tiles1.png contains the tiles
    // of scene 2
    Image tiles = Image.createImage("/tiles" + gamescene + ".png");

    // create scene (both scenes are same size so we
    // can use same TiledManager)
    TiledManager scene = new TitledManager(8, 9, tiles, 3, 3);

    for (int row=0; row<9; row++)
    {
        for (int col=0; col<8; col++)
        {
            scene.setCell(col, row, scene[gamescene][row][col]);
        }
    }
}

// the paint method of the GameCanvas

```





```
void paint(Graphics g)
{
    scene.paint(g);
    turtle.paint(g);
}
```

Even a better and more elegant change would be to use two TiledManagers: one for the sky tiles and another for the ground tiles. By doing this, tiles.png would be divided into three different .png files: skytiles.png, beachtiles.png and mountaintiles.png. But, we will leave that as an exercise for the reader. Let us now discuss the second scenario.

From figure 4.16, frame (1b), the platform caches the destination raster. In frame (2b), the platform caches and renders the background tiles. In frame (3b), the platform caches and renders the character sprite. During the second iteration of the paint method, the behavior is different. In frame (4b), the destination raster is not cached since it was cached during the first paint. Then, in frame (5b), the raster with the background tiles experiences full acceleration because it is cached already. Finally, in frame (6b), the character sprite experiences full acceleration also because there is no caching or demotion overhead. You can observe that the platform did take advantage of caching since it was always able to render from VRAM to VRAM any of the source rasters without repeatedly paying the cost of caching.

How much memory is there for caching? The following table shows how much VRAM is available per product. All numbers are approximate and in pixels. The backbuffer is assumed to be of the screen size and it supposed to be cached in VRAM during the first render operation that assigns it as the destination raster.

Product	Total VRAM available for rasters	Backbuffer (width x height = size). Does not include status bar.	Available VRAM for caching rasters (after caching backbuffer)
	51000	176 x 206* = 36256	14744
	673848	176 x 206* = 36256	637,592

*The height is 14 pixels smaller than the screen because of the status bar that indicates the signal strength and battery.

How much memory does a raster use? To calculate the amount of VRAM a raster takes simply multiply the width in pixels by the height in pixels. The available memory can be determined by subtracting the size of a raster from the total space. Finally, the last thing the developer must be aware of is that the platform has no way to coalesce empty VRAM blocks. In other words, the platform has no mechanism to avoid VRAM fragmentation.

Unfortunately, there is no way to monitor when a raster gets cached or demoted at run time. There is no way to know how much memory is available for caching and there is no way to



programmatically control the caching or demotion of a raster. In other words, there is no way prevent a raster from being cached or demoted. The only way available to evaluate the results of the graphics acceleration is to run the application in auto mode. Developers can observe the performance after pausing and resuming the application when the graphics accelerator is rendering the most intensive graphics screens.



4.9 Micro3D API

4.9.1 Overview



This API is only available on these handsets.

The Micro3D API allows developers to render 3D graphics. Content developers can use standard, commercial tools to create 3D content then convert that content to the Micro3D format. The Micro3D API supports Micro3D version 3.0, which provides the following capabilities:

- Translation, scaling, and rotation manipulation of 3D models
- Bone animation
- Rendering of primitives (triangles, quadrangles, lines, and points)
- Point sprite rendering
- White ambient lighting
- White, one-directional lighting
- Flat, Gouraud, and toon shading
- Environment mapping with textures
- Semi-transparency effects
- Parallel and perspective projection
- Drawing of multiple figures with Z-sorting
- Multiple textures

Developers should already have knowledge about basic 3D principles (transformations, viewpoints, textures, etc.) as this guide does not give detailed explanations of these concepts. This document also does not provide details about the creation of 3D model, texture, or action data. For more information on content creation consult the Micro3D Tool Manual.



4.9.2 The Micro3D Package

The Micro3D API is located in package `com.motorola.iden.micro3d`.

Below is the class hierarchy for the Micro3D package:

```
java.lang.Object
|
+ - com.motorola.iden.micro3d.ActionTable
|
+ - com.motorola.iden.micro3d.AffineTransform
|
+ - com.motorola.iden.micro3d.Layout3D
|
+ - com.motorola.iden.micro3d.Light
|
+ - com.motorola.iden.micro3d.Object3D
|   |
|   + - com.motorola.iden.micro3d.Figure
|   |
|   + - com.motorola.iden.micro3d.Primitive
|       |
|       + - com.motorola.iden.micro3d.Line
|       |
|       + - com.motorola.iden.micro3d.Point
|       |
|       + - com.motorola.iden.micro3d.PointSprite
|       |
|       + - com.motorola.iden.micro3d.Quadrangle
|       |
|       + - com.motorola.iden.micro3d.Triangle
|
+ - com.motorola.iden.micro3d.Renderer
|
+ - com.motorola.iden.micro3d.Texture
|   |
|   + - com.motorola.iden.micro3d.MultiTexture
|
+ - com.motorola.iden.micro3d.Utility
|
+ - com.motorola.iden.micro3d.Vector3D
```

The following sections describe the classes in the Micro3D API and outline how a simple 3D MIDlet could be created. Also included are instructions on using the Micro3D API with other J2ME™ classes to optimize rendering and enable animation.



4.9.3 Working with Graphics and Animation

Before we start using the Micro3D classes we'll need to be able to paint to a Graphics object and launch a Thread that will handle animating our 3D frames. We'll design our MIDlet with this in mind and create a class that will serve as a framework in which to place the rest of our 3D handling logic.

Getting 3D graphics displayed on the screen is very similar to drawing lines, rectangles, and Images in that you need a Graphics object to serve as the drawing target. While any mutable Graphics object will do, we'll use the GameCanvas to derive a Graphics instance. Since 3D rendering is processor intensive, the GameCanvas paradigm will help us run as efficiently as possible because we won't need to rely on repaint requests to get the Graphics object flushed to the screen.

We also want to be able to animate our 3D model. To this end, we'll need to be able to run an animation thread. Again, this approach is no different than how we might handle 2D animation.

Since GameCanvas is an abstract class we'll create a new class that extends it and implements Runnable so we can handle rendering, flushing, and animation all in one class. The following code snippet does just that:

```
public class GameCanvas3D extends GameCanvas implements Runnable {
    private Graphics myGraphics;

    private boolean paused;

    public GameCanvas3D() {
        super(true);
        paused = false;

        myGraphics = getGraphics();
    }

    public void paint(Graphics g) {
        g.setColor(0xFFFFFFFF);
        g.fillRect(0, 0, getWidth(), getHeight());
    }

    public void run() {
        while (!paused) {
            paint(myGraphics);
            flushGraphics();
        }
    }

    public synchronized void startAnimation() {
        paused = false;
        Thread t = new Thread(this);
        t.start();
    }

    public synchronized void stopAnimation() {
        paused = true;
    }
}
```



```
    }
}
```

We have kept things relatively simple so far. A few additions have been made that are worth noting, however. As previously mentioned, we need a Graphics object to draw to. We're using a GameCanvas so we can get the Graphics object from it. The myGraphics object holds the Graphics instance belonging to the instance of GameCanvas3D. Also added was the paused Boolean, which helps us start and stop the running animation thread; all we have to do is call `startAnimation()` when we want to begin our animation and `stopAnimation()` when we want to end it. Even at this early stage we have a class that is able to launch an animation thread that will paint and flush frames to the screen. Right now nothing useful is rendered but after we add a few simple 3D calls we'll have 3D graphics on the screen.

4.9.4 Creating a Figure

The Micro3D engine is capable of rendering Figure data and certain primitives. The Java Micro3D API has combined these concepts into an object-oriented structure allowing developers to use common methods for setting up figures and primitives and rendering them.

In order to provide this object-oriented structure the Object3D class was created as the parent class of all objects that can be rendered by the engine. This abstract class contains a set of methods common to *most* of its subclasses. Methods available with the Object3D class are shown in the following table:

Layout3D	getLayout()	Returns the Layout3D used for rendering this Object3D, or null if no Layout3D has been associated with this Object3D.
Texture	getSphereTexture()	Returns the Texture used for environmental texture mapping with this Object3D or null if no sphere texture has been associated with this Object3D.
Texture	getTexture()	Returns the Texture used for rendering this Object3D, or null if no Texture has been set associated with this Object3D.
void	setLayout(Layout3D layout)	Specifies a Layout3D to be used when rendering this Object3D.
void	setSphereTexture(Texture sphereTexture)	Sets the specified parameter as a sphere texture for this Object3D.
void	setTexture(Texture texture)	Associates a Texture with this Object3D.

However, some Object3D subclasses cannot support all of these methods, and although the methods may be callable, they will have no effect. For example, the `setTexture()` and `setSphereTexture()` methods are supported by the Figure subclass. Some subclasses of the Object3D subclass, Primitive, such as Point and Line, also inherit these methods but do not support them. Setting a texture or sphere texture to either will have no effect on their rendering. Always consult the Javadocs for method information specific to the Object3D subclasses. We'll go over primitives in more detail in a later section. For our example we want to use a model that was created on the PC. The model was previously converted into an mbac file that we'll package in our MIDlet's JAR file. 3D models are represented in the Micro3D API by the Figure class. There are two methods for creating Figure objects:



```
static Figure createFigure(byte[] data, int offset, int length)
    Creates a Figure which is decoded from the data stored in the specified byte array at the
    specified offset and length.

static Figure createFigure(String name)
    Creates a Figure from the data obtained from the named resource.
```

Both methods will return a Figure that can be rendered on-screen. In most cases, the `createFigure(String)` method is easiest to use, but downloaded Figure data can be loaded with `createFigure(byte[], int, int)`.

At this point we can add just a few lines of code to bring the Figure into our application. Code previously introduced is printed in regular face, new code is in bold face.

```
public class GameCanvas3D extends GameCanvas implements Runnable {
    private Graphics myGraphics;
    private Figure myFigure;

    private boolean paused;

    public GameCanvas3D() {
        super(true);
        paused = false;

        myGraphics = getGraphics();

        //Now we set up the actors in our 3d scene
        try {
            myFigure = Figure.createFigure("/figure_data.mbac");
        }
        catch (Exception e) {
            System.err.println("Error loading resources: " + e);
        }
    }

    ...
}
```

The `createFigure(String)` method only declares `IOException` and `NullPointerException` as thrown but it's possible that we could encounter other exceptions while loading the data. Since 3D data can be memory intensive, we want to be especially mindful of a possible `OutOfMemoryException` and be able to handle that.

The Figure class also contains methods that allow developers to animate the model. Those methods are shown in the following table:

<code>int</code>	<code>getActionIndex()</code>	Returns the index of the action this Figure is using for animation.
<code>ActionTable</code>	<code>getActionTable()</code>	Returns the ActionTable this Figure is using or null if no ActionTable has been set associated with this Figure.
<code>int</code>	<code>getFrameIndex()</code>	Returns the frame index value this Figure is using for animation.



```

int getNumberOfPatterns ()
    Returns the number of external appearance states for a Figure.

int getPattern ()
    Returns the pattern index to which this Figure is set.

void setActionTable (ActionTable actionTable)
    Sets the ActionTable for this Figure to use.

void setPattern (int pattern)
    Modifies the appearance of the figure by setting the pattern used during rendering.

void setPosture (int actionIndex, int frameIndex)
    Sets the posture for the Figure.

```

All of these methods change how the figure looks when rendered. The `getPattern()` and `setPattern(int)` methods allow you to switch between model patterns (vertex positions) defined within the figure's data. It's important to note that the value passed to `setPattern(int)` is a 32-bit integer mask, so it's possible to have multiple patterns rendered at once. The remaining methods work with action tables, which we'll cover in more detail later.

There's still more work to do with the Figure; right now it's just loaded but not being rendered. To get to that point requires that we use other classes in the Micro3D API.

4.9.5 Loading and Using Textures

Textures are BMP files the 3D engine uses when rendering and give 3D models their "skin". They can also be applied to certain primitives. Some textures can also be used to apply environment mapping to objects in your 3D scene, like a glossy surface for a car. The Micro3D engine also supports multiple textures for 3D models and even allows you to render your 3D scene to a texture. These "target textures" can then be applied to other objects in a 3D scene, for example, a mirror.

All of these different kinds of textures are supported by two classes: `Texture`, and its subclass, `MultiTexture`. The `Texture` class can represent regular textures and sphere textures. Its methods are shown in the following table:

```

static Texture createTexture (byte[] data, int offset, int length,
    boolean sphereTexture)
    Creates a Texture which is decoded from the data stored in the specified byte array at the
    specified offset and length.

static Texture createTexture (String name, boolean sphereTexture)
    Creates a Texture from the data obtained from the named resource.

boolean isSphereTexture ()
    Returns a Boolean indicating if the texture data is to be used for environment mapping.

```

Texture objects are created by the two static methods, `createTexture(byte[], int, int, boolean)` and `createTexture(String, boolean)`. The first method can load texture data already contained in a byte array, while the second can load data from the resource named by the string provided. The sphereTexture Boolean notifies the Micro3D engine whether the texture is a sphere texture. The engine does treat regular textures differently than sphere textures so methods



that call for regular textures will not take Texture objects whose `isSphereTexture()` method returns true and vice versa.

As mentioned, textures are BMP files. The files must be uncompressed with 8-bit, indexed color. Regular textures can be up to 256x256 pixels while sphere textures can be up to 64x64 pixels. Attempting to load textures that exceed the expected dimensions will cause the create methods to throw an `IllegalArgumentException`. It's important to note that all instances of `Texture` (except the subclass `MultiTexture`) take up around 82 KB of memory, regardless of actual size of the texture.

Target textures are also regular textures. The Micro3D engine can render a scene to any instance of `Texture`, except the defined subclass, `MultiTexture`. Once a scene has been rendered to an instance of `Texture`, its `isSphereTexture()` method will return false. Target textures are always 256x256 pixels in size.

The only defined subclass of `Texture` is `MultiTexture`. Some 3D models are created such that multiple textures are required to fully render them. The `MultiTexture` class is a subclass of `Texture`, but acts more like a wrapper for the `java.lang.util` class `Vector`, specifically holding only `Texture` objects. The following field value defines the maximum amount of textures that can be held, currently equal to 16:

```
static int MAX_TEXTURES
```

Defines the maximum amount of textures a `MultiTexture` can hold.

The `MultiTexture` class does not allocate memory for the texture – it only contains references to `Texture` objects - so `MultiTexture` objects are as lightweight as most other objects. Unlike `Texture`, `MultiTexture` has two constructors:

```
MultiTexture()
```

Constructs an empty `MultiTexture`.

```
MultiTexture(int capacity)
```

Constructs a `MultiTexture` capable of holding the specified amount of `Texture` objects.

Developers can use the default constructor, which automatically sets a capacity equal to `MAX_TEXTURES` or can specify the capacity with the other constructor.

A 3D model set to use a `MultiTexture` will use only as many of the textures being held in the `MultiTexture` as it needs, up to the current amount of textures contained. For example, if a `MultiTexture` is holding 16 textures, and a 3D model using that `MultiTexture` only needs two textures then only the `Texture` objects at position 0 and 1 in the `MultiTexture` will be used. If the `MultiTexture` only had one texture, then only that `Texture` at position 0 would be used causing the model to render incorrectly. `MultiTexture` cannot hold sphere textures.

Since `MultiTexture` is based on `Vector`, the methods should be familiar to most Java developers. The table below lists the methods in `MultiTexture`:

```
boolean addTexture(Texture texture)
    Adds the specified Texture to the end of the list.

int capacity()
    Returns the current capacity of this MultiTexture.

boolean contains(Texture texture)
    Tests if the specified Texture is contained in this MultiTexture.
```



```

int getCurrentIndex()
    Returns the current index for this MultiTexture.

Texture getCurrentTexture()
    Returns the Texture reference by the current index.

int indexOf(Texture texture)
    Searches for the first occurrence of the given Texture, testing for equality using the equals
    method.

int indexOf(Texture texture, int index)
    Searches for the first occurrence of the given Texture, beginning the search at index, and
    testing for equality using the equals() method.

boolean insertTextureAt(Texture texture, int index)
    Inserts the specified texture as a component in this MultiTexture at the specified index.

boolean isEmpty()
    Tests if this MultiTexture has no textures.

int lastIndexOf(Texture texture)
    Returns the index of the last occurrence of the specified texture in this MultiTexture.

int lastIndexOf(Texture texture, int index)
    Searches backwards for the specified texture, starting from the specified index, and returns
    an index to it.

void removeAllTextures()
    Removes all components from this MultiTexture and sets its size to zero.

boolean removeTexture(Texture texture)
    Removes the first occurrence of the specified texture from this MultiTexture.

void removeTextureAt(int index)
    Deletes the MultiTexture at the specified index.

void setCurrentIndex(int index)
    Sets the Texture at the index specified as the current Texture for this MultiTexture.

void setTextureAt(Texture texture, int index)
    Sets the texture at the specified index of this MultiTexture to be the specified Texture.

int size()
    Returns the number of textures in this MultiTexture.

Texture textureAt(int index)
    Returns the Texture at the specified index.

Enumeration textures()
    Returns an enumeration of the textures of this MultiTexture.

```

This document does not go into detail about all of the methods since most of them are explained by the Vector class and by the Javadocs for the Micro3D API. Note that the word “element” in the Vector methods is replaced by the word “texture” for MultiTexture methods, so the method `elements()` in Vector is called `textures()` here, but performs the same functionality.

The additional methods here are the `setCurrentIndex(int)`, `getCurrentIndex()`, and `getCurrentTexture()` methods. MultiTexture designates one of the textures it is holding as the current texture. The current texture is -1 by default. If the MultiTexture has no Texture objects the current texture is null while the current texture index remains at -1. The current texture index is validated as the amount of texture objects in the MultiTexture decreases. This means that if the



current texture index is 5 and the amount of textures in the MultiTexture changes from 10 to 4, then the current texture index is set to -1.

The current texture index is used for 3D objects that don't need multiple textures. Since a MultiTexture is a Texture, they can be used wherever a regular texture is called for. The Texture at the current index will be used in these cases. If the current index is -1 then the 3D object using the MultiTexture will essentially have no Texture associated with it.

Our example will use more than one texture, but for now we can just declare and load the regular texture that we'll use for the Figure we introduced in the previous section:

```
public class GameCanvas3D extends GameCanvas implements Runnable {
    private Graphics myGraphics;
    private Figure myFigure;
    private Texture myTexture;

    private boolean paused;

    public GameCanvas3D() {
        super(true);
        paused = false;

        myGraphics = getGraphics();

        //Now we set up the actors in our 3d scene
        try {
            myFigure = Figure.createFigure("/figure_data.mbac");
            myTexture =
                Texture.createTexture("/texture_data.bmp", false);
        }
        catch (IOException e) {
            System.err.println("Error loading resources: " + e);
        }
        myFigure.setTexture(myTexture);
    }
    ...
}
```

Since we now have the Texture loaded, we're able to revisit the figure and assign it the newly loaded Texture. In later sections we'll get into target textures and regular textures that support transparency. First, we'll cover the other 3D objects that we'll apply these effects to by examining the primitive support offered by the Micro3D engine.

4.9.6 Working with Vector3D

Before we get into the Primitives that we'll add to the scene, we need to go over the Vector3D class. The Vector3D class is a simple class, representing a point in 3D space. Among other things, Vector3Ds are used for specifying positions and dimensions of Primitives and setting up viewpoints for a scene.

A Vector3D contains three integers; an X, Y, and a Z value. Remember that the Micro3D engine uses no floating-point math. To simulate greater precision the X, Y, and Z values range from -32,678 to 32,677. A value of 4,096 is considered to be equivalent to a 1.0.

This class contains methods that allow you to manipulate a Vector3D by computing the inner and outer products and normalizing the vector, and also provides simple set and get methods for specifying and retrieving the X, Y, and Z component values. We won't need to use those in our



demo, but more interactive applications may need to perform additional vector calculations. Here they are for reference:

```
int getX()
    Returns the x value of this Vector3D object.

int getY()
    Returns the y value of this Vector3D object.

int getZ()
    Returns the z value of this Vector3D object.

int innerProduct(Vector3D multiplier)
    Returns the inner, or dot, product of this Vector3D with the specified Vector3D.

void normalize()
    Normalizes this Vector3D object.

static Vector3D normalize(Vector3D vector)
    Normalizes the specified Vector3D object.

void outerProduct(Vector3D multiplier)
    Calculates the outer, or cross, product of this Vector3D with the specified Vector3D.

static Vector3D outerProduct(Vector3D multiplicand, Vector3D multiplier)
    Calculates the outer, or cross, product of the specified Vector3D objects.

void set(int x, int y, int z)
    Sets the x, y, and z values of this Vector3D object.

void setX(int x)
    Sets the x value of this Vector3D object.

void setY(int y)
    Sets the y value of this Vector3D object.

void setZ(int z)
    Sets the z value of this Vector3D object.
```

For our purposes we'll just create our Vector3D instances with the numbers we want to use for our primitives. The Vector3D constructor can take no arguments, giving a <0, 0, 0> vector or can take the X, Y, and Z values as arguments as follows:

```
Vector3D()
    Creates a Vector3D object with x, y, and z values set to 0.

Vector3D(int x, int y, int z)
    Creates a Vector3D object with the specified x, y, and z values.
```

In the next section we'll be using the Vector3D constructors to set up our primitives, and later to work with layouts and viewpoints.



4.9.7 Creating Primitives

Now that we know how to use the `Vector3D` class, we can set up our primitives. The `Primitive` class extends the abstract `Object3D` class, and is also abstract. The primitives supported by the Micro3D engine are point (class `Point`), line (class `Line`), triangular polygon (class `Triangle`), quadrilateral polygon (class `Quadrangle`), and point sprite (class `PointSprite`).

The `Primitive` class supplies its subclasses with methods to set up the `Primitive` in 3D space and control its appearance.

Since the different `Primitive` types have differing amounts of vertices and normals, a generic method for setting these vectors is provided in the `Primitive` class. Specific vertices and normals are specified with constants defined in the `Primitive` class. The following constants are defined for this purpose:

```
static int FACE_NORMAL
    Identifier for a vector that is serving as a face normal.

static int NORMAL
    Identifier for a vector that is serving as a vertex normal.

static int VERTEX_A
    Identifier for vertex A.

static int VERTEX_B
    Identifier for vertex B.

static int VERTEX_C
    Identifier for vertex C.

static int VERTEX_D
    Identifier for vertex D.
```

These constants are used in defining the vector ID value passed to the `Primitive` methods below:

```
abstract Vector3D getVector(int vectorID)
    Returns the Vector3D for the specified vector.

abstract void setVector(int vectorID, Vector3D vector)
    Sets the specified vector to the Vector3D provided.
```

When setting a vertex's `Vector3D` for a `Primitive`, one of the `VERTEX_A`, `VERTEX_B`, `VERTEX_C`, or `VERTEX_D` constants needs to be used as the `vectorID` value. If the `Primitive` being defined is one of the `Triangle` or `Quadrangle` subclasses then normal vectors can also be defined. The Micro3D engine allows normals to be defined per vertex or for the face of the `Primitive`. When specifying normals per vertex, the vertex type constants must be combined with the constant `NORMAL`. If a face normal is being specified then only the `FACE_NORMAL` constant is needed as the `vectorID` value.

Primitives can also be colored as desired. Colors are specified by the familiar Graphics convention, `0xRRGGBB`. The `Primitive` class provides a method for setting and getting a primitive's color:

```
int getColor()
    Returns the color that is used for rendering this Primitive.

void setColor(int color)
    Sets the color that should be used when rendering a Primitive.
```



Primitives can also show color blending effects. Color blending takes effect when the Primitive is semi-transparent and allows the color of the Primitive to be combined with the background. Colors can be blended in different ways. The Primitive class defines constants that are used to define what kind of blending should take place:

```
static int BLENDING_ADD
    Blending type - additive blending (dest 100%+src 100%).

static int BLENDING_HALF
    Blending type - 50%.

static int BLENDING_NONE
    Blending type - none.

static int BLENDING_SUB
    Blending type - subtractive blending (dest 100%-src 100%).
```

There are also two simple methods for setting and getting the effect type:

```
int getBlendingType()
    Returns the blending type.

void setBlendingType(int blendingType)
    Sets the blending type.
```

Let's go ahead and create two simple primitives: Point and Line. These Primitives have no surface area in 3D space and therefore only support the Primitive methods we've covered so far. We can set all the values necessary for each Object in the constructor so we'll only need to add a few extra lines of code to our example:

```
public class GameCanvas3D extends GameCanvas implements Runnable {
    private Graphics myGraphics;
    private Point myPoint;
    private Line myLine;
    private Figure myFigure;
    private Texture myTexture;

    private boolean paused;

    public GameCanvas3D() {
        super(true);
        paused = false;

        myGraphics = getGraphics();

        //Now we set up the actors in our 3d scene
        try {
            myFigure = Figure.createFigure("/figure_data.mbac");
            myTexture =
                Texture.createTexture("/texture_data.bmp", false);
        }
        catch (IOException e) {
            System.err.println("Error loading resources: " + e);
        }
    }
}
```



```

    }
    myPoint = new Point(new Vector3D(-10, 10, 0),
                        null, 0xFF0000);
    myLine = new Line(new Vector3D(-10, 10, 0),
                     new Vector3D(50, 0, 0),
                     null, 0x00FF00);
    myFigure.setTexture(myTexture);
}
...
}

```

Since we haven't learned about scene layouts yet we can just pass in null for the Layout3D required for the constructors. We'll come back and revisit that in a later section.

The Primitive class also has a few more methods supported by other subclasses to enable color key transparency effects for textured polygons:

```

void enableColorKeyTransparency(boolean enable)
    Enables/disables color key transparency.

boolean hasColorKeyTransparency()
    Determines whether the primitive has color key transparency enabled.

```

Color key transparency uses the texture's color lookup table (CLUT) to determine which pixels of the texture will be rendered. When the color key transparency is enabled, the CLUT0 color in the polygon texture will be transparent. All other colors within the polygon texture will not be transparent. The CLUT0 color is the first entry in the color lookup table. It can be any color, but it is recommended that it is specified to be black (RGB 0x000000). The following pictures show a checkerboard texture that has CLUT0 set to black. The image on the left shows how the texture will be rendered when the color key transparency is disabled. The image on the right shows how the same texture will be rendered when the color key transparency is enabled.



Figure 4.9 Color Key Transparency

Since color key transparency is only applicable to textures, only the Triangle, Quadrangle, and PointSprite classes can support it. We'll add one of each of those classes to our example. The Triangle will have color blending, the Quadrangle will have a texture with color key transparency enabled, and the PointSprite will have two normal textures that we'll use for animation. Going back to our example we add the lines to create the last of the Primitives and the textures they will use:

```

public class GameCanvas3D extends GameCanvas implements Runnable {
    private Graphics myGraphics;

```



```

private Point myPoint;
private Line myLine;
private Triangle myTriangle;
private Quadrangle myQuadrangle;
private PointSprite myPointSprite;
private Figure myFigure;
private Texture myTexture;
private Texture myTransparentTexture;
private Texture mySpriteTexture;

private boolean paused;

public GameCanvas3D() {
    super(true);
    paused = false;

    myGraphics = getGraphics();

    //Now we set up the actors in our 3d scene
    try {
        myFigure = Figure.createFigure("/figure_data.mbac");
        myTexture = Texture.createTexture(
            "/texture_data.bmp", false);
        myTransparentTexture =
            Texture.createTexture("/trans_texture_data.bmp",
                false);
        mySpriteTexture =
            Texture.createTexture("/sprite_texture_data.bmp",
                false);
    } catch (IOException e) {
        System.err.println("Error loading resources: " + e);
    }
    myPoint = new Point(new Vector3D(-10, 10, 0),
        null, 0xFF0000);
    myLine = new Line(new Vector3D(-10, 10, 0),
        new Vector3D(50, 0, 0),
        null, 0x00FF00);
    myTriangle = new Triangle(new Vector3D(0, 0, 0),
        new Vector3D(0, 35, 0),
        new Vector3D(35, 0, 0), null,
        0x0000FF);
    myQuadrangle = new Quadrangle(new Vector3D(0, 0, 0),
        new Vector3D(10, 35, 0),
        new Vector3D(45, 35, 0),
        new Vector3D(35, 0, 0),
        0, 0, 20, 0, 20, 20, 0, 20,
        null, myTransparentTexture);
    myQuadrangle.enableColorKeyTransparency(true);
    myPointSprite = new PointSprite(new Vector3D(0, 0, 0),
        30, 30, 0, 0, 0, 30, 30,
        PointSprite.PIXEL_SIZE |
        PointSprite.PARALLEL_PROJECTION,
        null, mySpriteTexture);

```



```
myFigure.setTexture(myTexture);
}
```

We went ahead and enabled color key transparency for the quadrangle so we'll see some transparency when we render the scene.

Since the `Triangle`, `Quadrangle`, and `PointSprite` classes are the only Primitive subclasses that support textures, they have a few extra methods for dealing with them:

```
int getTextureCoordinateX(int vertexID)
    Returns the X component of the texture coordinate for the specified vertex.

int getTextureCoordinateY(int vertexID)
    Returns the Y component of the texture coordinate for the specified vertex.

void setTextureCoordinates(int vertexID, int x, int y)
    Sets the texture coordinates for the specified vertex.
```

These methods let you get and set portions of the texture that you want to use when rendering a primitive. Specifying a `vertexID` value is done just as it is done for the `setVector(int, Vector3D)` method mentioned earlier. Valid IDs are `VERTEX_A`, `VERTEX_B`, `VERTEX_C`, or `VERTEX_D`. Consult the Javadocs for each Primitive subclass to see which ID values are valid for that particular class. In addition to those methods, `PointSprite` defines a few more for setting its height, width, rotation, and display type:

```
int getDisplayType()
    Returns the display directive value for this PointSprite.

int getHeight()
    Returns the height of the PointSprite.

int getRotation()
    Returns the rotation of the PointSprite.

int getWidth()
    Returns the width of the PointSprite.

void setDisplayType(int displayType)
    Specifies whether the PointSprite's size is relative to the screen or model coordinate system and enables perspective projection on the PointSprite.

void setHeight(int height)
    Sets the height of the PointSprite.

void setRotation(int rotation)
    Sets the rotation angle of the PointSprite.

void setWidth(int width)
    Sets the width of the PointSprite.
```

`PointSprite`'s width and height are set explicitly instead of using multiple vertex values as with other primitives. The `getHeight()`, `setHeight(int)`, `getWidth()`, and `setWidth(int)` methods provide height and width support. A `PointSprite` can also be arbitrarily rotated with the `setRotation(int)` method. The angle value ranges from 0 to 4,096, where 4,096 is equivalent to 360 degrees.



The display type defines how the PointSprite should be rendered. The following constants should be used when specifying a display type value with `setDisplayType(int)`:

```
static int LOCAL_SIZE
    Specifies that a sprite's dimensions are in terms of the model coordinate system.

static int PARALLEL_PROJECTION
    Disables perspective projection when rendering a sprite.

static int PERSPECTIVE_PROJECTION
    Enables perspective projection when rendering a sprite.

static int PIXEL_SIZE
    Specifies that a sprite's dimensions are in terms of the screen coordinate system.
```

A PointSprite's size can be relative to the screen's coordinate system, or the model's coordinate system. In addition to the PointSprite's size relation, parallel or perspective projections can be applied to the PointSprite. The `PARALLEL_PROJECTION` and `PERSPECTIVE_PROJECTION` constants define both projection types. When passing a value to `setDisplayType(int)`, a projection type must be combined with one of the size constants. In our example we created the PointSprite with a display type of `PIXEL_SIZE | PARALLEL_PROJECTION`. Refer to the PointSprite Javadocs for descriptions of the possible combinations.

At this point we can take care of the point sprite's animation by adding some code to the `run()` method we created earlier. We'll move the texture coordinates around a bit and spin the PointSprite around:

```
public void run() {
    while (!paused) {
        if (myPointSprite.getTextureCoordinateY(Primitive.VERTEX_A) == 0)
            myPointSprite.setTextureCoordinates(Primitive.VERTEX_A,
                                                0, 30);
        else
            myPointSprite.setTextureCoordinates(Primitive.VERTEX_A,
                                                0, 0);
        myPointSprite.setRotation((myPointSprite.getRotation()+1024) %
                                   4096);
        paint(myGraphics);
        flushGraphics();
    }
}
```

The `setTextureCoordinates(int, int, int)` calls will cycle the PointSprite through the two animation frames contained in its one texture. The `setRotation(int)` call will make the PointSprite look like it is spinning.



4.9.8 Loading and Using Action Tables

The `ActionTable` class represents 3D action tables, which define how a figure can be animated. The data stored in an action table represents key-based animation. These actions can define how a person walks, a car door opens, etc.

An `ActionTable` can hold multiple actions and each action can have multiple frames. An instance of `ActionTable` has at least 1 action. Frames for each action are numbered in increments of 65,536. So the 0th frame's index is 0 and the next frame's index is 65,536. The following methods are defined in `ActionTable`:

```
static ActionTable createActionTable(byte[] data, int offset, int length)
    Creates an ActionTable which is decoded from the data stored in the specified byte
    array at the specified offset and length.

static ActionTable createActionTable(String name)
    Creates an ActionTable from the data obtained from the named resource.

int getNumberOfActions()
    Obtains the number of actions found in the ActionTable.

int getNumberOfFrames(int actionIndex)
    Obtains the number of frames for the specified action found in the ActionTable.
```

Like `Figure` and `Texture`, there is no constructor in the `ActionTable` class. `ActionTable` instances are similarly created via the static `createActionTable(byte[], int, int)` and `createActionTable(String)` methods. The `getNumberOfActions()` method returns the total number of actions for this `ActionTable` while `getNumberOfFrames(int)` returns the number of frames for a given action index in 65,536 increments.

To animate our `Figure` we'll need to associate an `ActionTable` with it and continuously update the action frame it is using. We want the animation to look smooth so we'll tie in our frame switching with values we read off of the system clock. Going back to our example we can load the `ActionTable` and modify the `run()` and `startAnimation()` methods:

```
public class GameCanvas3D extends GameCanvas implements Runnable {
    private Graphics myGraphics;
    private Point myPoint;
    private Line myLine;
    private Triangle myTriangle;
    private Quadrangle myQuadrangle;
    private PointSprite myPointSprite;
    private Figure myFigure;
    private Texture myTexture;
    private Texture myTransparentTexture;
    private Texture mySpriteTexture;
    private ActionTable myActionTable;

    private boolean paused;
    private int frame;
    private int frameTime;
    private long oldTime;
    private int maxFrames;
```



```

public GameCanvas3D() {
    super(true);
    paused = false;
    frame = 0;
    frameTime = 0;

    myGraphics = getGraphics();

    //Now we set up the actors in our 3d scene
    try {
        myFigure = Figure.createFigure("/figure_data.mbac");
        myTexture =
            Texture.createTexture("/texture_data.bmp", false);
        myTransparentTexture =
            Texture.createTexture("/trans_texture_data.bmp",
                                false);
        mySpriteTexture =
            Texture.createTexture("/sprite_texture_data.bmp",
                                false);
        myActionTable =
            ActionTable.createActionTable("/action_data.mtra");
    } catch (IOException e) {
        System.err.println("Error loading resources: " + e);
    }
    maxFrames = myActionTable.getNumberOfFrames(0);
    myPoint = new Point(new Vector3D(-10, 10, 0),
                        null, 0xFF0000);
    myLine = new Line(new Vector3D(-10, 10, 0),
                     new Vector3D(50, 0, 0),
                     null, 0x00FF00);
    myTriangle = new Triangle(new Vector3D(0, 0, 0),
                             new Vector3D(0, 35, 0),
                             new Vector3D(35, 0, 0), null,
                             0x0000FF);
    myQuadrangle = new Quadrangle(new Vector3D(0, 0, 0),
                                  new Vector3D(10, 35, 0),
                                  new Vector3D(45, 35, 0),
                                  new Vector3D(35, 0, 0),
                                  0, 0, 20, 0, 20, 20, 0, 20,
                                  null, myTransparentTexture);
    myQuadrangle.enableColorKeyTransparency(true);
    myPointSprite = new PointSprite(new Vector3D(0, 0, 0),
                                    30, 30, 0, 0, 0, 30, 30,
                                    PointSprite.PIXEL_SIZE |
                                    PointSprite.PARALLEL_PROJECTION,
                                    null, mySpriteTexture);
    myFigure.setTexture(myTexture);
    myFigure.setActionTable(myActionTable);
    myFigure.setPosture(0, frame);
}

...

```




```

public void run() {
    while (!paused) {
        if (myPointSprite.getTextureCoordinateY(Primitive.VERTEX_A)
            == 0)
            myPointSprite.setTextureCoordinates(
                Primitive.VERTEX_A, 0, 30);
        else
            myPointSprite.setTextureCoordinates(
                Primitive.VERTEX_A, 0, 0);
        myPointSprite.setRotation(
            (myPointSprite.getRotation()+1024) % 4096);
myFigure.setPosture(0, frame);
        paint(myGraphics);
        flushGraphics();

        //Here we'll update the frame we want to draw.
        //Remember that ActionTable frames range from
        //0 to (max frames * 65536) so updating
        //the frame is not as simple as an increment.
        long time = System.currentTimeMillis();
        frameTime = (int) (time - oldTime);
        oldTime = time;

        //We'll do our frame changes every second
        //thus the divide by 1000
        //We multiply by 30 because we created the animation table
        //at a 30 fps rate, so for every second we want to make
        //sure we are at frame (65536*30*second).
        frame = frame + ((65536 * 30 * frameTime)/1000);
        frame = frame % maxFrames;
    }
}

public synchronized void startAnimation() {
    paused = false;
    Thread t = new Thread(this);

    //save the time here so we can compare when the thread starts
    oldTime = System.currentTimeMillis();
    t.start();
}
...

```

Here we introduced a variable to track which frame we're on (`frame`) and a variable that contains the amount of time elapsed since we rendered our last frame (`frameTime`). We also need to track the system time value for each frame we render (`oldTime`). System time is initially stored when our `startAnimation()` method is entered and we start the animation thread. From then on we check the elapsed time with every `run()` iteration and set the new frame as needed. Notice that we had to add an extra multiplication by 30 when calculating the new frame value. This is because the action table contains an animation based on a 30-frames-per-second rate. We get this number when we create the animation table, not during run-time, so it's important to have that number available when creating your 3D application. Also notice that we saved the maximum amount of frames for action



index 0 and used that stored value in our animation loop. This makes our code run faster since we don't need to query the engine for that value for every frame.

4.9.9 Setting the Scene: Light

The `Light` class is a container for the values that specify what kind of lighting a scene should have. The Micro3D engine supports both ambient and single, directional white lighting. Values for light intensity range from 0 to 4,096, with 4,096 being equivalent to a factor of 1.0. One instance of `Light` can be used to specify both directional and ambient lighting. The `Light` constructor allows developers to set any combination of lighting parameters they choose:

Light(int ambientIntensity, Vector3D direction, int directionalIntensity)
Creates a `Light` object with the specified parameters.

While ambient and directional lighting can be combined, it is possible to drown out directional light effects when full ambient lighting is present. Therefore, effective use of directional lighting requires careful use of ambient lighting. If directional lighting is not desired the direction vector can be set to null or the directional intensity can be set to 0. The former choice is more efficient since the engine can skip the calculations needed to resolve the effect of the light's directional vector with surfaces in the scene.

A `Light` instance's parameters can also be changed later or retrieved through the get and set methods provided in the `Light` class:

```
int getAmbientIntensity()
    Returns the ambient light intensity for this Light object.

int getDirectionalIntensity()
    Returns the directional light intensity for this Light object.

Vector3D getDirectionVector()
    Returns the Vector3D representing the direction of light for this Light object.

void setAmbientIntensity(int intensity)
    Sets the ambient light intensity for this Light object.

void setDirectionalIntensity(int intensity)
    Sets the directional light intensity for this Light object.

void setDirectionVector(Vector3D direction)
    Sets and enables or disables directional lighting for this Light object.
```

To specify the light's direction we'll need to add a new `Vector3D`. Vectors used for describing a light direction cannot be 0 vectors ($<0, 0, 0>$). Knowing that, we can go ahead and create the lighting that our scene will use in our example:

```
public class GameCanvas3D extends GameCanvas implements Runnable {
    private Graphics myGraphics;
    private Light myLight;
    private Point myPoint;
    private Line myLine;
    private Triangle myTriangle;
    private Quadrangle myQuadrangle;
    private PointSprite myPointSprite;
```



```
private Figure myFigure;
private Texture myTexture;
private Texture myTransparentTexture;
private Texture mySpriteTexture;
private ActionTable myActionTable;

private boolean paused;
private int frame;
private int frameTime;
private long oldTime;
private int maxFrames;

public GameCanvas3D() {
    super(true);
    paused = false;
    frame = 0;
    frameTime = 0;

    myGraphics = getGraphics();

    //Here we initialize the Layout3D, the lighting,
    //and the viewpoint for our scene
    Vector3D lightDirection = new Vector3D(-4096, 0, 0);
    myLight = new Light(2048, lightDirection, 4096);

    //Now we set up the actors in our 3d scene
    try {
        myFigure = Figure.createFigure("/figure_data.mbac");

        ...
    }
}
```

Later, we'll associate the Light instance we created with an object that specifies how the scene should be rendered.

4.9.10 Using Affine Transformations

Before we can set up our layout object we need to know how to work with affine transformations. Affine transformations are represented in the Micro3D API by the AffineTransform class. This class allows you to specify how 3D space should be interpreted with your local coordinate system.

The AffineTransform class serves mainly as a container to the set of 12 matrix values contained in an affine transformation. As with other classes, the AffineTransform class uses a larger scale to compensate for the lack of floating point math. Affine transformation values range from -32,768 to 32,767 with 4,096 representing a value of 1.0. AffineTransform defines constants for these maximum and minimum values, as well as constants for directly accessing the individual matrix elements:

```
static int M00
    Specifies matrix element at row 1, column 1.

static int M01
    Specifies matrix element at row 1, column 2.

static int M02
    Specifies matrix element at row 1, column 3.
```



```
static int M03
    Specifies matrix element at row 1, column 4.

static int M10
    Specifies matrix element at row 2, column 1.

static int M11
    Specifies matrix element at row 2, column 2.

static int M12
    Specifies matrix element at row 2, column 3.

static int M13
    Specifies matrix element at row 2, column 4.

static int M20
    Specifies matrix element at row 3, column 1.

static int M21
    Specifies matrix element at row 3, column 2.

static int M22
    Specifies matrix element at row 3, column 3.

static int M23
    Specifies matrix element at row 3, column 4.

static int MAX_VALUE
    The maximum value that can be used by the AffineTransform.

static int MIN_VALUE
    The minimum value that can be used by the AffineTransform.
```

An AffineTransform can be constructed with initial values set to 0 or with values specified in a two-dimensional integer array, as shown in the following table:

AffineTransform()

Constructs an AffineTransform with all of its elements set to 0.

AffineTransform(int[][] elements)

Constructs an AffineTransform with its elements set to the values in the specified two-dimensional array.

The array passed into the second constructor must be an array whose outer length is three while the lengths of the three arrays contained as elements are four.

The AffineTransform class has many convenience methods for matrix operations. The operations include matrix multiplication, normalization, rotation, creation of an identity matrix, and point translation:

```
int get(int fieldID)
    Gets the specified AffineTransform matrix element value.

static AffineTransform getViewPointTransform(Vector3D position,
    Vector3D look, Vector3D up)
    Calculates a viewpoint transformation matrix.
```



```

void multiply(AffineTransform multiplier)
    Multiplies this AffineTransform object against another.

static AffineTransform multiply(AffineTransform multiplicand,
    AffineTransform multiplier)
    Multiplies two AffineTransform objects against another.

static void multiply(AffineTransform destination,
    AffineTransform multiplicand,
    AffineTransform multiplier)
    Multiplies two AffineTransform objects and stores the result of the operation
    in the specified destination.

void normalize()
    Normalizes this AffineTransform.

void rotateV(Vector3D axis, int angle)
    Rotates this transform about an arbitrary unit vector.

void rotateX(int angle)
    Rotates this transform about the x-axis.

void rotateY(int angle)
    Rotates this transform about the y-axis.

void rotateZ(int angle)
    Rotates this transform about the z-axis.

void set(int[][] elements)
    Sets the AffineTransform matrix elements with the integers in the specified
    two-dimensional array.

void set(int fieldID, int value)
    Sets the specified AffineTransform matrix element with the specified value.

void setIdentity()
    Converts this AffineTransform to an identity matrix.

Vector3D transformPoint(Vector3D source)
    Returns a Vector3D transformed from a given point in this AffineTransform.

```

Note that matrix multiplication is offered in three methods. Each method provides a different mechanism for delivering the resultant AffineTransform. The transform can be stored in the multiplicand, in a given destination AffineTransform, or returned as a new AffineTransform instance. If matrix multiplication is to be done repeatedly, for example in an animation thread, it is recommended that developers avoid the static multiplication method to avoid heap fragmentation.

When setting up a 3D scene, a viewpoint transformation matrix is required. A viewpoint transformation can be created automatically by using the `getViewPointTransform(Vector3D, Vector3D, Vector3D)` method. This method takes the position, look, and up directional vectors to derive an equivalent viewpoint transformation matrix.

In the next section we'll learn how the Layout3D class uses the viewpoint transformation matrix.



4.9.11 Setting the Scene: Layout3D

In the previous sections we covered loading your 3D content data and preparing it for display. One of the final steps is to provide a description for how the scene should be rendered. The Layout3D class is provided for that reason.

The Layout3D class sets the stage of a 3D scene. With the Layout3D class developers can specify lighting, shading, projection style, and camera placement. Complex scenes can be developed by using multiple instances of Layout3D with several figures and primitives and combining them at render time. Of all the classes covered so far, Layout3D has the most impact on how a figure or primitive is rendered.

For example, consider a scene that calls for two orbs to be placed side by side with one lit by a directional light source and one dimmed and receiving little or no light. Although we have two orbs that have different appearances, we only need to create one Figure instance that represents the orb. The rest of the scene direction can be handled by the creation of two Layout3D instances and a Light instance. To set up our well lit orb we can associate one Figure with the Layout3D containing a powerful, ambient light then make another call to draw with the same Figure, but this time associated with the Layout3D that has no light, or a very dimmed directional light.

Since figures and textures consume the most memory, limiting the amount we need to instantiate allows us to create more complex scenes with less demand on memory.

Creating a Layout3D is done by invoking the default constructor:

Layout3D ()

Creates a Layout3D for use with rendering Figure and Primitive objects.

The constructor sets up the Layout3D with most options disabled: there is no lighting, no semi-transparent effects, no shading, and no perspective projection. As we'll see, the Layout3D is capable of producing more interesting scenes by turning on these effects.

4.9.11.1 Lighting, Shading, and Transparency

We've already added Light to our example. Now we can create a Layout3D and associate that Light instance with it. When light is added to a scene it is possible to enable shading effects. The Micro3D engine supports Gouraud and cell, or toon, shading as long as the figure or primitive itself were created to support lighting. When no Light object is available the engine defaults to flat shading which means that every pixel in the figure or primitive is rendered at full intensity (100%). The table below describes the methods available for setting lighting and shading:

Light	getLight ()	Returns the Light used with this Layout3D.
int	getToonHighColor ()	Returns the high color value used for toon shading.
int	getToonLowColor ()	Returns the low color value used for toon shading.
int	getToonThreshold ()	Returns the threshold value used for toon shading.



```

boolean isSemiTransparent()
    Checks whether or not semi-transparent rendering is enabled for this Layout3D.

boolean isToonShaded()
    Checks whether toon shading is enabled for this Layout3D.

void setLight(Light light)
    Sets a Light to be used for this Layout3D.

void setProjection(int type, int[] parameters)
    Sets the projection used by this Layout3D.

void setSemiTransparent(boolean transparent)
    Enables or disables semi-transparency for this Layout3D.

void setToonShading(boolean toon)
    Enables or disables toon shading for this Layout3D.

void setToonShading(int threshold, int highColor, int lowColor)
    Sets the toon shading parameters to the specified values and enables toon shading.

```

The Light instance we created previously can now be set to a Layout3D with `setLight(Light)`. Once a Layout3D has a Light instance, Gouraud shading is enabled. The engine can also do cell shading by enabling it and providing threshold and high and low color values. Shaded colors are compared to the threshold colors and blended with the high color value or low color value depending on whether the threshold was exceeded or not.

Semi-transparency is not dependent on the availability of a Light instance in the Layout3D. However, figures and primitives associated to a Layout3D with semi-transparency enabled must themselves support semi-transparency. For figures this should be done at content creation time. We previously covered semi-transparency for primitives.

4.9.11.2 Setting the Projection Type

The Micro3D engine provides both parallel and perspective projection. One common method is provided for setting up the projection type desired and several constants are defined that describe how projection type is being specified:

```

static int PARALLEL_SCALE
    Identifier for parallel projection specified by setting the scale of the view-to-screen
    transformation.

static int PARALLEL_WIDTH
    Identifier for parallel projection specified by setting the projection plane's width.

static int PARALLEL_WIDTH_HEIGHT
    Identifier for parallel projection specified by setting the projection plane's width and
    height.

static int PERSPECTIVE_FOV
    Identifier for perspective projection specified by field of view parameters.

```



```
static int PERSPECTIVE_WIDTH
    Identifier for perspective projection specified by the width of the near clipping plane's
    projection plane.

static int PERSPECTIVE_WIDTH_HEIGHT
    Identifier for perspective projection specified by width and height of the near clipping
    plane's projection plane.
```

These constants are used in conjunction with the `setProjection(int, int[])` method. Layout3D projection methods are listed in the table below:

```
int[] getProjectionParameters()
    Returns a copy of the projection parameters set for this Layout3D.

int getProjectionType()
    Returns the projection type set for this Layout3D.

void setProjection(int type, int[] parameters)
    Sets the projection used by this Layout3D.
```

Three of the constants above define parallel projection and three define perspective projection. Depending on how a developer wants to set up the projection style any of these constants can be used. Each constant requires a specific number of projection parameters. For example, `PERSPECTIVE_SCALE` requires only two integer arguments, one for the width ratio of the screen transformation and one for the height ratio of the screen transformation. Since the projection handles the 3D translation to the screen coordinate system, the arguments required in the integer array relate to the width and height of the screen. By setting the width and height parameters larger or smaller than the actual screen size, the figures and primitives can be scaled to the desired size.

See the Layout3D Javadocs for complete descriptions of all of the projection constants. Also see the Micro3D Tool Manual for description of parallel and perspective projections.

4.9.12 Automatic View Transformation

While the Layout3D class allows for elaborate scene direction, it is also the most common reason for scenes that don't render correctly. The model, view, and screen coordinates must be set up correctly for any scene to render (model coordinates are contained in the model itself and scene coordinates were covered above). If these values are not set appropriately the Micro3D engine may not render anything at all when a scene is flushed. The Layout3D class contains methods and mechanisms to simplify setting up the view coordinates by manipulating the view and viewpoint transformations.

We covered the viewpoint transformation in the previous section. Now we will see how the viewpoint transformation can be used to automatically set up a view transformation for us.

When a Layout3D instance contains an AffineTransform representing the viewpoint transformation, it can implicitly generate a view transformation, applying rotations along the X, Y, Z, or any arbitrary vectors. The following methods are provided for automatically generating a view transform:



`AffineTransform` **getViewPointTransform()**

Returns the affine transform serving as the viewpoint transformation matrix for this `Layout3D`.

`AffineTransform` **getViewTransform()**

Returns the `AffineTransform` used as the view transformation for this `Layout3D`.

`void` **setViewPoint**(`Vector3D` position, `Vector3D` look,
`Vector3D` up)

Creates an affine transform from the provided vectors that will serve as the viewpoint transformation matrix used when determining the view transformation for this `Layout3D`.

`void` **setViewPointTransform**(`AffineTransform` viewPointTransform)

Sets the affine transform that will serve as the viewpoint transformation matrix used when determining the view transformation for this `Layout3D`.

Although we previously covered creating a viewpoint transformation with the `AffineTransform` class's `getViewPointTransform(Vector3D, Vector3D, Vector3D)` method we now have an alternative way to get the viewpoint transformation into our `Layout3D`. The `setViewPoint(Vector3D, Vector3D, Vector3D)` method creates the viewpoint `AffineTransform` and automatically associates it with the `Layout3D`. Once a viewpoint transformation is associated, the `Layout3D` is able to create the view transformation it needs when translating the figure's points from model coordinate system into the view's coordinate system.

The `Layout3D` allows us to rotate the view transform to give objects in it the appearance of having rotated. Four methods for setting rotation are provided:

`void` **rotateV**(`Vector3D` axis, `int` angle)

Rotates the view transformation matrix about an arbitrary unit vector.

`void` **rotateX**(`int` angle)

Rotates the view transformation matrix about the X axis.

`void` **rotateY**(`int` angle)

Rotates the view transformation matrix about the Y axis.

`void` **rotateZ**(`int` angle)

Rotates the view transformation matrix about the Z axis.

The angle values range from 0 to 4,096, with 4,096 being equivalent to 360 degrees. The rotation is done either by specifying the rotation about the X, Y, and Z axis, or by specifying a rotation about an arbitrary axis. These rotations are mutually exclusive, so once the X, Y, or Z angle is specified, any existing rotation about the arbitrary axis is ignored. Similarly, once the rotation is specified with `rotateV(Vector3D, int)` method, the existing X, Y, and Z angles are ignored. The vector supplied to `rotateV(Vector3D, int)` must be a unit vector. The rotation methods have a secondary function, as well: calling these methods reestablishes automatic handling of the view transformation in the case that a `Layout3D` instance needs to be switched from manual view transformation control to automatic. Note that the rotate methods simply set the rotation angle that will be used when the scene is flushed. The methods are not cumulative and no view transform calculations take place during these calls.



4.9.13 Manual View Transformation

Layout3D also allows developers to create their own transforms to serve as the view transformation. However, it is generally recommended to use automatic view transformation handling for improved performance.

View transformations are typically created by setting up a series of affine transformations then multiplying those transformations against a viewpoint transformation. The following code snippet illustrates how an AffineTransform can be created and used as a view transform (this code will not be used as part of our example):

```
AffineTransform xtrans = new AffineTransform();
AffineTransform ytrans = new AffineTransform();
AffineTransform ztrans = new AffineTransform();
AffineTransform transform = new AffineTransform();

.
.
.
xtrans.set(AffineTransform.M03, 0);
xtrans.set(AffineTransform.M13, 0);
xtrans.set(AffineTransform.M23, 0);
ytrans.set(AffineTransform.M03, 0);
ytrans.set(AffineTransform.M13, 0);
ytrans.set(AffineTransform.M23, 0);
ztrans.set(AffineTransform.M03, 0);
ztrans.set(AffineTransform.M13, 0);
ztrans.set(AffineTransform.M23, 0);

xtrans.rotateX(xAngle);
ytrans.rotateY(yAngle);
ztrans.rotateZ(zAngle);

transform = AffineTransform.multiply(viewpoint, xtrans);

transform.multiply(ytrans);
transform.multiply(ztrans);

layout3d.setViewTransform(transform);
```

Once the AffineTransform is created it can be associated to the Layout3D or retrieved with the following methods:

```
AffineTransform getViewTransform()
    Returns the AffineTransform used as the view transformation for this Layout3D.

void setViewTransform(AffineTransform viewTransform)
    Sets an AffineTransform to be used as the view transformation for this
    Layout3D.
```

The `getViewTransform()` method will return the AffineTransform that was set via `setViewTransform(AffineTransform)`. If the Layout3D is in automatic view transform control you cannot retrieve the automatically generated view transform via `getViewTransform()` and the method will return null.



Going back to our example we'll want to create a `Layout3D` instance for all of the `Object3D` objects we created previously. We then need to set the scene parameters for it and associate it with the figure and primitives:

```
public class GameCanvas3D extends GameCanvas implements Runnable {
    private Graphics myGraphics;
    private Layout3D myLayout3D;
    private Light myLight;
    private Point myPoint;
    private Line myLine;
    private Triangle myTriangle;
    private Quadrangle myQuadrangle;
    private PointSprite myPointSprite;
    private Figure myFigure;
    private Texture myTexture;
    private Texture myTransparentTexture;
    private Texture mySpriteTexture;
    private ActionTable myActionTable;

    private boolean paused;
    private int frame;
    private int frameTime;
    private long oldTime;
    private int maxFrames;

    public GameCanvas3D() {
        super(true);
        paused = false;
        frame = 0;
        frameTime = 0;

        myGraphics = getGraphics();

        //Here we initialize the Layout3D, the lighting,
        //and the viewpoint for our scene
        myLayout3D = new Layout3D();
        Vector3D position = new Vector3D(0, 0, 400);
        Vector3D look = new Vector3D(0, 0, 400);
        Vector3D up = new Vector3D(0, 0, 400);
        myLayout3D.setViewPoint(position, look, up);
        myLayout3D.setProjection(Layout3D.PARALLEL_WIDTH_HEIGHT,
                                new int[]{getWidth()*2,
                                           getHeight()*2});
        Vector3D lightDirection = new Vector3D(-4096, 0, 0);
        lightDirection.normalize();
        myLight = new Light(2048, lightDirection, 4096);
        myLayout3D.setLight(myLight);

        //Now we set up the actors in our 3d scene
        try {
            myFigure = Figure.createFigure("/figure_data.mbac");
            myTexture = Texture.createTexture(
                "/texture_data.bmp", false);
            myTransparentTexture =
```



```

        Texture.createTexture("/trans_texture_data.bmp",
                               false);
    mySpriteTexture =
        Texture.createTexture("/sprite_texture_data.bmp",
                               false);
    myActionTable =
        ActionTable.createActionTable("/action_data.mtra");
} catch (IOException e) {
    System.err.println("Error loading resources: "
                       + e);
}
maxFrames = myActionTable.getNumberOfFrames(0);
myPoint = new Point(new Vector3D(-10, 10, 0),
                    myLayout3D, 0xFF0000);
myLine = new Line(new Vector3D(-10, 10, 0),
                  new Vector3D(50, 0, 0),
                  myLayout3D, 0x00FF00);
myTriangle = new Triangle(new Vector3D(0, 0, 0),
                          new Vector3D(0, 35, 0),
                          new Vector3D(35, 0, 0),
                          myLayout3D,
                          0x0000FF);
myQuadrangle = new Quadrangle(new Vector3D(0, 0, 0),
                              new Vector3D(10, 35, 0),
                              new Vector3D(45, 35, 0),
                              new Vector3D(35, 0, 0),
                              0, 0, 20, 0, 20, 20, 0, 20,
                              myLayout3D,
                              myTransparentTexture);
myQuadrangle.enableColorKeyTransparency(true);
myPointSprite = new PointSprite(new Vector3D(0, 0, 0),
                                30, 30, 0, 0, 0, 30, 30,
                                PointSprite.PIXEL_SIZE |
                                PointSprite.PARALLEL_PROJECTION,
                                myLayout3D, mySpriteTexture);
myFigure.setLayout(myLayout3D);
myFigure.setTexture(myTexture);
myFigure.setActionTable(myActionTable);
myFigure.setPosture(0, frame);
}

```

By going back and revisiting the primitive constructors we've made them ready for rendering. Likewise, the figure is now associated with the Layout3D we just created. You may notice that we set up our projection using the `PARALLEL_WIDTH_HEIGHT` parameter. We then used the width and height returned by the GameCanvas instance and multiplied them by two. The width and height that are reported for projection allow you to scale the objects to a desired size.



4.9.14 Rendering

With the rest of the pieces of our 3D application in place we can finally move onto rendering our scene to the screen. The Micro3D API provides the `Renderer` class as a utility and container class for preparing the 3D scene to be written to a `Graphics` buffer and flushed onto the screen.

Before a scene can be rendered an instance of `Renderer` must be created. `Renderer` uses the default constructor as follows:

`Renderer()`

Creates a new `Renderer`.

The `Renderer` class has one method to register an `Object3D` instance to be drawn by the Micro3D engine. Developers can also specify where that object should be rendered on the buffer when the `Renderer` `paint(Graphics)` or `paint(Texture)` method is called. The following table lists the draw and paint methods for `Renderer`:

```
void
draw(Object3D object3d, int x, int y)
Registers an Object3D to be drawn by this Renderer the next time paint() method is called.
```

```
void
paint(javax.microedition.lcdui.Graphics g)
Paints the primitives and figures registered with this Renderer to the specified Graphics object.
```

```
Texture
paint(Texture texture, int color)
Paints the primitives and figures registered with this Renderer to the specified Texture object.
```

Note that upon returning from a call to `draw`, no rendering or setup is actually performed. The `draw(Object3D, int, int)` method simply stores a reference to the `Object3D` that was specified and copies the coordinates specified. The `Renderer` also stores a separate reference to the `Layout3D` and `Texture` objects in use by the `Object3D` when `draw(Object3D, int, int)` was called. This allows developers to keep a small amount of `Object3D` instances that can take on different appearances by matching them up with different `Texture` and `Layout3D` instances. However, keep in mind that any changes made to the `Object3D` (excluding its layout and textures), `Layout3D`, and `Texture` objects it referred to before `paint()` is called will take effect at render time.

Rendering is actually performed when a call to `paint(Graphics)` or `paint(Texture)` is made. The Micro3D engine can render to a Java `Graphics` object or a `Texture`. When rendering to a previously loaded texture or sphere texture all previous surface data will be lost. The API does not allow rendering to a `MultiTexture`. When rendering to a `Graphics` object the Micro3D engine will honor the clip and translation coordinates. The current color for the `Graphics` object has no effect on the Micro3D engine.

At this point we can make the final changes to our example. We'll create a new `PointSprite` and target texture and add our paint and draw calls:



```

public class GameCanvas3D extends GameCanvas implements Runnable {
    private Graphics myGraphics;
    private Renderer myRenderer;
    private Layout3D myLayout3D;
    private Light myLight;
    private Point myPoint;
    private Line myLine;
    private Triangle myTriangle;
    private Quadrangle myQuadrangle;
    private PointSprite myPointSprite;
    private PointSprite targetSprite;
    private Figure myFigure;
    private Texture myTexture;
    private Texture myTargetTexture;
    private Texture myTransparentTexture;
    private Texture mySpriteTexture;
    private ActionTable myActionTable;

    private boolean paused;
    private int frame;
    private int frameTime;
    private long oldTime;
    private int maxFrames;

    public GameCanvas3D() {
        super(true);
        paused = false;
        frame = 0;
        frameTime = 0;

        myGraphics = getGraphics();
        myRenderer = new Renderer();

        //Here we initialize the Layout3D, the lighting,
        //and the viewpoint for our scene
        myLayout3D = new Layout3D();
        Vector3D position = new Vector3D(0, 0, 400);
        Vector3D look = new Vector3D(0, 0, 400);
        Vector3D up = new Vector3D(0, 0, 400);
        myLayout3D.setViewPoint(position, look, up);
        myLayout3D.setProjection(Layout3D.PARALLEL_WIDTH_HEIGHT,
                                new int[]{getWidth()*2,getHeight()*2});
        Vector3D lightDirection = new Vector3D(-4096, 0, 0);
        myLight = new Light(2048, lightDirection, 4096);
        myLayout3D.setLight(myLight);
        //Now we set up the actors in our 3d scene
        try {
            myFigure = Figure.createFigure("/figure_data.mbac");
            myTexture =
                Texture.createTexture("/texture_data.bmp", false);
            myTransparentTexture =
                Texture.createTexture("/trans_texture_data.bmp",
                                     false);
        }
    }
}

```



```

mySpriteTexture =
    Texture.createTexture("/sprite_texture_data.bmp",
                          false);

myActionTable =
    ActionTable.createActionTable("/action_data.mtra");
} catch (IOException e) {
    System.err.println("Error loading resources: " + e);
}
maxFrames = myActionTable.getNumberOfFrames(0);
myPoint = new Point(new Vector3D(-10, 10, 0),
                    myLayout3D, 0xFF0000);
myLine = new Line(new Vector3D(-10, 10, 0),
                  new Vector3D(50, 0, 0),
                  myLayout3D, 0x00FF00);
myTriangle = new Triangle(new Vector3D(0, 0, 0),
                          new Vector3D(0, 35, 0),
                          new Vector3D(35, 0, 0), myLayout3D,
                          0x0000FF);
myQuadrangle = new Quadrangle(new Vector3D(0, 0, 0),
                              new Vector3D(10, 35, 0),
                              new Vector3D(45, 35, 0),
                              new Vector3D(35, 0, 0),
                              0, 0, 20, 0, 20, 20, 0, 20,
                              myLayout3D, myTransparentTexture);
myQuadrangle.enableColorKeyTransparency(true);
myPointSprite = new PointSprite(new Vector3D(0, 0, 0),
                                30, 30, 0, 0, 0, 30, 30,
                                PointSprite.PIXEL_SIZE |
                                PointSprite.PARALLEL_PROJECTION,
                                myLayout3D, mySpriteTexture);
targetSprite = new PointSprite(new Vector3D(10, 0, 0), 30, 30,
                                0, 70, 10, 30, 30,
                                PointSprite.PIXEL_SIZE |
                                PointSprite.PARALLEL_PROJECTION,
                                myLayout3D,
                                myTargetTexture);

myFigure.setLayout(myLayout3D);
myFigure.setTexture(myTexture);
myFigure.setActionTable(myActionTable);
myFigure.setPosture(0, frame);
}

public void paint(Graphics g) {
    g.setColor(0xFFFFFFFF);
    g.fillRect(0, 0, getWidth(), getHeight());
    myRenderer.draw(myFigure, getWidth()/2,
                    (getHeight()/2)+20);
    myRenderer.draw(myLine, (getWidth()/2)+35,
                    (getHeight()/2)+20);
    myRenderer.draw(myPoint, (getWidth()/2)+55,
                    (getHeight()/2)+40);
    myRenderer.draw(myTriangle, (getWidth()/2)+40,
                    (getHeight()/2)+60);
}

```



```

myRenderer.draw(myQuadrangle, (getWidth()/2)-60,
                (getHeight()/2)+20);
myRenderer.draw(myPointSprite, (getWidth()/2)-60,
                (getHeight()/2)+40);

//For the first frame we'll render to the target texture
//with a purple background
if (myTargetTexture == null) {
    myTargetTexture = myRenderer.paint(null, 0xAA00FF);
    targetSprite.setTexture(myTargetTexture);
}
else {
    myRenderer.draw(targetSprite, (getWidth()/2)-60,
                    (getHeight()/2)-30);
    myRenderer.paint(g);
}
}

public void run() {
    while (!paused) {
        if (myPointSprite.getTextureCoordinateY(Primitive.VERTEX_A)
            == 0)
            myPointSprite.setTextureCoordinates(
                Primitive.VERTEX_A, 0, 30);
        else
            myPointSprite.setTextureCoordinates(
                Primitive.VERTEX_A, 0, 0);
        myPointSprite.setRotation(
            ((myPointSprite.getRotation()+1024) % 4096));
        myFigure.setPosture(0, frame);
        paint(myGraphics);
        flushGraphics();

        //Here we'll update the frame we want to draw.
        //Remember that ActionTable frames range from
        //0 to (max frames * 65536) so updating
        //the frame is not as simple as an increment.
        long time = System.currentTimeMillis();
        frameTime = (int)(time - oldTime);
        oldTime = time;

        //We'll do our frame changes every second
        // thus the divide by 1000
        //We multiply by 30 because we created the animation table
        //at a 30 fps rate, so for every second we want to make
        //sure we are at frame (65536*30*second).
        frame = frame + ((65536 * 30 * frameTime)/1000);
        frame = frame % maxFrames;
    }
}

public synchronized void startAnimation() {
    paused = false;
}

```




```

Thread t = new Thread(this);

//save the time here so we can compare when the thread starts
oldTime = System.currentTimeMillis();
t.start();
}

public synchronized void stopAnimation() {
    paused = true;
}
}

```

When rendering to a texture we check if the target texture is null. If it is, then we create a new instance of Texture object and use it as the target texture. The new texture is returned by the `paint()` method. After that we go ahead and assign the target texture to a PointSprite in the scene.

Our GameCanvas3D example is finished now and can be set as the current Displayable in a MIDlet. The `startAnimation()` method call should be called from `startApp()` and the `stopAnimation()` method should be called in `pauseApp()`.

4.9.15 Utility

The Utility class provides miscellaneous methods for performing integer-based mathematical functions. It also provides a method for retrieving the version of the Micro3D engine in use. The following table lists all of the Utility methods:

<code>static int cos(int angle)</code>	Obtains the cosine approximation of the specified angle.
<code>static String getVersion()</code>	Returns the version of the Micro3D engine.
<code>static int sin(int angle)</code>	Obtains the sine approximation of the specified angle.
<code>static int squareRoot(int x)</code>	Obtains a square root approximation of the specified value.

See the Micro3D API Javadocs for more information on the Utility class.

4.9.16 Memory

Using the Micro3D engine is memory intensive. When developing a 3D MIDlet, always try to conserve the amount of Figures, Textures, and ActionTables used by your application. Other objects in the Micro3D package are generally not memory heavy. The MultiTexture object is simply a collection of references to Texture objects and the Primitives are basically collections of Vector3D, Texture, and texture coordinate information.

However all non-MultiTexture Texture objects require the same amount of memory—around 82K, regardless of the actual size of the texture. When using Textures for Primitives try to combine several images in one Texture for multiple Primitives or animations when possible. The API allows developers to specify sections of the Texture to be used for Primitives.



The `Renderer` class, while unrestricted by the API itself, has practical memory restrictions to be aware of. The `draw()` method allows for 65,535 objects to be registered for painting, however the limited amount of memory available will usually mean a much lower limit. Also, calls to the `Renderer` class's `paint()` methods require extra memory for the Micro3D engine to actually compose the scene. If insufficient memory is available when paint is called the scene will not be rendered. The amount of memory required by the engine varies depending on the complexity of the scene. If your application renders the same or similar scenes on every paint call, try to limit the amount of object instantiations made between paints. This will keep the Java heap relatively static and will speed up rendering over time.

4.9.17 Tips /

- Reuse, reuse, reuse! The API is designed so that complex scenes can be composed with as few objects as possible.
- Combine as much Texture data as you can into one Texture. Non-sphere textures can be up to 256 by 256 pixels.
- Try to allocate all the objects you will need up front when your application initializes to speed up rendering.
- Get familiar with view and viewpoint transforms—most blank rendering results can be traced back to bad coordinates.
- Use the MIDP 2.0 Gaming API to receive key presses.
- Use the Gaming APIs' `GameCanvas` to render your scenes faster.
- The `Renderer`'s `paint` method will try to reclaim memory if necessary by performing garbage collection but heap compaction will not take place. If your heap is fragmented and you have insufficient memory after garbage collection your scene will not render (see tip 1 and 3).

4.9.18 Caveats

Calls to `System.gc()` can be made to ensure that the maximum amount of memory is available for a call to `Renderer.paint()` but can noticeably slow down your application.

4.9.19 Compiling & Testing Micro3D MIDlets

The Micro3D API stub classes will allow you to compile your MIDlet, but should not be used for off-device execution or debugging. The Motorola SDK provides emulation of the Micro3D API.



4.10 Mobile 3D Graphics API



This API is only available on these handsets.

Motorola has implemented all features defined in the Mobile 3D Graphics API. The complete specification is defined in JSR 184 at <http://www.jcp.org>. Our implementation uses HI Corp's Micro3D engine version 4. For more information about HI Corp visit <http://www.hicorp.co.jp>.

The basic features of the Mobile 3D Graphics API are:

- Translation, scaling, and rotation (4x4 matrix.) manipulation on the model
- 3D world drawing by scenegraph API (*)
- Strengthening primitive drawing by Immediate Mode (*)
- Bone animation with SkinnedMesh
- Morphing with MorphMesh
- 2D sprite drawing
- Parallel projection and perspective projection (*)
- Alpha blending (*)
- The accurate drawing by Z buffer
- Point light and spot light can be used on top of ambient light and directional light (*)
- Multiple light support
- Fog effect (*)
- Specification of shading method (flat, gouraud.)
- Multiple textures
- Texture animation

(*) features added to JSR 184 implementation, not included in Micro3D version 3.

4.10.1 Immediate mode and retained mode rendering

There are four different rendering methods. The first method is for rendering an entire World. When this method is used, we say that the API operates in retained mode. The second method is for rendering scene graph nodes, including Groups. The third and fourth methods are for rendering an individual submesh. When the node and submesh rendering methods are used, the API is said to operate in immediate mode.



There is a current camera and an array of current lights in Graphics3D. These are used by the immediate mode rendering methods only. The retained mode rendering method `render(World)` uses the camera and lights that are specified in the World itself, ignoring the Graphics3D camera and lights. Instead, `render(World)` replaces the Graphics3D current camera and lights with the active camera and lights in the rendered World. This allows subsequent immediate mode rendering to utilize the same camera and lighting setup as the World.

4.10.2 Steps for Creating a 3D Application using the Mobile 3D Graphics API

Create 3D data using 3DStudio MAX, a commercial 3D design tool, or any other tool supported by HI.

Create a 3D Graphics File Format that complements the Mobile 3D Graphics API (M3G) using the HI plug-in, the file converter, or any other appropriate tool provided by HI.

*.m3g files contain 3D data (includes models, textures, animation, and scenegraph)

Create J2ME application using i860 SDK or any other J2ME SDKs to load the above files and execute 3D animation. This is usually called rendering the data in retained mode.

Test the program on the i860 phone.

If there are errors, go back to (1) or (3) and correct the data or the MIDlet

Test the application on the target environment.

Note: Simple 3D applications can be created using the immediate mode API and without using commercial tools to create data.

4.10.3 Code Examples

The following is an example of creating a retained mode application:

The application needs to obtain the Graphics3D instance (there is only one), bind a target to it, load m3g data, render everything, and release the target. This is shown in the code fragments below.

If Canvas is used:

```
public class MyCanvas extends Canvas
{
    //Get the Graphics3D instance
    Graphics3D myG3D = Graphics3D.getInstance();

    //Load the ready made m3g data
    Object3D aObject3d[] = Loader.load("3D_data.m3g");

    //Retrieve the world object
    World myWorld = (World)aObject3d[0];

    public void paint(Graphics g) {
        try {
            //Bind the rendering target
            myG3D.bindTarget(g);
        }
    }
}
```



```

... update the scene ...

//Render the 3D context
myG3D.render(myWorld);

} finally {
    myG3D.releaseTarget();
}
}

```

If GameCanvas is used:

```

class MyCanvas extends GameCanvas implements Runnable {

    Graphics3D myG3D2 = Graphics3D.getInstance();
    public void run() {
        try {
            Graphics g = getGraphics();
            myG3D2.bindTarget(g);
            ... update the scene ...
            ... render the scene ...
        } finally {
            myG3D2.releaseTarget();
            flushGraphics();
        }
    }
}

```

The following is example of an immediate mode application:

Class MyCanvas:

```

import javax.microedition.lcdui.*;
import javax.microedition.m3g.*;

public class MyCanvas extends Canvas {

    private Graphics3D      iG3D;
    private Camera          iCamera;
    private Light           iLight;
    private float           iAngle = 0.0f;
    private Transform       iTransform = new Transform();
    private Background      iBackground = new Background();
    private VertexBuffer    iVb;      // positions, normals, colors,
                                     // texcoords
}

```



```
private IndexBuffer    iIb;    // indices to iVB, forming triangle
                           //strips

private Appearance     iAppearance; // material, texture,
                           //compositing, ...

private Material       iMaterial = new Material();

private Image          iImage;

/**
 * Construct the Displayable.
 */
public MyCanvas() {
    // set up this Displayable to listen to command events
    setCommandListener(new CommandListener() {
        public void commandAction(Command c, Displayable d) {
            if (c.getCommandType() == Command.EXIT) {
                // exit the MIDlet
                MIDletMain.quitApp();
            }
        }
    });
    try {
        init();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}

/**
 * Component initialization.
 */
private void init() throws Exception {
    // add the Exit command
    addCommand(new Command("Exit", Command.EXIT, 1));
}
```



```
// get the singleton Graphics3D instance
iG3D = Graphics3D.getInstance();

// create a camera
iCamera = new Camera();
iCamera.setPerspective( 60.0f,           // field of view
    (float)getWidth()/ (float)getHeight(), // aspectRatio
    1.0f,           // near clipping plane
    1000.0f ); // far clipping plane

// create a light
iLight = new Light();
iLight.setColor(0xffffffff);           // white light
iLight.setIntensity(1.25f);           // overbright

// init some arrays for our object (cube)

// Each line in this array declaration represents a triangle
// strip for one side of a cube. The only primitive we can draw
// with is the triangle strip so if we want to make a cube with
// hard edges we need to construct one triangle strip per face
// of the cube.
// 1 * * * * * 0
//   * *      *
//   *  *    *
//   *      * *
// 3 * * * * * 2
// The ASCII diagram above represents the vertices in the first
// line
// (the first tri-strip)
short[] vert = {
// front
    10, 10, 10,  -10, 10, 10,   10,-10, 10,  -10,-10, 10,
// back
    -10, 10,-10,   10, 10,-10,  -10,-10,-10,   10,-10,-10,
```



```
// left
    -10, 10, 10,  -10, 10,-10,  -10,-10, 10,  -10,-10,-10,
// right
    10, 10,-10,   10, 10, 10,   10,-10,-10,   10,-10, 10,
// top
    10, 10,-10,  -10, 10,-10,   10, 10, 10,  -10, 10, 10,
// bottom
    10,-10, 10,  -10,-10, 10,   10,-10,-10,  -10,-10,-10 };

// create a VertexArray to hold the vertices for the object
VertexArray vertArray = new VertexArray(vert.length / 3, 3, 2);
vertArray.set(0, vert.length/3, vert);

// The per-vertex normals for the cube;
// these match with the vertices
// above. Each normal is perpendicular to the
// surface of the object at the corresponding vertex.
byte[] norm = {
    0, 0, 127,    0, 0, 127,    0, 0, 127,    0, 0, 127,
    0, 0,-127,    0, 0,-127,    0, 0,-127,    0, 0,-127,
   -127, 0, 0,   -127, 0, 0,   -127, 0, 0,   -127, 0, 0,
    127, 0, 0,    127, 0, 0,    127, 0, 0,    127, 0, 0,
    0, 127, 0,    0, 127, 0,    0, 127, 0,    0, 127, 0,
    0,-127, 0,    0,-127, 0,    0,-127, 0,    0,-127, 0 };

// create a vertex array for the normals of the object
VertexArray normArray = new VertexArray(norm.length / 3, 3, 1);
normArray.set(0, norm.length/3, norm);

// per vertex texture coordinates
short[] tex = {
    1, 0,        0, 0,        1, 1,        0, 1,
    1, 0,        0, 0,        1, 1,        0, 1,
    1, 0,        0, 0,        1, 1,        0, 1,
    1, 0,        0, 0,        1, 1,        0, 1,
```




```
1, 0,      0, 0,      1, 1,      0, 1,
1, 0,      0, 0,      1, 1,      0, 1 };

// create a vertex array for the texture
// coordinates of the object
VertexArray texArray = new VertexArray(tex.length / 2, 2, 2);
texArray.set(0, tex.length/2, tex);

// the length of each triangle strip
int[] stripLen = { 4, 4, 4, 4, 4, 4 };

// create the VertexBuffer for our object
VertexBuffer vb = iVb = new VertexBuffer();
vb.setPositions(vertArray, 1.0f, null); // unit scale, zero bias
vb.setNormals(normArray);
vb.setTexCoords(0, texArray, 1.0f, null); //unit scale, zero bias

// create the index buffer for our object (this tells how to
// create triangle strips from the contents
// of the vertex buffer).
iIb = new TriangleStripArray( 0, stripLen );

// load the image for the texture
iImage = Image.createImage( "/texture.png" );

// create the Image2D (we need this so we can make a Texture2D)
Image2D image2D = new Image2D( Image2D.RGB, iImage );

// create the Texture2D and enable mipmapping
// texture color is to be modulated with the lit material color
Texture2D texture = new Texture2D( image2D );
texture.setFiltering(Texture2D.FILTER_NEAREST,
                    Texture2D.FILTER_NEAREST);
texture.setWrapping(Texture2D.WRAP_CLAMP,
                    Texture2D.WRAP_CLAMP);
```



```
texture.setBlending(Texture2D.FUNC_MODULATE);

// create the appearance
iAppearance = new Appearance();
iAppearance.setTexture(0, texture);
iAppearance.setMaterial(iMaterial);
iMaterial.setColor(Material.DIFFUSE, 0xFFFFFFFF); // white
iMaterial.setColor(Material.SPECULAR, 0xFFFFFFFF); // white
iMaterial.setShininess(100.0f);

iBackground.setColor(0xf54588); // set the background color
}

/**
 * Paint the scene.
 */
protected void paint(Graphics g) {

    // Bind the Graphics of this Canvas to our Graphics3D. The
    // viewport is automatically set to cover the entire clipping
    // rectangle of the Graphics object. The parameters indicate
    // that z-buffering, dithering and true color rendering are
    // enabled, but antialiasing is disabled.

    iG3D.bindTarget(g, true,
        Graphics3D.DITHER |
        Graphics3D.TRUE_COLOR);

    // clear the color and depth buffers
    iG3D.clear(iBackground);

    // set up the camera in the desired position
    Transform transform = new Transform();
    transform.postTranslate(0.0f, 0.0f, 30.0f);
    iG3D.setCamera(iCamera, transform);
}
```



```
// set up a "headlight": a directional light shining
// from the direction of the camera
iG3D.resetLights();
iG3D.addLight(iLight, transform);

// update our transform (this will give us a rotating cube)
iAngle += 1.0f;
iTransform.setIdentity();
iTransform.postRotate(iAngle,          // rotate 1 degree per frame
                    1.0f, 1.0f, 1.0f); // rotate around this axis

// Render our cube. We provide the vertex and index buffers
// to specify the geometry; the appearance so we know what
// material and texture to use; and the transform to tell
// where to render the object
iG3D.render(iVb, iIb, iAppearance, iTransform);

// flush
iG3D.releaseTarget();
}
}

Class MIDletMain:
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.*;

public class MIDletMain extends MIDlet
{
    static MIDletMain instance;
    MyCanvas displayable = new MyCanvas();
    Timer iTimer = new Timer();
```



```
/**
 * Construct the MIDlet.
 */
public MIDletMain() {
    this.instance = this;
}

/**
 * Main method.
 */
public void startApp() {
    Display.getDisplay(this).setCurrent(displayable);
    iTimer.schedule( new MyTimerTask(), 0, 40 );
}

/**
 * Handle pausing the MIDlet.
 */
public void pauseApp() {
}

/**
 * Handle destroying the MIDlet.
 */
public void destroyApp(boolean unconditional) {
}

/**
 * Quit the MIDlet.
 */
public static void quitApp() {
    instance.destroyApp(true);
    instance.notifyDestroyed();
    instance = null;
}
```



```

/**
 * Our timer task for providing animation.
 */
class MyTimerTask extends TimerTask {
    public void run() {
        if( displayable != null ) {
            displayable.repaint();
        }
    }
}
}

```

4.10.4 The Classification in JSR 184

Below are JSR 184 classes categorized by features:

4.10.4.1 Basic Class

Drawing, transformation, data loader

Class Name	Outline
Object3D	The basic abstract class for most of the 3D classes
Graphics3D	3D drawing class (rendering is executed only by this class)
Transform	Transformation matrix (4x4)
Transformable	Abstract class for Node/Texture2D
Loader	Data loader

Scenegraph Structure

Class Name	Outline
Node	The basic abstract class for all scenegraph elements
World	Top level node of scenegraph
Group	Group of scenegraph elements



4.10.4.2 Basic Geometry

Mesh and Sprite: Scenegraph element

Class Name	Outline
Mesh	Basic 3D object
MorphingMesh	Same as above (for morphing)
SkinnedMesh	Same as above (for animation)
Sprite3D	Sprite

*Mesh is a 3D object composed of polygons (and appearance), and it is the most basic object in JSR 184

The Mesh component element (Submesh: Polygon = vertices + indices and its appearance)

Class Name	Outline
Appearance	Appearance (how polygons look)
TriangleStripArray	Neighboring triangle polygon (index buffer)
VertexBuffer	Vertex buffer

The Appearance component element

Class Name	Outline
PolygonMode	Polygon attribute (Two sided, Flat, Smooth, etc.)
CompositeMode	Pixel drawing attribute (alpha blending)
Fog	Fog attribute
Material	Material attribute (The characteristic of each light)
Texture2D	Texture

The Sub Mesh/Appearance component element

Class Name	Outline
VertexArray	Vertex data array (for vertex buffer, etc.)
Image2D	2D image data for Sprite and Texture

Other

Class Name	Outline
RayIntersection	Light beam and the common part of Mesh/Sprite



4.10.4.3 Background, Camera, Light

Background, camera, light: Scenegraph element

Class Name	Outline
Background	background
Camera	camera
Light	light

Animation

Class Name	Outline
AnimationController	Manages each animation
AnimationTrack	Animation management for each attribute
KeyFrameSequence	Sequence definition for animation data

4.10.5 Tips /

- The Java heap size allocated to 3D is limited (2MB). When making or selecting 3D data, the uncompressed data size should be less than the limit and leave some spaces for the Java objects. One way to check how much space the data requires is to open the data file using HI's V4Examiner; the bottom status bar of this tool will tell you how much space is being used.
- Animation speed (fps – frame rate per second) is affected by some features, such as camera type, so the user has to sacrifice features to get desired fps. One way to check how much effect a feature will have is to toggle the feature on and off through the V4Examiner tool and check the fps.
- To use your favorite Java SDK that doesn't support JSR 184 to develop a JSR 184 application, you need to add JSR 184 classes to the MIDP APIs. In the case of KtoolBar, JSR 184 classes with absolute path information need to be added to \WTK21\lib\midpapi20.jar.
- In order to clear depth buffer contents (and color buffer contents, if so desired) after binding a rendering target, the application must call the clear method, either explicitly or implicitly, by rendering a World.



4.11 Multimedia

4.11.1 Overview



This chapter deals with the audio and video features of the iDEN platform. The Mobile Media API (JSR 135) is an optional API targeted at J2ME™ CLDC-based devices that defines a framework to support playback and recording of audio and video, photo capture from an onboard camera, in addition to tone generation.

The Mobile Media API does not mandate support for any specific media type. It does require that an implementation supporting a given media type must implement certain features in the form of controls.

In each of the class or interface descriptions that follow, if a particular *method* is not listed it is implemented as described in the Sun Javadocs for JSR-135 (MMA 1.0 or MMA 1.1). If a *class* or *interface* listed in the Javadocs is not here, then it is either not implemented or it contains no methods.




These tables provide a quick glance at the supported content types, controls, protocols, and media files. See “4.11.7 Tips” section on page 204 for encoding details on these media files.

Content Types vs. Supported Media Files




	Media File Types							
	TONE	MIDI	WAV	AU	 MP3	iDEN Voicenote	Digital Camera	 Video
Content Type	audio/x-tone-seq	audio/mid audio/midi	audio/x-wav audio/wav	audio/basic	audio/mpeg	audio/x-idenvselp audio/x-idenambe audio/amr	image/jpeg image/rgb565	video/mpeg video/mp4v-es video/quicktime



Controls vs. Supported Media Files

		Media File Types							
		Tone Sequences							
			MIDI	WAV	AU	MP3	IDEN Voicenote	Digital Camera	Video
Control	Volume	Y	Y	Y	Y	Y			Y, if the movie clip has audio
	Tone	Y							
	Tempo		Y						
	Record						Y		 video/quicktime ONLY
	Video							Y	Y

Locator Protocols vs. Supported Media Files

		Media File Types							
		Tone Sequences							
			MIDI	WAV	AU	MP3	IDEN Voicenote	Digital Camera	Video
Protocol	device://	Y							
	file://		Y	Y	Y	Y	Y		Y
	http://		Y	Y	Y	Y	Y		Y
	capture://							Y	 video/quicktime ONLY

There are nine different system properties that can be queried using the method `System.getProperty(String key)`. There are three conditions that are required to say a device supports mixing. iDEN handsets meet one of those conditions— a **MIDI or Tone Sequence** can play simultaneously with a **WAV or AU or MP3**. The default encoding for audio recording and snapshots are in bold.



Device Specific Reported System Properties



Key	System.getProperty(Key)
microedition.media.version	1.1
supports.mixing	false
supports.audio.capture	true
supports.video.capture	false
supports.recording	true
audio.encodings	encoding=idenvselp encoding=idenambe&rate=2200 encoding=idenambe&rate=4400
video.encodings	null
video.snapshot.encodings	null
streamable.contents	null



Key	System.getProperty(Key)
microedition.media.version	1.1
supports.mixing	false
supports.audio.capture	true
supports.video.capture	false
supports.recording	true
audio.encodings	encoding=idenvselp encoding=idenambe&rate=2200 encoding=idenambe&rate=4400
video.encodings	null
video.snapshot.encodings	encoding=jpeg&width=80&height=64 encoding=jpeg&width=128&height=96 encoding=jpeg&width=160&height=120 encoding=jpeg&width=320&height=240 encoding=jpeg&width=640&height=480
streamable.contents	null



Key	System.getProperty(Key)
microedition.media.version	1.1
supports.mixing	false
supports.audio.capture	true
supports.video.capture	false
supports.recording	true
audio.encodings	encoding=idenvselp encoding=idenambe&rate=2200 encoding=idenambe&rate=4400 encoding = amr
video.encodings	null
video.snapshot.encodings	null
streamable.contents	null

i275
only

Key	System.getProperty(Key)
microedition.media.version	1.1
supports.mixing	false
supports.audio.capture	true
supports.video.capture	false
supports.recording	true
audio.encodings	encoding=idenvselp encoding=idenambe&rate=2200 encoding=idenambe&rate=4400
video.encodings	null
video.snapshot.encodings	encoding=jpeg&width=120&height=96 encoding=jpeg&width=130&height=130 encoding=jpeg&width=160&height=120 encoding=jpeg&width=320&height=240 encoding=jpeg&width=640&height=480
streamable.contents	null



Key	System.getProperty(Key)
microedition.media.version	1.1
supports.mixing	false
supports.audio.capture	true
supports.video.capture	true
supports.recording	true
audio.encodings	encoding=idenvselp encoding=idenambe&rate=2200 encoding=idenambe&rate=4400 encoding = amr
video.encodings	encoding=3gp
video.snapshot.encodings	encoding=jpeg&width=128&height=96 encoding=jpeg&width=160&height=120 encoding=jpeg&width=176&height=220 encoding=jpeg&width=320&height=240 encoding=jpeg&width=640&height=480
streamable.contents	null

4.11.2 Class Description

The Multimedia APIs are all located in package class javax.microedition.media. The following is the Class Hierarchy for the Multimedia API:

```
java.lang.Object
|
+-- javax.microedition.media.Manager
|
+-- javax.microedition.media.protocol.ContentDescriptor
|
+-- javax.microedition.media.protocol.DataSource
```



The following is the Interface Hierarchy for the Multimedia API:

```

java.lang.Object
|
+-- javax.microedition.media.Controllable
|
|   +-- javax.microedition.media.Player
|   |
|   +-- javax.microedition.media.protocol.SourceStream
|
+-- javax.microedition.media.PlayerListener
|
+-- javax.microedition.media.Control
|
|   +-- javax.microedition.media.control.RecordControl
|   |
|   +-- javax.microedition.media.control.ToneControl
|   |
|   +-- javax.microedition.media.control.VolumeControl
|   |
|   +-- javax.microedition.media.control.TempoControl
|   |
|   +-- javax.microedition.media.control.VideoControl
|
+-- javax.microedition.media.TimeBase

```

4.11.3 Method Descriptions

4.11.3.1 Manager Methods

4.11.3.1.1 createPlayer

Creates a `Player` from an input locator.

```
public static Player createPlayer (String locator)
```

throws `IOException`, `MediaException`

For music content stored in the JAR file or in a RMS database, use the protocol `file://`. See also the "Locator Protocols vs. Supported Media Files" table on page 189.

```
public static Player createPlayer (InputStream stream,
String type) throws IOException, MediaException
```

The `type` will be checked against the known accepted content-types. However, during the realize process the actual content of `stream` will be analyzed for compatible data.

```
public static Player createPlayer (DataSource source)
```

throws `IOException`, `MediaException`

This method will always throw `MediaException`. See "4.11.7.8 `javax.microedition.media.protocol.Tips`" section on page 205.



4.11.3.1.2 `getSupportedContentTypes`

This method returns the list of supported content types for the given protocol. For example, if the given `protocol` is "http", then the supported content types that can be played back with the `http` protocol will be returned.

4.11.3.1.3 `getSupportedProtocols`

This method returns the list of supported protocols given the content type. The protocols are returned as strings which identify what locators can be used for creating `Players`. For example, if the given `content_type` is "audio/x-wav", then the supported protocols that can be used to play back audio/x-wav will be returned.

4.11.3.1.2 `getSystemTimeBase`

This method will always return null.

4.11.3.1.3 `playTone`

Plays a tone specified by a note and duration.

```
public static void playTone (int note, int duration, int volume)
throws MediaException
```

Duration is limited to a maximum of 4 seconds, but exceptions are not thrown if `duration` is longer. Likewise, the minimum limit for `duration` is 2 milliseconds. Similarly, the maximum and minimum limits on `note` are 103 and 63. This method throws a `MediaException` if any `Player` object is in the `STARTED` state or a previous tone is still playing.

4.11.3.2 Player Methods

4.11.3.2.1 `prefetch`

Acquires resources and processes as much data as necessary to reduce the start latency.

```
public void prefetch() throws MediaException
```

An exception is thrown if:

- the handset is in Vibe-All mode, (does not apply to video-only `Player` instances)
- Java does not have focus, or
- another `Player` object of this media-type or other media type which is compatible with this media type is already in the `PREFETCHED` state.

Keep in mind this other prefetched `Player` may be from another MIDlet or MIDlet suite.



4.11.3.2.2 start

Starts the `Player` as soon as possible.

```
public void start() throws MediaException
```

If playback does not start it may be due to the device sending a `DEVICE_UNAVAILABLE` event after the `Player` was prefetched. Avoid calling `start()` in a `MIDlet's startApp()` method as this may cause the `MIDlet's` UI to freeze at `MIDlet` starting or `MIDlet` resuming screens.

4.11.3.2.3 stop

Stops the `Player`.

```
public void stop() throws MediaException
```

Pause and resume functionality is not supported. Stopping any media type means a subsequent call to `start()` will play from the beginning of the file.

4.11.3.2.4 getSnapshot



On handsets featuring a built-in camera or supporting a camera accessory, this method gets a snapshot of the displayed content.

```
public byte[] getSnapshot(String imageType) throws MediaException
```

An exception is thrown if:

- `initDisplayMode` has not been called.
- the requested format is not supported.
- the caller does not have the security permission to take the snapshot.






For phones without a built-in camera, the `getSnapshot()` may be used to take a picture when the camera accessory is connected.

The `imageType` parameter allows developers to specify certain attributes in the `getSnapshot()` method. Note that all attribute and value pairs must be lower-case and separated with an "&".

The table below shows which attributes can be set and what values can be used with them. It is important to use the attributes in the same order as listed in the table. For example, calling `getSnapshot("lighting=normal&brightness=75")` will fail, however `getSnapshot("brightness=75&lighting=normal")` will pass.




The default value for each attribute is used each time `getSnapshot()` is called. This means that one call with brightness set to 40 will not affect successive calls where the brightness attribute is not specified.

Attribute	Description	Min	Max	Default
<div>preview</div> <div></div> <div>i265 only</div>	When preview is “yes” the returned image is always 128x96 and encoded in a raw RGB format used by the hardware. The data can be passed to <code>Image.createImage()</code> and displayed on a <code>Canvas</code> . The speed advantage of using a preview image makes it ideal for showing a viewfinder. A preview image is not effected by the quality attribute.	no	yes	no
<div>brightness</div> <div></div> <div>i265 only</div>	Adjusts the brightness of the captured picture, with 1 being very dark and 100 being very light.	1	100	50
<div>lighting</div> <div></div> <div>i265 only</div>	<div><div></div><div>Developers can also turn “on” the flash. It behaves more like an on/off switch than a flash.</div></div> <div>“Low” adjusts the camera aperture for low light conditions. Use “Normal” in any other condition.</div>	<div>“low” “normal”</div> <div> “on”</div>	“normal”	

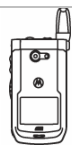
getSnapshot Attribute Values

Table continued on next page.






Attribute	Description	Min	Max	Default
quality	Sets the image quality of an encoded <i>jpeg</i> capture. Even though a large range of values is available for use, the effectiveness is limited to 1-59 (normal), 60-79 (better), and 80-100 (best). It is recommended to use 50, 65, or 80.	1	100	100
				<div> i275 only</div> 65

getSnapshot Attribute Values (continued)



i275 only






Handsets with a built-in camera have the ability to specify other attributes in the `createPlayer()` method just like `getSnapshot()` above. The attribute and value pairs are as follows:

Attribute	Description	Min	Max	Default
encoding	The type of video to capture.	"jpeg"		"jpeg"
			"3gp"	
audio	Indicate whether audio should be captured.	"yes" "no"		"yes"
width	The width of the captured image, NOT the viewfinder. See also "video.snapshot.encodings"	128	640	 176
				 128 i275 only

createPlayer Attribute Values

Table continued on next page.



Attribute	Description	Min	Max	Default
height	The height of the captured image, NOT the viewfinder. See also "video.snapshot.encodings"	96	480	 220
				 96 i275 only
 i860 only	Zoom factor for the camera. There are three supported zooms: 1X, 2X, and 4X.	1	4	1
 i860 only	Adjusts the brightness of the viewfinder and captured picture, with 1 being very dark and 100 being very light.	1	100	50
 i860 only	Adjusts the contrast of the viewfinder and captured picture, with 0 being very dark and 10 being very light.	0	10	6

createPlayer Attribute Values (continued)



4.11.4 Video Playback

This feature enables playback of the multimedia capabilities of the iDEN handset based on Multimedia API JSR 135 for J2ME™ devices. An implementation supporting video must provide a `VideoControl`.

Players are created using the `createPlayer(...)` factory method of the `javax.microedition.media.Manager` class. The `createPlayer(...)` method has the following signatures:

```
Public static Player createPlayer(String locator)

Public static Player createPlayer(InputStream stream, String
type)

Public static Player createPlayer(DataSource source)
```

Playing video is similar to playing audio. However, the video player needs to be told where to display the video. Therefore you get a video control from the video player and display it either as a Form item or in a Canvas.

To display video as a Form item:

```
Player player=Manager.createPlayer( "file://video/3gpp");
player.realize();
VideoControl vc=(VideoControl)player.getControl("VideoControl");
if(vc!=null)
{
    Item it=(Item)vc.initDisplayMode(VideoControl.USE_GUI_PRIMITIVE,null);
    myForm.append(it);
    player.start();
}
```

To display video in a Canvas

```
Player player=Manager.createPlayer("file://video/3gpp");
player.realize();
VideoControl vc=(VideoControl)player.getControl("VideoControl");
if(vc!=null)
{
    vc.initDisplayMode(VideoControl.USE_DIRECT_VIDEO,myCanvas);
    vc.setVisible(true);
    player.start();
}
```

The `run()` method, mandated by the `Runnable` interface, contains the initialization of the Player. The rationale behind putting this initialization in a separate thread is that in many cases (although not this particularly simple example) it may be desirable to perform the potentially time consuming initialization in the background, so that when rendering of the video is required it starts with minimum latency. Another reason for performing the initialization in a separate thread is so that it is possible to update a progress gauge, giving the user valuable feedback as to how the video acquisition is proceeding. Since `VideoCanvas` implements the `PlayerListener` interface we register this instance with the Player to receive calls back. The `prefetch()` and `realize()` methods are called on the player and a gauge updated at the return of each method.



```
public void run() {
    try {
        player = Manager.createPlayer(url);
        player.addPlayerListener(this);
        player.realize();
        player.prefetch();
    } catch (IOException ioe)
    {
        //handle
    } catch (MediaException me)
    {
        //handle
    }
    playVideo();
}
```

Once the player is in the PREFETCHED state we are ready to render the video content. In this example our `playVideo()` method is called immediately.

```
public void playVideo() {
    try {
        videoControl =
        (VideoControl)player.getControl("VideoControl");
        if (videoControl != null) {

            videoControl.initDisplayMode(videoControl.USE_DIRECT_VIDEO,
            this);
        }
        display.setCurrent(this);
        videoControl.setVisible(true);
        player.start();
    } catch (MediaException me)
    {
        //handle
    }
}
```

The `playVideo()` method handles rendering the video onto the Canvas. To do this we must obtain a `VideoControl`, by calling `getControl()` on a realized `Player`, and cast down appropriately. The `initDisplayMode()` method is used to initialize the video mode that determines how the video is displayed. This method takes an integer mode value as its first argument with two predefined values `USE_GUI_PRIMITIVE` or `USE_DIRECT_VIDEO`. In the case of the MIDP implementations (supporting the LCDUI) `USE_GUI_PRIMITIVE` will result in an instance of a `javax.microedition.lcdui.Item` being returned. For example:



```
Item display =
control.initDisplayMode(control.USE_GUI_PRIMITIVE,"javax.microedition.l
cdui.Item");
```

Since our class is an instance of Canvas we must implement a `paint()` method as shown below

```
public void paint(Graphics g){
    g.setColor(128, 128, 128);
    g.fillRect(0, 0, getWidth(), getHeight());
}
```

Here our implementation simply fills the canvas with a suitable background color. The video content is then rendered directly onto the Canvas by the VideoControl. Since our VideoCanvas class implements the PlayerListener interface we must provide a `playerUpdate()` method:

```
public void playerUpdate(Player p, String event, Object eventData) {
    if (event == PlayerListener.END_OF_MEDIA)
    {
        if (rePlay == null) {
            rePlay = new Command("re-play", Command.SCREEN, 1);
            addCommand(rePlay);
        }
    }
}
```

In the code shown above we simply listen for the `END_OF_MEDIA` event and add a replay option to our commands when the trailer has finished playing.

Finally we shall have a look at the `commandAction()` method mandated by `CommandListener`.

```
public void commandAction(Command c, Displayable s) {
    if(c == rePlay)
    {
        try{
            player.start();
        } catch (MediaException me) {
            //handle
        }
    }
    else if(c == close)
    {
        player.close();
        parent.form.delete(1);
        display.setCurrent(parent.form);
        url=null;
        parent=null;
    }
}
```



Supported video file type extensions:

Video File Type	File Extensions
MJPEG	.avi .mpeg .mpg .mpe
MPEG4	.mp4
3GP	.mov .qt .3gp

4.11.5 Tips and Code Examples /

See Sun's JSR 135 Javadocs for more use-case scenarios and code examples.

4.11.5.1 Basic Playback

The simplest way to begin playback is the following. It assumes that the MIDI file "piano_solo.mid" is in the root path of the JAR file.

```
{
    Player p = Manager.createPlayer("file://piano_solo.mid");
    p.start();
}
```

You can also pass the `InputStream` as a parameter. It assumes that `x` contains the buffer of the media data.

```
{
    Player player =Manager.createPlayer(x,"audio/x-idenvselp");
    player.start();
}
```

4.11.5.2 Capture Picture

The most straightforward way to capture and display an image from a digital camera is the following. It assumes the camera is attached. The i860 has a built-in viewfinder capability. Calling `Player.start()` will begin the viewfinder and `Player.stop()` will close it. Taking a snapshot will take the picture, but then close the viewfinder, putting the player goes into the PREFETCHED state.

```
{
    // Create the Player object using the default
    // resolution.
    Player p = Manager.createPlayer("capture://video");
    p.realize();
}
```



```
VideoControl vc = (VideoControl)p.getControl("VideoControl");
if (vc != null)
{
    vc.initDisplayMode(VideoControl.USE_GUI_PRIMITIVE, null);

    p.start();

    //Take the picture and create Image object
    byte[] jpgImage = vc.getSnapshot(null);
    Image myImage =
Image.createImage(jpgImage, 0, jpgImage.length);

    //Assume Graphics object g is present
    g.drawImage(myImage, 0, 0, g.TOP | g.LEFT);
}
}
```

4.11.5.3 Common Mistake

Try to avoid the mistake of reusing the same Player object without first calling its `close()` method. It can cause hours of debugging frustration.

```
{
    Player p = Manager.createPlayer("file://xyz.mid");
    p.start();

    // Object "p" could still be playing. Must call p.close()
    // before reusing. Besides, there would no longer be a
    // reference to the above midi and eventually, the maximum
    // number of Player objects in REALIZED state or beyond
    // will be reached.
    p.close();

    //Reuse "p"
    p = Manager.createPlayer("http://www.soap108.com/abc.mid");
}
```



```

p.start();    //  <-- Note: An exception would have been
                //      thrown during prefetch() if p.close()
                //      had not been called.
                //      See bullet #3 in section 4.8.3.2.1.
    }

```

4.11.6 Compiling & Testing MMA MIDlets

Use Sun's Wireless Tool Kit 2.0 to compile code and package JAR / JAD files.

4.11.7 Tips /

4.11.7.1 General Tips:

- Voicenote files are played according to voice volume, not Java volume.
- The packages `com.motorola.midi` and `com.motorola.iden.voicenote` are deprecated.
- The i860 and i605 supports a maximum of 32 `Player` objects in the `REALIZED` state or beyond. All other handsets support a maximum of 10 `Player` objects in the `REALIZED` state or beyond. An existing `Player` object must be closed before a new `Player` object can be realized, when the maximum is reached.
- MIDI files may be either Type-0 or Type-1. The SP-MIDI format is also supported, but note that iDEN handsets allow for up to 16 instruments at once.
- WAV files must be 8 kHz, 8-bit, mono PCM encoded files, formatted as a RIFF WAV file with little-endian byte order. RIFX WAV files that support the big-endian byte-ordering scheme are not recognized. In the file's 44-byte header, the "format" data must be 16 bytes in length. This follows the canonical WAV format. Note that Windows WAV files often have 18 bytes of "format" data. If the `Player` cannot be realized with such a file, simply remove the two extra bytes. A PC tool that can automatically strip out these bytes and ensure a canonical format is at <http://www.bluechillies.com/browse/W/B/H/>. The application name is "StripWav 2.0.3".
- AU files must be 8 kHz, 8 bit, mono u-law encoded files. Only big-endian AU files with a magic number of ".snd" are considered. The magic field name identifies files as a Next/Sun sound file and is used for type checking and byte ordering information.
- For the i860 and i605, MP3 file must be 8 kHz, 32/48 kbps, mono encoded files.
- When using the camera accessory, the baud rate must be set to Auto. This option is found in the Main Menu under Settings > Advanced > Baud Rate. . A phone with a built-in camera is unaffected by the baud rate.
- The first time `getSnapshot()`, `setRecordLocation()`, or `setRecordStream()` is called, a system security screen pops up and prompts the user to grant or deny. This screen suspends the MIDlet and causes all `Players` to go to `REALIZED` state. Once the selection is made, the MIDlet resumes but the above methods throw an exception because the `Player` is no longer in the right state. The `Player` will need to be started



again and another call to this method will be needed. Thus, if permission is granted, it is highly encouraged that “one shot” not be used. Always select “blanket” or “session.”

Note: The i860 behaves a little differently for `getSnapshot()`. If permission is granted, the method will at least return the data for that picture. The player object will however go back to the REALIZED state.

4.11.7.2 PlayerListener Tips

The following events are not used: `BUFFERING_STARTED`, `BUFFERING_STOPPED`, `DURATION_UPDATED`, and `STOPPED_AT_TIME`. No OEM events are implemented.

4.11.7.3 VolumeControl Tips

The lower the volume the more likely the audio will sound scratchy. By default, a `Player`'s volume is set to 100% of the Java Volume. See the “Definitions, Abbreviations and Acronyms” section on page 12.

4.11.7.4 ToneControl Tips

The effective duration of a note should be at most 2.68 seconds. If not, the note will be clipped to this limit. `SILENT` notes are not held to this limit. On a PC, this duration limit will not necessarily exist.

4.11.7.5 TempoControl Tips

Only `TempoControl` is implemented on iDEN handsets (for MIDI files only), despite inheriting methods from `RateControl`. All `RateControl` methods return `-1`. The range of values supported for `setTempo()` are 10 - 300 beats per minute.

4.11.7.6 RecordControl Tips

This control can be used to capture iDEN Voicenote Files. Recordings to a location beginning with `"file:/"` are put in a temporary folder and are removed when the application exits. To preserve such a recording it must be copied to a permanent location.

4.11.7.7 VideoControl Tips

The method `getSnapshot()` may be used to take a picture if the camera accessory is connected or if the handset features an onboard camera. See also “`video.snapshot.encodings`” in section 4.13.3.1.3 `getSnapshot` on page 258.

4.11.7.8 javax.microedition.media.protocol Tips

This MMA implementation does not provide any OEM data sources.



4.12 Real-time Protocol

4.12.1 Overview



This API is only available
on this handset.

The RTP API provides RTP protocol integration for MIDlets. J2ME MIDlets can receive and transmit real-time data such as audio, video or simulation data over a network using the RTP API.

4.12.2 Class Description

The RTP API is divided into four packages:

Package Summary	
com.motorola.iden.media	Defines the media related classes
com.motorola.iden.media.rtp	Implements the abstract classes and interfaces in javax.media.rtp
com.motorola.iden.media.rtp.event	Defines all the event classes
com.motorola.iden.media.protocol	This package is defined for supporting the MMAPI
Package com.motorola.iden.media Class Summary	
UnsupportedFormatException	Defines the exception when media format is not supported.



Package com.motorola.iden.media.rtp Class Summary

RTPManager	RTPManager, the starting point for creating, maintaining and closing an RTP session.
ReceiveStream	Represents a receiving stream within an RTP session.
ReceiveStreamListener	Generates the callback for all RTPManager Events.
RTPStream	Base interface represents a stream within an RTP session.
SendStream	Represents a sending stream within an RTP session.
SendStreamListener	Generates the callback for RTPManager Events.
SessionAddress	Encapsulates the RTP session address
SessionManagerException	Defines the basic session exception
InvalidSessionAddressException	Defines the invalid session address exception

Package com.motorola.iden.media.rtp.event Class Summary

NewReceiveStreamEvent	Informs the RTP listener that a new stream of RTP data packets has been detected.
NewSendStreamEvent	Informs the RTP listener that a new transmitting stream has been created in this RTPManager.
ReceiveStreamEvent	The ReceiveStreamEvent will notify a listener of all events that are received on a particular ReceiveStream.
RTPEvent	The Base class of all event notification in the RTPManager.
SendStreamEvent	Notifies a listener of all events that are received on a particular SendStream.
StreamClosedEvent	Informs the RTP listener that a transmitting stream has been closed in the RTPManager

Package com.motorola.iden.media.protocol Class Summary

RTPContentDescriptor	Defines the media content in javax.microedition.media.protocol.DataSource
-----------------------------	---



Class Hierarchy

The following is the class hierarchy for the RTP API:

```
com.motorola.iden.media.UnsupportedFormatException
```

```
com.motorola.iden.media.rtp.RTPManager  
com.motorola.iden.media.rtp.SendStream  
com.motorola.iden.media.rtp.ReceiveStream  
com.motorola.iden.media.rtp.ReceiveStreamListener  
com.motorola.iden.media.rtp.RTPStream  
com.motorola.iden.media.rtp.SendStreamListener  
com.motorola.iden.media.rtp.InvalidSessionAddressException  
com.motorola.iden.media.rtp.SessionManagerException  
com.motorola.iden.media.rtp.SessionAddress
```

```
com.motorola.iden.media.rtp.event.NewReceiveStreamEvent  
com.motorola.iden.media.rtp.event.NewSendStreamEvent  
com.motorola.iden.media.rtp.event.ReceiveStreamEvent  
com.motorola.iden.media.rtp.event.RTPEvent  
com.motorola.iden.media.rtp.event.SendStreamEvent  
com.motorola.iden.media.rtp.event.StreamClosedEvent
```

```
com.motorola.iden.media.RTPContentDescriptor
```

4.12.2.1 Class RTPManager

4.12.2.1.1 addFormat

```
public void addFormat(RTPContentDescriptor f, int payload)
```

This method is used to add a dynamic payload to format mapping to the RTPManager. The RTPManager maintains all static payload numbers and their corresponding formats as mentioned in the Audio/Video profile document. Before streaming dynamic payloads, an RTPContentDescriptor object needs to be created for each of the dynamic payload types and associated with a dynamic payload number.

Parameters:

f - The RTPContentDescriptor to be associated with this dynamic payload number.

payload - The RTP payload number must between [96,127]



24.12.2.1.2 addReceiveStreamListener

```
public void addReceiveStreamListener(ReceiveStreamListener listener)
```

Adds a `ReceiveStreamListener`. This listener listens to all the events that cause state transitions for a particular `ReceiveStream`. If the listener is null, no actions will be taken. To remove a `ReceiveStreamListener` use `removeReceiveStreamListener(ReceiveStreamListener listener)`.

Parameters:

listener - the `ReceiveStreamListener` added to the list of listeners for this `RTPManager`

4.12.2.1.3 addSendStreamListener

```
public void addReceiveStreamListener(ReceiveStreamListener listener)
```

Adds a `ReceiveStreamListener`. This listener listens to all the events that cause state transitions for a particular `ReceiveStream`. If the listener is null, no actions will be taken. To remove a `ReceiveStreamListener` use `removeReceiveStreamListener(ReceiveStreamListener listener)`.

Parameters:

listener - the `ReceiveStreamListener` added to the list of listeners for this `RTPManager`

4.12.2.1.4 removeTarget

```
public void removeTarget(SessionAddress remoteAddress, java.lang.String reason)
```

throws `InvalidSessionAddressException`

Closes all open streams associated with the endpoint defined by `remoteAddress`.

Parameters:

`remoteAddress` - The RTP session address of a remote endpoint for this session. i.e. the IP address/port of a remote host.

`reason` - A string that the RTCP will send out to other participants as the reason the local participant has quit the session. This RTCP packet will go out with the default SSRC of the session. If supplied as null, a default reason will be supplied by the `RTPManager`.

Throws:

`InvalidSessionAddressException` if the `SessionAddress` is null or cannot be parsed into a valid IP address

4.12.2.1.5 removeTargets

```
public void removeTargets(java.lang.String reason)
```

Closes the open streams associated with all remote endpoints that have been added previously by subsequent `addTarget()` calls.

Parameters:

`reason` - A string that RTCP will send out to other participants as the reason the local participant has quit the session. This RTCP packet will go out with the default SSRC of the session. If supplied as null, a default reason will be supplied by the `RTPManager`.



4.12.2.1.6 createSendStream

```
public SendStream createSendStream(javax.microedition.media.protocol.DataSource  
dataSource, int streamIndex)
```

throws `UnsupportedFormatException`, `java.io.IOException`

This method is used to create a sending stream within the RTP session. For each time the call is made, a new sending stream will be created.

The RTP payload that is used to send this stream is found from the format set on the `SourceStream` of the data source supplied.

Parameters:

`dataSource` - This data source may contain more than one stream. The stream which is used in creating this RTP stream is specified by the `streamIndex` parameter.

`streamIndex` - The index of the sourcestream from which data is sent out on this RTP stream. An index of 1 would indicate the first sourcestream of this data source should be used to create the RTP stream. If the index is set to zero, it would indicate a RTP mixer operation is desired. i.e. all the streams of this data source must be mixed into one single stream from one single SSRC.

Returns:

The `SendStream` created by the `RTPManager`.

Throws:

`UnsupportedFormatException` - (`javax.media.format.UnsupportedFormatException`).

This exception is thrown if the format is not set on the `SourceStream` or a RTP payload cannot be located for the format set on the `SourceStream`.

`java.io.IOException` - Thrown for two possible reasons which will be specified in the message part of the exception 1) If there was any problem opening the sending sockets

4.12.2.1.7 dispose

```
public void dispose()
```

Releases all objects allocated in the course of the session and prepares the `RTPManager` to be garbage collected. This method should be called at the end of any RTP session.

4.12.2.1.8 getReceiveStreams

```
public java.util.Vector getReceiveStreams()
```

Returns the a `Vector` of `ReceiveStream` objects created by the `RTPManager`. These are streams formed when the `RTPManager` detects a new source of RTP data. The `ReceiveStream` objects returned are a snapshot of the current state in the `RTPManager`. The `ReceiveStreamListener` interface may be used to get notified of additional streams.

Returns:

A `Vector` containing all the `ReceiveStream` objects created by this `RTPManager`



4.12.2.1.9 getSendStreams

```
public java.util.Vector getSendStreams()
```

Returns the SendStreams created by the RTPManager. SendStreams returned are a snapshot of the current state in the RTPSession and the SendStreamListener interface may be used to get notified of additional streams.

Returns:

A Vector containing all the SendStream objects created by this RTPManager

4.12.2.1.10 initialize

```
public void initialize(SessionAddress localAddress)
```

throws InvalidSessionAddressException, java.io.IOException

Initializes the session. Once this method has been called, the session is "initialized" and this method cannot be called again.

Parameters:

localAddress - Encapsulates the local control and data addresses to be used for the session.

Throws:

InvalidSessionAddressException - if the localAddress is null or not a valid local address.

java.io.IOException - if meeting problems during initialize network.

4.12.2.1.11 addTarget

```
public void addTarget(SessionAddress remoteAddress)
```

throws InvalidSessionAddressException, java.io.IOException

This method opens the session, causing RTCP reports to be generated and callbacks to be made through the SessionListener interface. This method must be called after session initialization and prior to the creation of any streams on a session.

Parameters:

remoteAddress - the RTP session address of a remote endpoint for this session. i.e. the IP address/port of a remote host

Throws:

InvalidSessionAddressException - if the remote control and data addresses given in localAddress parameter are not valid session addresses.

java.io.IOException - if meeting problems during open network connection.



4.12.2.1.12 removeReceiveStreamListener

```
public void removeReceiveStreamListener(ReceiveStreamListener listener)
```

Removes a ReceiveStreamListener.

Parameters:

listener - the ReceiveStreamListener to be removed

4.12.2.1.13 removeSendStreamListener

```
public void removeSendStreamListener(SendStreamListener listener)
```

Removes a SendStreamListener.

Parameters:

listener - the SendStreamListener to be removed

4.12.2.1.14 newInstance

```
public static RTPManager newInstance()
```

Create an RTPManager object for the underlying implementation class.

4.12.2.2 Class UnsupportedOperationException

4.12.2.2.1 UnsupportedOperationException

```
public UnsupportedOperationException()
```

Constructs a new UnsupportedOperationException with no message string.

4.12.2.2.2 UnsupportedOperationException

```
public UnsupportedOperationException(java.lang.String message)
```

Constructs a new UnsupportedOperationException with the specified parameters.

Parameters:

message - A String that contains a message associated with the exception



4.12.2.3 Class RTPContentDescriptor

4.12.2.3.1 RTPContentDescriptor

```
public RTPContentDescriptor(java.lang.String contentType, int sample, int ts)
```

Construct the RTPContentDescriptor object .

Parameters:

contentType - A String that represents the media content

sample - the sample rate of the media

ts - the time stamp unit in the RTP Header

4.12.2.3.2 getSampleRate

```
public int getSampleRate()
```

Returns:

the sample rate in the RTPContentDescriptor

4.12.2.3.3 getTimeStampUnit

```
public int getTimeStampUnit()
```

Returns:

the time stamp unit in the RTPContentDescriptor

4.12.2.4 Class SessionAddress

4.12.2.4.1 SessionAddress

```
public SessionAddress(java.lang.String saddr, int port)
```

Constructor to create a SessionAddress given the data internet address and data port.

Parameters:

saddr - the internet address

port - the data port of rtp session.

Note: If the value of the port parameter is set to ANY_PORT, the SessionAddress created will not specify a specific port.

4.12.2.4.2 getIPAddr

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

This method returns the internet address of this SessionAddress.

4.12.2.4.3 getDataPort

```
public int getDataPort()
```

This method returns the data port of this SessionAddress.



4.12.2.5 Interface ReceiveStream

4.12.2.5.1 getDataSource

```
public javax.microedition.media.protocol.DataSource getDataSource()
```

Returns the datasource of the stream

4.12.2.6 Interface ReceiveStreamListener

4.12.2.6.1 update

```
public void update(ReceiveStreamEvent event)
```

Call back method used to provide notifications of all ReceiveStream Events.

Parameters:

event - the related RTP event.

4.12.2.7 Interface RTPStream

4.12.2.7.1 getDataSource

```
public javax.microedition.media.protocol.DataSource getDataSource()
```

Returns the datasource of the stream.

4.12.2.8 Interface SendStream

4.12.2.8.1 close

```
public void close()
```

Removes the stream from the session. When this method is called the RTPSM deallocates all resources associated with this stream and releases internal references to this object as well as the Player which had been providing the send stream.

4.12.2.8.2 stop

```
public void stop()
```

throws java.io.IOException

Will temporarily stop the RTPSendStream i.e. the local participant will stop sending out data on the network at this time.

Throws:

java.io.IOException - Thrown if there was any IO problems when stopping the RTPSendStream. A stop to the SendStream will also cause a stop() to be called on the stream's datasource. This could also throw an IOException, consistent with datasources in JMF.



4.12.2.8.3 start

public void start()

throws java.io.IOException

Will resume data transmission over the network on this RTPSendStream.

Throws:

java.io.IOException - Thrown if there was any IO problems when starting the RTPSendStream. A start to the SendStream will also cause a start() to be called on the stream's datasource. This could also throw an IOException, consistent with datasources in JMF.

4.12.2.9 Interface SendStreamListener

4.12.2.9.1 update

public void update(SendStreamEvent event)

Call back method used to provide notifications of all SendStream Events

Parameters:

event - the related RTP event

4.12.2.10 Class InvalidSessionAddressException

4.12.2.10.1 InvalidSessionAddressException

public InvalidSessionAddressException()

Constructs the InvalidSessionAddressException object with no description string.

4.12.2.10.2 InvalidSessionAddressException

public InvalidSessionAddressException(java.lang.String reason)

Construct the InvalidSessionAddressException object with the specified description string.

Parameters:

reason - the description string for the exception

4.12.2.11 Class SessionManagerException

4.12.2.11.1 SessionManagerException

public SessionManagerException()

Construct the SessionManagerException object with no description string.



4.12.2.11.2 SessionManagerException

`public SessionManagerException(java.lang.String reason)`

Construct the SessionManagerException object with the specified description string.

Parameters:

reason - the description string for the exception

4.12.2.12 Class RTPEvent

4.12.2.12.1 RTPEvent

`public RTPEvent(RTPManager source)`

Construct an RTP event.

Parameters:

source - the RTPManager generating this event.

4.12.2.12.2 getSessionManager

`public RTPManager getSessionManager()`

Returns:

The RTPManager generating this event.

4.12.2.12.3 getSource

`public RTPManager getSource()`

Returns:

the RTPManager generating this event.

4.12.2.13 Class ReceiveStreamEvent

4.12.2.13.1 ReceiveStreamEvent

`public ReceiveStreamEvent(RTPManager source, ReceiveStream stream)`

Construct the ReceiveStreamEvent object.

Parameters:

source - the RTP manager that produces this event

stream - the ReceiveStream related to this event.

4.12.2.13.2 getReceiveStream

`public ReceiveStream getReceiveStream()`

returns the ReceiveStream object related to this event



4.12.2.14 Class NewReceiveStreamEvent extends ReceiveStreamEvent

4.12.2.14.1 NewReceiveStreamEvent

`public NewReceiveStreamEvent(RTPManager source, ReceiveStream receiveStream)`

Construct the NewReceiveStreamEvent object.

Parameters:

source - the RTPManager generating the event.

receiveStream - the ReceiveStream related to this event.

4.12.2.15 Class SendStreamEvent

4.12.2.15.1 SendStreamEvent

`public SendStreamEvent(RTPManager source, SendStream stream)`

Construct the SendStreamEvent object.

Parameters:

source - the RTPManager generating this event

stream - the SendStream related to this event.

4.12.2.15.2 getSendStream

`public SendStream getSendStream()`

get the SendStream related to this event.

4.12.2.16 Class NewSendStreamEvent extends SendStreamEvent

4.12.2.16.1 NewSendStreamEvent

`public NewSendStreamEvent(RTPManager source, SendStream sendStream)`

Construct the NewSendStreamEvent object.

Parameters:

source - the RTP manager generating this event

sendStream - the SendStream related to this event



4.12.2.17 Class StreamClosedEvent extends SendStreamEvent

4.12.2.17.1 StreamClosedEvent

`public StreamClosedEvent(RTPManager source, SendStream sendStream)`

Constructs a new StreamClosedEvent indicating that a transmitting stream has been closed in an RTPSessionManager.

Parameters:

source - The RTPManager generating this event

sendStream - the send stream related to this event

4.12.3 Code Example

This is an example of using RTP

```
import javax.microedition.midlet.*;
import com.motorola.iden.media.rtp.*;
import com.motorola.iden.media.rtp.event.*;
import javax.microedition.lcdui.*;
import com.motorola.iden.media.protocol.*;
import com.mot.cldc.io.*;
import com.mot.security.*;
import javax.microedition.media.protocol.*;

public class SendAndRecvTest4 extends javax.microedition.midlet.MIDlet implements
CommandListener, SendStreamListener, ReceiveStreamListener {
    RTPManager Amanager;
    SessionAddress localaddr;
    SessionAddress remoteaddr;
    private Display myDisplay;
    Form fmAnswer;
    public StringItem siRTPText;
    public StringItem siRTPText2;
    Command testCmd;
    Command closeCmd;
    Command exitCmd;
    SendStream sendStream = null;
    //DataSource ds = null;
    public SendAndRecvTest4() {
        fmAnswer = new Form("RTP TEST");
        siRTPText = new StringItem("RTP Send Status:", "");
        siRTPText2 = new StringItem("RTP Recv Status:", "");
        testCmd = new Command("Test", 1, 1);
        closeCmd = new Command("CloseStream", 1, 1);
        exitCmd = new Command("Exit", 1, 2);
        fmAnswer.addCommand(testCmd);

        fmAnswer.addCommand(exitCmd);
        fmAnswer.append(siRTPText);
        fmAnswer.append(siRTPText2);
        fmAnswer.setCommandListener(this);
        myDisplay = Display.getDisplay(this);
    }
}
```



```

myDisplay.setCurrent(fmAnswer);
System.out.println("before new instance");
Amanager = RTPManager.newInstance();
System.out.println("after new instance");
}
public void startApp() {
    RTPContentDescriptor mpegcd = new RTPContentDescriptor("video/mpeg",8100,100);
    RTPContentDescriptor xwavcd = new RTPContentDescriptor("audio/x-wav",8100,100);
    System.out.println("before addFormat");
    Amanager.addFormat(mpegcd,99);
    Amanager.addFormat(xwavcd,100);
    System.out.println("after addFormat");
    //localaddr= new SessionAddress("127.0.0.1", 55000);
    //localaddr= new SessionAddress("127.0.0.1", 55000);
    localaddr= new SessionAddress("173.49.50.117", 55000);
    //remoteaddr = new SessionAddress("127.0.0.1", 55000);
    remoteaddr = new SessionAddress("173.49.50.117", 55000);
    System.out.println("after construct session Address");
    try {
        System.out.println("here to init");
        System.out.println(localaddr.getIPAddr());
        Amanager.initialize(localaddr);
        siRTPText.setText("RTP Initialized");
    }catch (Exception ex) {
        System.out.println("error");
        ex.printStackTrace();
    }
    try {
        System.out.println("before addtarget");
        Amanager.addTarget(remoteaddr);
        System.out.println("after addtarget");
    }catch(Exception ex) {
        //System.out.println("addtarget error");
        //System.out.println(ex.toString());
    }
}

public void pauseApp() {
}
public void destroyApp(boolean unconditional) {
}

public void update(SendStreamEvent ev) {
    System.out.println("send stream event got");
    if (ev instanceof NewSendStreamEvent) {
        //siRTPText.setText("begin to send now");
        System.out.println("here are new send stream");
    }
    if(ev instanceof StreamClosedEvent) {
        siRTPText.setText("send closed");
        System.out.println("here are closed send stream");
    }
}

```



```

}

public void update(ReceiveStreamEvent ev) {
    System.out.println("receive stream event got");
    if (ev instanceof NewReceiveStreamEvent) {
        System.out.println("here are new receive stream");
        ReceiveStream receiveStream = ev.getReceiveStream();
        DataSource ds = receiveStream.getDataSource();
        try{
            ds.start();
        }catch(Exception e)
        {
        }
        SourceStream[] streams = ds.getStreams();
        System.out.println("begin to start thread to receive");
        siRTPText2.setText("begin to receive");
        new Thread(new datasourceReader(streams[0],this)).start();
    }
}

public void commandAction(Command command, Displayable displayable)
{
    int fileLen = 0;
    if(command == testCmd)
    {
        try{
            RandomAccessFile inFile = new
RandomAccessFile("ChatInvt.wav",com.mot.cldc.io.File.R_RSC,"r",new
SecurityToken(SecurityToken.SM_MIDLET_TOKEN));
            fileLen = inFile.length();
            inFile.close();
        }catch(Exception e) {
            System.out.println(e.toString());
        }
        String uri="file://ChatInvt.wav";
        FileDataSource fds = new FileDataSource();
        fds.setLocator(uri);
        //fds.connect();
        Amanager.addSendStreamListener(this);
        Amanager.addReceiveStreamListener(this);
        try {
            fds.connect();
            sendStream = Amanager.createSendStream(fds,1);
            System.out.println("after create sendStream");
            sendStream.start();
        }catch(Exception ex2) {
            System.out.println("create sendStream error " +ex2.toString());
        }
        siRTPText.setText("pls check receive, send file's length is " + fileLen);
        fmAnswer.removeCommand(testCmd);
        fmAnswer.addCommand(closeCmd);
    }
}

```




```

        if(command == closeCmd)
        {
            System.out.println("begin to close");
            sendStream.close();
            //Amanager.removeTargets(null);
            System.out.println("after to close");
            fmAnswer.removeCommand(closeCmd);
            fmAnswer.addCommand(testCmd);
        }
        if(command == exitCmd)
        {
            Amanager.removeTargets(null);
            Amanager.dispose();
            destroyApp(false);
            notifyDestroyed();
        }
    }

class datasourceReader implements Runnable {
    private SourceStream stream = null;
    private RandomAccessFile outFile;
    private SendAndRecvTest4 test;
    datasourceReader(SourceStream stream, SendAndRecvTest4 test){
        this.stream=stream;
        this.test = test;
    }
    public void run() {
        System.out.println("begin to read");
        int len=-1;
        int recv_count = 0;
        byte[] buf = new byte[256];
        while(true) {
            try {
                len = stream.read(buf, 0, buf.length);
                //System.out.println("here come data");
                recv_count = recv_count + len;
                test.siRTPText2.setText("have receive " + recv_count+ " bytes");
                //System.out.println("Test Case: receive " + recv_count+ " bytes");
                //outFile.write(buf,0,len);
            }catch(Exception e) {
                System.out.println(e.toString());
                continue;
            }
        }
    }
}

```



The following examples are DSR & RTP integration examples

```
import javax.microedition.midlet.*;
import com.motorola.iden.media.rtp.*;
import com.motorola.iden.media.rtp.event.*;
import javax.microedition.lcdui.*;
import com.motorola.iden.media.protocol.*;
import com.mot.cldc.io.*;
import com.mot.security.*;
import javax.microedition.media.protocol.*;
import com.motorola.iden.speech.recognition.dsr.*;

public class SendDSRData extends javax.microedition.midlet.MIDlet implements
CommandListener, SendStreamListener {
    RTPManager Amanager;
    SessionAddress localaddr;
    SessionAddress remoteaddr;
    private Display myDisplay;
    Form fmAnswer;
    public StringItem siRTPText;
    public StringItem siRTPText2;
    Command testCmd;
    Command closeCmd;
    Command exitCmd;
    SendStream sendStream = null;
    DSRDataSource dsr = null;

    public SendDSRData() {
        fmAnswer = new Form("RTP TEST");
        siRTPText = new StringItem("RTP Send Status:", "");
        siRTPText2 = new StringItem("RTP Recv Status:", "");
        testCmd = new Command("Test", 1, 1);
        closeCmd = new Command("CloseStream", 1, 1);
        exitCmd = new Command("Exit", 1, 2);
        fmAnswer.addCommand(testCmd);
        fmAnswer.addCommand(exitCmd);
        fmAnswer.append(siRTPText);
        fmAnswer.append(siRTPText2);
        fmAnswer.setCommandListener(this);
        myDisplay = Display.getDisplay(this);
        myDisplay.setCurrent(fmAnswer);
        Amanager = RTPManager.newInstance();
    }

    public void startApp() {
        RTPContentDescriptor mpegcd = new RTPContentDescriptor("video/mpeg", 8100, 100);
        RTPContentDescriptor xwavcd = new RTPContentDescriptor("audio/dsr", 8100, 100);
        //System.out.println("here cd"+cd.getContentype());
        Amanager.addFormat(mpegcd, 100);
        Amanager.addFormat(xwavcd, 99);
        //localaddr= new SessionAddress("127.0.0.1", 55000);
        //localaddr= new SessionAddress("10.23.6.1", 55000);
        int port = 55000;
```



```

localaddr= new SessionAddress("173.49.50.117", port);
remoteaddr = new SessionAddress("173.49.50.117", port);
try {
    System.out.println("here to init");
    System.out.println(localaddr.getIPAddr());
    Amanager.initialize(localaddr);
    siRTPText.setText("RTP Initialized");
} catch (Exception ex) {
    System.out.println("error");
    ex.printStackTrace();
}
try {
    System.out.println("before addtarget");
    Amanager.addTarget(remoteaddr);
    System.out.println("after addtarget");
} catch (Exception ex) {
    //System.out.println("addtarget error");
    //System.out.println(ex.toString());
}
}

public void pauseApp() {
}
public void destroyApp(boolean unconditional) {
}

public void update(SendStreamEvent ev) {
    System.out.println("send stream event got");
    if (ev instanceof NewSendStreamEvent) {
        //siRTPText.setText("begin to send now");
        System.out.println("here are new send stream");
    }
    if (ev instanceof StreamClosedEvent) {
        siRTPText.setText("send closed");
        System.out.println("here are closed send stream");
    }
}

public void commandAction(Command command, Displayable displayable)
{
    int fileLen = 0;
    if(command == testCmd)
    {
        dsr = new DSRDataSource();
        Amanager.addSendStreamListener(this);
        try {
            dsr.connect();
            sendStream = Amanager.createSendStream(dsr,1);
            System.out.println("after create sendStream");
            sendStream.start();
        } catch (Exception ex2) {

```



```

        System.out.println("create sendStream error " + ex2.toString());
    }
    fmAnswer.removeCommand(testCmd);
    fmAnswer.addCommand(closeCmd);
    }
    if(command == closeCmd)
    {
        System.out.println("begin to close");
        sendStream.close();

        dsr.disconnect();
        //Amanager.removeTargets(null);
        System.out.println("after to close");
        fmAnswer.removeCommand(closeCmd);
        fmAnswer.addCommand(testCmd);
    }
    if(command == exitCmd)
    {
        Amanager.removeTargets(null);
        Amanager.dispose();
        destroyApp(false);
        notifyDestroyed();
    }
    }
}

```

The following examples are DSR & RTP integration examples

```

import javax.microedition.midlet.*;
import com.motorola.iden.media.rtp.*;
import com.motorola.iden.media.rtp.event.*;
import javax.microedition.lcdui.*;
import com.motorola.iden.media.protocol.*;
import com.mot.cldc.io.*;
import com.mot.security.*;
import javax.microedition.media.protocol.*;
import com.motorola.iden.speech.recognition.dsr.*;

public class SendDSRData extends javax.microedition.midlet.MIDlet implements
CommandListener, SendStreamListener {
    RTPManager Amanager;
    SessionAddress localaddr;
    SessionAddress remoteaddr;
    private Display myDisplay;
    Form fmAnswer;
    public StringItem siRTPText;
    public StringItem siRTPText2;
    Command testCmd;
    Command closeCmd;
    Command exitCmd;
    SendStream sendStream = null;

```



```

DSRDataSource dsr = null;

public SendDSRData() {
    fmAnswer = new Form("RTP TEST");
    siRTPText = new StringItem("RTP Send Status:", "");
    siRTPText2 = new StringItem("RTP Recv Status:", "");
    testCmd = new Command("Test", 1, 1);
    closeCmd = new Command("CloseStream", 1, 1);
    exitCmd = new Command("Exit", 1, 2);
    fmAnswer.addCommand(testCmd);
    fmAnswer.addCommand(exitCmd);
    fmAnswer.append(siRTPText);
    fmAnswer.append(siRTPText2);
    fmAnswer.setCommandListener(this);
    myDisplay = Display.getDisplay(this);
    myDisplay.setCurrent(fmAnswer);
    Amanager = RTPManager.newInstance();
}

public void startApp() {
    RTPContentDescriptor mpegcd = new RTPContentDescriptor("video/mpeg", 8100, 100);
    RTPContentDescriptor xwavcd = new RTPContentDescriptor("audio/dsr", 8100, 100);
    //System.out.println("here cd"+cd.getContentTypes());
    Amanager.addFormat(mpegcd, 100);
    Amanager.addFormat(xwavcd, 99);
    //localaddr= new SessionAddress("127.0.0.1", 55000);
    //localaddr= new SessionAddress("10.23.6.1", 55000);
    int port = 55000;
    localaddr= new SessionAddress("173.49.50.117", port);
    remoteaddr = new SessionAddress("173.49.50.117", port);
    try {
        System.out.println("here to init");
        System.out.println(localaddr.getIPAddr());
        Amanager.initialize(localaddr);
        siRTPText.setText("RTP Initialized");
    } catch (Exception ex) {
        System.out.println("error");
        ex.printStackTrace();
    }
    try {
        System.out.println("before addtarget");
        Amanager.addTarget(remoteaddr);
        System.out.println("after addtarget");
    } catch (Exception ex) {
        //System.out.println("addtarget error");
        //System.out.println(ex.toString());
    }
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional) {

```



```
}

public void update(SendStreamEvent ev) {
    System.out.println("send stream event got");
    if (ev instanceof NewSendStreamEvent) {
        //siRTPText.setText("begin to send now");
        System.out.println("here are new send stream");
    }
    if(ev instanceof StreamClosedEvent) {
        siRTPText.setText("send closed");
        System.out.println("here are closed send stream");
    }
}

}

public void commandAction(Command command, Displayable displayable)
{
    int fileLen = 0;
    if(command == testCmd)
    {
        dsr = new DSRDataSource();
        Amanager.addSendStreamListener(this);
        try {
            dsr.connect();
            sendStream = Amanager.createSendStream(dsr,1);
            System.out.println("after create sendStream");
            sendStream.start();
        }catch(Exception ex2) {
            System.out.println("create sendStream error " +ex2.toString());
        }
        fmAnswer.removeCommand(testCmd);
        fmAnswer.addCommand(closeCmd);
    }
    if(command == closeCmd)
    {
        System.out.println("begin to close");
        sendStream.close();

        dsr.disconnect();
        //Amanager.removeTargets(null);
        System.out.println("after to close");
        fmAnswer.removeCommand(closeCmd);
        fmAnswer.addCommand(testCmd);
    }
    if(command == exitCmd)
    {
        Amanager.removeTargets(null);
        Amanager.dispose();
        destroyApp(false);
        notifyDestroyed();
    }
}
}
```



```
}
```

```
import javax.microedition.midlet.*;
import com.motorola.iden.media.rtp.*;
import com.motorola.iden.media.rtp.event.*;
import javax.microedition.lcdui.*;
import com.motorola.iden.media.protocol.*;
import com.mot.cldc.io.*;
import com.mot.security.*;
import javax.microedition.media.protocol.*;
import com.motorola.iden.speech.recognition.dsr.*;

public class SendDSRData2 extends javax.microedition.midlet.MIDlet implements
CommandListener, SendStreamListener, DSRLListener {
    RTPManager Amanager;
    SessionAddress localaddr;
    SessionAddress remoteaddr;
    private Display myDisplay;
    Form fmAnswer;
    public StringItem siRTPText;
    public StringItem siRTPText2;
    Command testCmd;
    Command closeCmd;
    Command exitCmd;
    SendStream sendStream = null;
    DSRDataSource dsr = null;

    public SendDSRData2() {
        fmAnswer = new Form("RTP TEST");
        siRTPText = new StringItem("RTP Send Status:", "");
        siRTPText2 = new StringItem("RTP Recv Status:", "");
        testCmd = new Command("Test", 1, 1);
        closeCmd = new Command("CloseStream", 1, 1);
        exitCmd = new Command("Exit", 1, 2);
        fmAnswer.addCommand(testCmd);
        fmAnswer.addCommand(exitCmd);
        fmAnswer.append(siRTPText);
        fmAnswer.append(siRTPText2);
        fmAnswer.setCommandListener(this);
        myDisplay = Display.getDisplay(this);
        myDisplay.setCurrent(fmAnswer);
        Amanager = RTPManager.newInstance();

        RTPContentDescriptor mpegcd = new RTPContentDescriptor("video/mpeg", 8100, 100);
        RTPContentDescriptor xwavcd = new RTPContentDescriptor("audio/dsr", 8100, 100);
        Amanager.addFormat(mpegcd, 100);
        Amanager.addFormat(xwavcd, 99);
        int port = 55000;
        localaddr = new SessionAddress("173.49.50.117", port);
        remoteaddr = new SessionAddress("173.49.50.117", port);
        try {
```



```

        System.out.println("here to init");
        System.out.println(localaddr.getIPAddr());
        Amanager.initialize(localaddr);
        siRTPText.setText("RTP Initialized");
    }catch (Exception ex) {
        System.out.println("error");
        ex.printStackTrace();
    }
    try {
        System.out.println("before addtarget");
        Amanager.addTarget(remoteaddr);
        System.out.println("after addtarget");
    }catch (Exception ex) {
        System.out.println("addtarget error");
        System.out.println(ex.toString());
    }
}
public void startApp() {
}

public void pauseApp() {
}
public void destroyApp(boolean unconditional) {
}

public void processEvent(String event)
{
    if (event.compareTo(DSRListener.DSR_AVAILABLE)==0)
    {
        try
        {
            System.out.println("Receive Event DSR_AVAILABLE");
            sendStream.start();
            System.out.println("SendStream Start Again!");
            fmAnswer.addCommand(exitCmd);
        }catch (Exception ex2)
        {
            System.out.println("start sendStream error " + ex2.toString());
        }
    }

    if (event.compareTo(DSRListener.DSR_STOPPED)==0)
    {
        System.out.println("Receive Event DSR_STOPPED");
    }
}

public void update(SendStreamEvent ev) {
    System.out.println("send stream event got");
    if (ev instanceof NewSendStreamEvent) {
        System.out.println("here are new send stream");
    }
}

```




```
        if(ev instanceof StreamClosedEvent) {
            siRTPText.setText("send closed");
            System.out.println("here are closed send stream");
        }
    }

    public void commandAction(Command command, Displayable displayable)
    {
        int fileLen = 0;
        if(command == testCmd)
        {
            dsr = new DSRDataSource();
            Amanager.addSendStreamListener(this);
            dsr.setDSRListener(this);

            try {
                System.out.println("DSR connect!");
                dsr.connect();
                System.out.println("createSendStream!");
                sendStream = Amanager.createSendStream(dsr, 1);
                System.out.println("after create sendStream");
                sendStream.start();
            } catch (Exception ex2) {
                System.out.println("create sendStream error " + ex2.toString());
            }
            fmAnswer.removeCommand(testCmd);
            fmAnswer.addCommand(closeCmd);
        }
        if(command == closeCmd)
        {
            System.out.println("begin to close");
            sendStream.close();

            dsr.disconnect();
            System.out.println("after to close");
            fmAnswer.removeCommand(closeCmd);
            fmAnswer.addCommand(testCmd);
        }
        if(command == exitCmd)
        {
            Amanager.removeTargets(null);
            Amanager.dispose();
            destroyApp(false);
            notifyDestroyed();
        }
    }
}
```



4.12.4 Q & A

1. Q: Why doesn't my `ReceiveStreamListener` take effect?

A: make sure you have called `RTPManager`'s `initialize` and `addTarget` before calling `addReceiveStreamListener` or `addSendStreamListener`. Otherwise callback methods will not be invoked.

2. Q: Can I call `RTPManager`'s `newInstance` twice?

A: On a second call to `newInstance`, the return value will be null. Only after the previous `RTPManager` object has been disposed will subsequent calls to `newInstance` return a new `RTPManager` object.

3. Q: Why does `addTarget` throw `InvalidSessionAddressException` with session addresses such as 192.13.12.2 30003?

A: The RTP protocol, requires that the port number be an even number.

4. Q: What is the correct sequence of using `RTPManager`?

A:

- First construct an `RTPManager` Object.

```
RTPManager Amanager=RTPManager.newInstance();
```

- Create the local endpoint for the local interface on a designed port 5000 or any valid port.

```
SessionAddress localaddr=new SessionAddress(A.ipaddress, 5000);
```

- Initialize the RTP session.

```
Amanager.initialize(localaddr);
```

- If client A wants to receive host B's data through RTP, A will do the following:

```
Amanager.addTarget(B_ip_address);
```

```
Amanager.addReceiveStreamListener(A);
```



- If client A wants to send datasource's 1st stream to the RTP session, A will construct a SendStream.

```
SendStream send=Amanager.createSendStream(A_dataSource, 1);
```

- A starts transmitting.
- A stops transmitting.
- Send.stop();
- A exits the RTP session.

```
Send.close();
```

```
Amanager.dispose();
```

Host B's operation is similar to A.

5. Q: How to transmit dynamic payload data between A and B?

A: They should register the dynamic payload media format and its payload number before RTPManager.initialize. Suppose the media payload names "media-a&b", the payload number is 100, the timestamp unit is 1.

```
RTPContentDescriptor dynamic_payload=new RTPContentDescriptor("media-a&b",8000,40);
```

```
Amanager.addFormat(dynamic_payload, 100)
```

6. Q: How does A get DataSource from ReceiveStream?

A: A can implement the ReceiveStreamListener as below:

```
update(ReceiveStreamEvent e){
    if(e instanceof NewReceiveStreamEvent) {
        ReceiveStream rcvStream=e.getReceiveStream();
        DataSource ds=rcvStream.getDataSource();
        // do application stuff
        .....
    }
}
```



```
}
```

7. Q: What is the correct sequence of using DSR & RTP APIs when using RTP to send DSR FE data?

- First construct an RTPManager Object.

```
RTPManager Amanager=RTPManager.newInstance();
```
- Create the local endpoint for the local interface on a valid port.

```
SessionAddress localaddr=new SessionAddress(A.ipaddress, 5000);
```
- Initialize the RTP session.

```
Amanager.initialize(localaddr);
```
- If client A want to receive host B's data through RTP, A will do the following:

```
Amanager.addTarget(B_ip_address);
```

```
Amanager.addReceiveStreamListener(A);
```
- Create a DSRDataSource stream.

```
dsr = new DSRDataSource();
```

```
dsr.connect();
```
- Send DSR stream to the RTP session and construct a SendStream.

```
SendStream send=Amanager.createSendStream(dsr, 1);
```
- A starts transmitting

```
Send.start();
```
- A stops transmitting.

```
Send.stop();
```
- A exits the RPT session and disconnects the DSRDataSource.

```
Send.close();
```

```
dsr.disconnect();
```

```
Amanager.dispose();
```

Note: When a MIDlet uses RTP to send out DSR FE data, it should regard DSR media as a kind of DataSource and only connect/disconnect DSR DataSource. It should not call `start()` or `stop()` DSR engine or `read()` to get DSR FE data. All these APIs are called by RTP APIs.



4.13 Distributed Speech Recognition

4.13.1 Overview



This API is only available on this handset.

A DSR system uses an error protected data channel to send a parameterized representation of speech, which is suitable for recognition. The processing is distributed between the terminal and the network. The terminal is the front-end of the speech recognition system and performs the feature parameter extraction. These features are transmitted over a data channel to a remote, back-end recognizer. An RTP channel or other data channel can be used to transmit that data. DSR FE data can be regarded as a custom media type. Mobile Media API Specification (JSR 135) supports custom media types.

4.13.2 Class Description

The DSR API is located in package `com.motorola.iden.speech.recognition.dsr`

Interface Summary

DSRListener	Interface that specifies callbacks for suspend/resume events.
--------------------	---

Class Summary

DSRDataSource	<p>A DSRDataSource is an abstraction for media DSR protocol-handlers. It hides the details of how the data is read from source--whether the data is coming from a Microphone or a voice file. It provides methods to access the input data.</p> <p>A DSRDataSource contains one SourceStream because only one DSR engine can be used at one time. One SourceStream represents one elementary data stream of the source.</p> <p>DSRDataSource manages the life-cycle of the media source by providing a simple connection protocol.</p>
----------------------	--



Package Tree

Class Hierarchy

The following will be the class hierarchy for the DSR API:

- class javax.microedition.media.protocol.DataSource
 - class com.motorola.iden.speech.recognition.dsr.**DSRDataSource**
- Interface com.motorola.iden.speech.recognition.dsr.**DSRListener**

4.13.2.1 Class DSRDataSource

4.13.2.1.1 connect

public void **connect**() throws java.io.IOException

Open a connection to the DSR engine.

Throws java.io.IOException - if the DSR engine has already been connected.

Note: The DSR engine on the handset only supports one connection at any time. Multiple MIDlets cannot access the DSR engine concurrently.

4.13.2.1.2 disconnect

public void **disconnect**()

Close the connection to the DSR engine source and free resources which are used to maintain the connection. If no resources are in use, disconnect is ignored. If stop hasn't already been called, calling disconnect implies a stop.

4.13.2.1.3 start

public void **start**() throws java.io.IOException, java.lang.IllegalStateException

Start the DSR engine and initiates data-transfer. The start method must be called before data is available for reading.

Throws java.lang.IllegalStateException - if the DSRDataSource is not connected.

Throws java.io.IOException - if the DSRDataSource is already started.



4.13.2.1.4 stop

`public void stop()` throws `java.io.IOException`

Stop the DSR engine and stop data-transfer. If the `DSRDataSource` has not been connected and started, stop is ignored.

Throws `java.io.IOException` - if the `DSRDataSource` can not be stopped successfully

4.13.2.1.5 getStreams

`public javax.microedition.media.protocol.SourceStream[] getStreams()`

throws `java.lang.IllegalStateException`

Get the collection of streams that this source manages. The collection of streams is entirely content dependent. The MIME type of this `DSRDataSource` provides the only indication of what streams may be available on this connection.

Returns:

The collection of streams for this source

Throws `java.lang.IllegalStateException` - if the source is not connected.

Tips

If the DSR engine is disconnected, any streams retrieved from a previous connection should not be used. When the DSR engine is connected again, new streams will be created. Call `getStreams()` to get the most current streams.

4.13.2.1.6 getControls

`public javax.microedition.media.Control[] getControls()`

Obtain the collection of Controls from the object that implements `Controllable` interface. If no Control is supported, a zero length array is returned.

Returns:

The collection of Control objects.

The `DSRDataSource` class does not support any Controls. Calling `getControls()` will always return a zero length array.



4.13.2.1.7 getControl

```
public javax.microedition.media.Control getControl(java.lang.String controlType)
```

Obtain the object that implements the specified Control interface. If the specified Control interface is not supported then null is returned. If the Controllable supports multiple objects that implement the same specified Control interface, only one of them will be returned. To obtain all the Control's of that type, use the getControls method and check the list for the requested type. The DSRDataSource class does not support any Controls. This method always returns null.

Parameters:

controlType - the class name of the Control.

Returns:

null.

4.13.2.1.8 setDSRPackets

```
public void setDSRPackets(byte packets)
```

throws java.lang.IllegalStateException,
 java.lang.IllegalArgumentException

Set the number of DSR packets that should be buffered before the DSR engine sends them to the MIDlet. The amount of DSR packets to buffer should be set after the DSR engine is connected but before it is started. If the DSR packet buffer is not set, the buffer size is set to the maximum supported by the DSR engine.

Parameters:

packets - number of DSR packets the engine should buffer

4.13.2.1.9 getMaxPackets

```
public byte getMaxPackets()
```

Get the maximum number of packets that can be buffered by the DSR engine.

Returns:

the maximum number of packets that can be buffered by the DSR engine.



4.13.2.1.10 setDSRListener

```
public void setDSRListener(DSRListener listener)
```

Sets DSRListener for this DSRDataSource. If a MIDlet calls setDSRListener twice, the previous listener will be replaced by the most recent. To remove an existing listener, call setDSRListener(null).

Parameters:

Listener - the DSRListener for this DSRDataSource or null to remove an existing listener

4.13.2.1.11 setNullFrames

```
public void setNullFrames(byte nullFrames)
```

throws IllegalArgumentException

Set how many number of null frames will be constructed when DSR is stopped.

Default null frames is 1.

Parameter:

nullFrames – number of null frames.

Tips

Throw IllegalArgumentException if the nullFrames parameter is larger than max number of null frames or less than 1. To get max null frames, call getMaxNullFrames().

4.13.2.1.12 getMaxNullFrames

```
public byte getMaxNullFrames()
```

Get the max number of null frames which will be constructed when DSR is stopped.

Returns:

the max number of null frames.



4.13.2.2 Interface DSRListener

Interface that generates the callback for suspend/resume events.

4.13.2.2.1 processEvent

```
public void processEvent(String event);
```

Callback method for notification of suspend and resume events.

Parameter:

event – String indicating suspend/resume events.

4.13.3 Code Example

This is an example of using DSR API.

```
import com.motorola.iden.speech.recognition.dsr.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.microedition.media.*;
import javax.microedition.media.protocol.*;
import javax.microedition.media.Controllable;
import java.io.IOException;

public class ReadNullFrameTest extends MIDlet implements CommandListener,
    DSRListener
{
    DSRDataSource      dsr = null;
    SourceStream        stream = null;
    ContentDescriptor   dsrContentDescriptor = null;
    int                 bufLen = 0;

    Form                fmAnswer;
    Command              exitCmd;
    Command              testCmd;
    public StringItem    siRTPTText;
    private Display      myDisplay;

    public ReadNullFrameTest()
    {

```



```
dsr = new DSRDataSource();
try
{
    dsr.connect();
}
catch(Exception e)
{
    System.out.println(e);
}

fmAnswer = new Form("DSR TEST");
siRTPTText = new StringItem("DSR Test Status:", "");
fmAnswer.append(siRTPTText);
siRTPTText.setText("BasicTest has Finished!");

testCmd = new Command("Test", 1, 1);
exitCmd = new Command("Exit", 1, 2);

fmAnswer.addCommand(testCmd);
fmAnswer.addCommand(exitCmd);

fmAnswer.setCommandListener(this);
myDisplay = Display.getDisplay(this);
myDisplay.setCurrent(fmAnswer);
}

public void startApp()
{
}

public void pauseApp()
{
}
```



```
public synchronized void destroyApp(boolean flag)
{
}

public void processEvent(String event)
{
    if (event.compareTo(DSRListener.DSR_AVAILABLE)==0)
    {
        siRTPText.setText("MIDlet receive Event DSR_AVAILABLE");
        fmAnswer.addCommand(testCmd);
        fmAnswer.addCommand(exitCmd);
    }

    if (event.compareTo(DSRListener.DSR_STOPPED)==0)
    {
        System.out.println("MIDlet receive Event DSR_STOPPED");
    }
}

public void commandAction(Command command, Displayable displayable)
{
    if(command == testCmd)
    {
        dsr.setDSRListener(this);
        stream = dsr.getStreams()[0];

        byte packets = 1;
        dsr.setDSRPackets(packets);

        try
        {
            dsr.start();
        }
        catch (Exception e)
        {
        }
    }
}
```



```
System.out.println(e);
    }

    DSRReader dsrReader = new DSRReader(stream);
    new Thread(dsrReader).start();

    fmAnswer.removeCommand(testCmd);
    fmAnswer.addCommand(exitCmd);
}

if(command == exitCmd)
{
    try
    {
        dsr.stop();
        dsr.disconnect();
    }
    catch (Exception e)
    {
        System.out.println(e);
    }

    destroyApp(false);
    notifyDestroyed();
}
}

class DSRReader implements Runnable
{
    SourceStream stream;

    public DSRReader(SourceStream stream)
    {
        this.stream = stream;
    }
}
```



```
}

public void run()
{
    int    bytesRead = 0;
    int    i = 0;
    int    j = 0;
    int    numOfNullFrames = 0;
    int    bufLen = stream.getTransferSize();
    byte[] dsrBuf = new byte[bufLen];

    while(true)
    {
        try
        {
            bytesRead = stream.read(dsrBuf,0,bufLen);
            i++;
            System.out.println("Read DSR Data Block " + i
+ ", bytes" + bytesRead);

            boolean isNullFrame = true;
            for (j=0; j<bytesRead; j++)
            {
                if (dsrBuf[j] != 0)
                {
                    isNullFrame = false;
                    break;
                }
            }

            if (isNullFrame)
            {
                System.out.println("*****Get 1 Null Frame!*****");
                numOfNullFrames++;
            }
        }
    }
}
```



```
        }  
    catch (Exception e)  
    {  
        System.out.println(e);  
        System.out.println("*****DSR engine stopped!  
numOfNullFrames = " + numOfNullFrames + "*****");  
        break;  
    }  
}  
}
```



The following illustrates DSR & RTP integration.

```
import javax.microedition.midlet.*;
import com.motorola.iden.media.rtp.*;
import com.motorola.iden.media.rtp.event.*;
import javax.microedition.lcdui.*;
import com.motorola.iden.media.protocol.*;
import com.mot.cldc.io.*;
import com.mot.security.*;
import javax.microedition.media.protocol.*;
import com.motorola.iden.speech.recognition.dsr.*;

public class SendDSRData extends javax.microedition.midlet.MIDlet implements
    CommandListener, SendStreamListener {
    RTPManager Amanager;
    SessionAddress localaddr;
    SessionAddress remoteaddr;
    private Display myDisplay;
    Form fmAnswer;
    public StringItem siRTPText;
    public StringItem siRTPText2;
    Command testCmd;
    Command closeCmd;
    Command exitCmd;
    SendStream sendStream = null;
    DSRDataSource dsr = null;

    public SendDSRData() {
        fmAnswer = new Form("RTP TEST");
        siRTPText = new StringItem("RTP Send Status:", "");
        siRTPText2 = new StringItem("RTP Recv Status:", "");
        testCmd = new Command("Test", 1, 1);
        closeCmd = new Command("CloseStream", 1, 1);
        exitCmd = new Command("Exit", 1, 2);
        fmAnswer.addCommand(testCmd);
```




```
fmAnswer.addCommand(exitCmd);
fmAnswer.append(siRTPText);
fmAnswer.append(siRTPText2);
fmAnswer.setCommandListener(this);
myDisplay = Display.getDisplay(this);
    myDisplay.setCurrent(fmAnswer);
    Amanager = RTPManager.newInstance();

}

public void startApp() {
    RTPContentDescriptor      mpegcd      =      new
    RTPContentDescriptor("video/mpeg",8100,100);

    RTPContentDescriptor      xwavcd      =      new
    RTPContentDescriptor("audio/dsr",8100,100);

        //System.out.println("here cd"+cd.getContentType());
        Amanager.addFormat(mpegcd,100);
        Amanager.addFormat(xwavcd,99);

    //localaddr= new SessionAddress("127.0.0.1", 55000);
    //localaddr= new SessionAddress("10.23.6.1", 55000);
    int port = 55000;
    localaddr= new SessionAddress("173.49.50.117", port);
    remoteaddr = new SessionAddress("173.49.50.117", port);
    try {
        System.out.println("here to init");
        System.out.println(localaddr.getIPAddr());
        Amanager.initialize(localaddr);
        siRTPText.setText("RTP Initialized");
    }catch (Exception ex) {
        System.out.println("error");
        ex.printStackTrace();
    }
    try {
        System.out.println("before addtarget");
        Amanager.addTarget(remoteaddr);
        System.out.println("after addtarget");
    }catch(Exception ex) {
```



```
//System.out.println("addtarget error");
//System.out.println(ex.toString());
}
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional) {
}

public void update(SendStreamEvent ev) {
    System.out.println("send stream event got");
    if (ev instanceof NewSendStreamEvent) {
        //siRTPText.setText("begin to send now");
        System.out.println("here are new send stream");
    }
    if(ev instanceof StreamClosedEvent) {
        siRTPText.setText("send closed");
        System.out.println("here are closed send stream");
    }
}

public void commandAction(Command command, Displayable displayable)
{
    int fileLen = 0;
    if(command == testCmd)
    {
        dsr = new DSRDataSource();
        Amanager.addSendStreamListener(this);
        try {
            dsr.connect();
            sendStream = Amanager.createSendStream(dsr,1);
            System.out.println("after create sendStream");
            sendStream.start();
        }
    }
}
```



```

        }catch(Exception ex2) {
            System.out.println("create      sendStream      error      "
+ex2.toString());
        }
        fmAnswer.removeCommand(testCmd);
        fmAnswer.addCommand(closeCmd);
    }
    if(command == closeCmd)
    {
        System.out.println("begin to close");
        sendStream.close();

        dsr.disconnect();
        //Amanager.removeTargets(null);
        System.out.println("after to close");
        fmAnswer.removeCommand(closeCmd);
        fmAnswer.addCommand(testCmd);
    }
    if(command == exitCmd)
    {
        Amanager.removeTargets(null);
        Amanager.dispose();
        destroyApp(false);
        notifyDestroyed();
    }
}
}
}

```

4.13.4 Using DSR With RTP

- First construct an RTPManager Object

```
RTPManager Amanager = RTPManager.newInstance();
```

- Create the local endpoint for the local interface on any valid port

```
SessionAddress localaddr = new SessionAddress(A_ip_address, 5000);
```



- Initialize the RTP session

```
Amanager.initialize(localaddr);
```

- If A wants to receive B's data through RTP and do the application stuff, A will do the following:

```
Amanager.addTarget(B_ip_address);
```

```
Amanager.addReceiveStreamListener(A);
```

- Create a DSRDataSource stream;

```
dsr = new DSRDataSource();
```

```
dsr.connect();
```

- Send DSR stream to the RTP session and construct a SendStream:

```
SendStream send=Amanager.createSendStream(dsr, 1);
```

- A starts transmitting

```
Send.start();
```

- A stops transmitting

```
Send.stop();
```

- A exits the RTP session and disconnects DSRDataSource.

```
Send.close();
```

```
dsr.disconnect();
```

```
Amanager.dispose();
```

Note: When a MIDlet uses RTP to send DSR FE data, it should regard DSR media as a kind of DataSource and only connect/disconnect DSR DataSource. It should not call start() or stop() on the DSR engine or read() to get DSR FE data. All these APIs are called by the RTP APIs.



4.14 Lighting

4.14.1 Overview

The Lighting API lets a MIDlet turn on and off various lights on the phone.

4.14.2 Class Description

The Lighting API is located in package `com.mot.iden.multimedia`

```
java.lang.Object
|
+ - com.mot.iden.multimedia.Lighting
```

4.14.3 Method Description

4.14.3.1 Lighting Methods

4.14.3.1.1 setLighting

Sets the specified light to the specified state

```
public static void setLighting(int light, int state)
    throws IllegalStateException
```

The table below lists the valid values for a light and state.

Valid Values for Light and State

Light	State
LIGHT_DISPLAY	LIGHT_STATE_ON / LIGHT_STATE_OFF
LIGHT_KEYPAD	LIGHT_STATE_ON / LIGHT_STATE_OFF
LIGHT_STATUS	LIGHT_STATE_OFF LIGHT_STATE_RED LIGHT_STATE_GREEN LIGHT_STATE_AMBER
LIGHT_CALL_INDICATOR	LIGHT_STATE_OFF LIGHT_STATE_RED LIGHT_STATE_GREEN LIGHT_STATE_BLUE LIGHT_STATE_YELLOW LIGHT_STATE_MAGENTA LIGHT_STATE_CYAN LIGHT_STATE_WHITE



If you pass an invalid value for either light or state, this method throws an `IllegalStateException`.



These models do not have a status light, but have a call indicator light instead. If you pass `LIGHT_STATUS` as the value for light, this method changes the state of the call indicator light. This lets you continue to run older applications that use the status light without needing to modify them.



This model does not have a status light, but its external display backlight serves as a call indicator light instead. If you pass `LIGHT_STATUS` as the value for light, this method changes the state of the call indicator light. This lets you continue to run older applications that use the status light without needing to modify them.



These models have no status or call indicator light. Setting `LIGHT_STATUS` on these handsets has no effect.

4.14.3.1.2 `javaOverrideLighting`

Gives an application complete control of the device's lights.

```
public static void javaOverrideLighting(boolean state)
```

If state is true, this application is totally responsible for managing the device's lights while the application has focus and is in control of the display. The phone itself does not change the state of the lights.

If state is false, then the phone controls the state of the lights. The application can change the state of a light with `setLighting()`, but the phone can change it at any time.

Note that this setting is reset to false when an application loses focus on a display. If the application wishes to override this setting, it has the opportunity to do so when it regains control of the display again.



4.14.3.1.3 **getPhotoSensorLevel**

Returns the current amount of ambient light according to the photo sensor.

```
public static int getPhotoSensorLevel()
```

The returned value ranges from 0 to 255, with 0 being no ambient light and 255 being a great deal of ambient light.

4.14.3.2 **Deprecated APIs**

The following APIs have been deprecated since the release of the i95cl.

```
public static void backlightOn()  
public static void backlightOff()  
public static void keypadLightOn()  
public static void keypadLightOff()  
public static void setStatusLight(int color)
```

4.14.4 **Tips /**

When an application uses the Lighting API and does not request that it override the native ergonomic settings, then the state of the light may appear to do strange things. An example of this is the photo sensor. Say the photo sensor is turned on and a user hits a key, which triggers the photo sensor light to turn off. Then say the light in the room is low and the photo sensor light turns back on when the user didn't press any key. It is recommended if you're going to be doing more than just flashing the lights, override the light settings to prevent the lights from changing states unexpectedly.

When overriding the lights, none of the native ergonomic battery savings for powering off the lights is in effect and great care should be taken. Leaving all of the lights on for extended periods of time can drain the phone's battery quickly.



4.15 Vibrator API

4.15.1 Overview

The Vibrator class lets a MIDlet turn the phone's vibrator on and off. It also provides the user with reoccurring effects that can be used with the vibrator. These effects allow the vibrator to be turned on and off in reoccurring patterns. This feature is useful for games or alarms.

Note that the vibrator can be turned on for a maximum of 500 ms at any given time, and it must remain off for at least 50 ms before being turned back on. This duty cycle is enforced in the API. These are important for periodic vibration since these constraints can affect how a MIDlet can use this class.

4.15.2 Class Description

The API for Vibrator is located in package `com.mot.iden.multimedia`. This class contains various multimedia classes like the Vibrator API.

```
java.lang.Object
|
+ -- com.mot.iden.multimedia.Vibrator
```

4.15.3 Method Descriptions

4.15.3.1 Vibrator Methods

4.15.3.1.1 **vibrateFor**

Turns on the vibrator for the specified amount of time.

```
public static void vibrateFor(int timeOnInMs)
```

`timeInMs` is amount of time in milliseconds to vibrate the phone.

4.15.3.1.2 **vibratePeriodically**

Turns the vibrator on and off repeatedly.

```
public static void vibratePeriodically(int timeOnInMS)
```

This method continuously turns the vibrator on and off for equal amounts of time.

`timeOnInMS` is both the amount of time the vibrator is turned on and the amount of time it's turned off.

To stop the vibrator from turning on and off, call `vibratorOff()`.

```
public static void vibratePeriodically(int timeOnInMS,
    int timeOffInMS)
```




This method continuously turns the vibrator on for one amount of time and turns it off for another amount of time. `timeOnInMs` is the amount of time to turn the vibrator on in milliseconds. `timeOffInMs` is the amount of time to turn the vibrator off in milliseconds.

4.15.3.1.3 **vibratorOff / vibratorOn**

The following methods allow a MIDlet to turn the vibrator on and off:

```
public static void vibratorOff()
public static void vibratorOn()
```

`vibratorOff()` stops the vibrator. `vibratorOn()` turns the vibrator on for `MAX_VIBRATE_TIME`, 500ms.

4.15.4 Code Examples

4.15.4.1 **Example 1**

The example below will vibrate the phone for 300 milliseconds

```
public void vibratePhone()
{
    /* this will have the phone vibrate for 300ms */
    Vibrator.vibrateFor(300);
}
```

4.15.4.2 **Example 2**

The following example allows the vibrator to vibrate periodically for 300ms using the `vibrateFor()` and `Thread.sleep()` methods:

```
public void vibratePhone()
{
    while(true){
        /* this will have the phone vibrate for 300ms */
        Vibrator.vibrateFor(300);

        /* have the phone rest for 300 ms */
        Thread.sleep(300);
    }
}
```

While this works well, the above example would tie up the execution thread. Using the `vibratePeriodically()` method is ideal for this example.

```
public void vibratePhone()
{
    /* this will have the phone vibrate for periodically 300ms */
    Vibrator.vibratePeriodically(300)
}
```



4.15.5 Tip /

Vibrating the phone drains the battery. To extend the battery life, limit the use of the vibrator.

4.15.6 Emulator Stub Classes

When a MIDlet uses the Vibrator class, it prints the action being performed on the transcript window. For example if a MIDlet turns on the vibrator for 10 milliseconds, "Vibrator Turned On" is displayed on the window.



4.16 Java Image Utility Library

4.16.1 Overview



This API is only available on these handsets.

The Java Image Utility Library consists of JPEG encoding, image resizing, and thumbnail retrieving and embedding. The library is used to provide support for Java Picture-Editor, Java Icon-decoder and other Java MIDlets through a JSR 135 extending API.

For JSR 135, please refer to the following web page for details:
<http://jcp.org/aboutJava/communityprocess/final/jsr135/>

This section focuses on introducing the JSR 135 extending API with several code examples explaining how to get started using the API.

4.16.2 Class Description

The JSR 135 extending API consists of a subset of the methods found in the following classes and interfaces:

```
java.lang.Object
|
+ - javax.microedition.media.Manager
|
+ - javax.microedition.media.Control
|
+ - javax.microedition.media.Player
```

Manager is the access point for obtaining system dependent resources such as **Players** for multimedia processing. A **Player** is an object used to control and render media that is specific to the content type of the data. **Manager** provides access to an implementation specific mechanism for constructing **Players**.

A **Control** object is used to control some media processing functions. The set of operations are usually functionally related. Thus a **Control** object provides a logical grouping of media processing functions.

Player controls the rendering of time based media data. It provides the methods to manage the **Player**'s life cycle, controls the playback progress, obtains the presentation components, controls and provides the means to synchronize with other **Players**.



4.16.3 Method Description

4.16.3.1 `javax.microedition.media.Player`

The JSR 135 extending API is involved in the following methods:

createPlayer

```
public static Player createPlayer(java.lang.String locator)  
    throws java.io.IOException, MediaException
```

Create a **Player** for an input locator.

4.16.3.1.1 Examples: input locators format

```
/* Thumbnail retrieving with file protocol */  
Player p = Manager.createPlayer("file://thumb.jpg");  
  
/* Screen capturing on internal buffer */  
Player p =  
Manager.createPlayer("capture://screen?display=internal");
```

Parameters:

locator - A locator string in URI syntax that describes the media content.

Returns: A new **Player**.

Throws:

IllegalArgumentException - Thrown if locator is null.

MediaException - Thrown if a Player cannot be created for the given locator.

java.io.IOException - Thrown if there was a problem connecting with the source pointed to by the locator.

SecurityException - Thrown if the caller does not have security permission to create the Player.

Tips:

- For file protocol, the file is located at current Java MIDlet's RSC directory.
- The default value of Screen Capturing Buffer is LCD internal buffer (176x220).
- Doesn't support "screen?display=external" (external buffer) in i860.



4.16.3.1.2 Create a Player for an InputStream.

createPlayer

```
public static Player createPlayer(java.io.InputStream stream,  
java.lang.String type)  
  
throws java.io.IOException, MediaException
```

The **type** argument specifies the content-type of the input media. If null is given, **Manager** will attempt to determine the **type**. For JSR 135 extending functions, the **type** should be "image/jpeg" or "image/x-rgb565" to indicate JPEG or RGB565 input stream respectively. However, since determining the media type is non-trivial for some media types, it may not be feasible. The **Manager** may throw a `MediaException` to indicate that.

Examples: different input streams

```
/* JPEG as inputstream */  
Player p = Manager.createPlayer(bis, "image/jpeg");  
  
/* RGB565 as inputstream */  
Player p = Manager.createPlayer(bis, "image/x-rgb565");
```

Parameters:

stream - The `InputStream` that delivers the input media.

type - The `ContentType` of the media.

Returns:

A new `Player`.

Throws:

`IllegalArgumentException` - Thrown if *stream* is null.

`MediaException` - Thrown if a `Player` cannot be created for the given stream and type.

`java.io.IOException` - Thrown if there was a problem reading data from the `InputStream`.

`SecurityException` - Thrown if the caller does not have security permission to create the `Player`.

Tips:

- For the RGB565 as input stream, the 16 bytes Motorola signature ("MOT16BIT" + height + width) should be added into the input stream as header.
- Doesn't support RGB888 as input stream in i860.



4.16.3.1.3 getSnapshot

Gets a snapshot of the displayed content. Features and format of the captured image are specified by the `imageType`.

```
public byte[] getSnapshot(java.lang.String imageType)  
    throws MediaException
```

There are five parameters used in `getSnapshot()` to support the JSR 135 extending image functions:

- *encoding*: JPEG encoding or not
- *quality*: JPEG encoding quality
- *resize*: enlarging/shrinking size
- *retrieve*: thumbnail retrieving or not
- *thumbnail* thumbnail size & embedding or not

Examples: different operation mode

```
/* Create JPEG image with resizing to 640x480 and quality = 85 */  
byte[] rawImage = vc.getSnapshot("resize=640x480&quality=85");  
  
/* Create JPEG image with resizing to 176x220 */  
byte[] rawImage = vc.getSnapshot("resize=176x220");  
  
/* Thumbnail retrieving */  
byte[] rawImage = vc.getSnapshot("retrieve=yes");  
  
/* Thumbnail creating */  
byte[] rawImage = vc.getSnapshot("resize=40x30");  
  
/* Create JPEG image embedding thumbnail */  
byte[] rawImage = vc.getSnapshot("thumbnail=30x40");
```

Parameters:

imageType - Format and resolution of the returned image. If null is given, it will be set to using the default values.

Returns:

image as a byte array in required format.

**Throws:**

`IllegalStateException` - Thrown if `initDisplayMode` has not been called.

`MediaException` - Thrown if the requested format is not supported.

`SecurityException` - Thrown if the caller does not have the security permission to take the snapshot.

Tips:

- The default parameters of the `getSnapshot()` are as follows:
 - *encoding=yes*
 - *quality=80*
 - *retrieve=no*
 - *resizing: no*
 - *thumbnail embedding: no*
- For some small size image, the JPEG encoding may be failed if set higher quality.
- Doesn't support "*encoding=no*" or "*encoding=rgb565*" in i860.
- The range of *quality* is suggested to be set between 55 and 85.
- The maximum of *resize* is: 640x480.
- The minimum of *resize* is: 30x30.
- For screen capturing, the *retrieve* is always set to be *no*.
- If using RGB565 as input stream, the *retrieve* is always set to be *no*.

The parameters used in `createPlayer()` and `getSnapShot()` are summarized in the Table below:



createPlayer()		getSnapshot()		
Protocol	Type	imageType	Operations	Return Data
File "file://wxyz.jpg"		"encoding=jpeg"	Thumbnail Retrieving	JPEG format
Capture "capture://screen?display=internal"		"encoding=jpeg&resize=640x480&quality=85&thumbnail=30x40"	JPEG Encoding, Image Resizing, Thumbnail Embedding,	JPEG format
InputStream (RGB or JPEG)	"image/jpeg"	(1) "resize=220x176&thumbnail=40x30&quality=79&retrieve=1&encoding=jpeg" (2) "retrieve=yes"	JPEG Encoding, Thumbnail Retrieving, Image Resizing, Thumbnail Embedding	Image/Thumbnail with JPEG format
	"image/x-rgb565"	"encoding=jpeg&quality=78&thumbnail=30x40&retrieve=0&resize=100x100"	JPEG Encoding, Image Resizing, Thumbnail Embedding,	JPEG format

4.16.4 Code Example

1. Image Resizing (JPEG as InputStream)

```

...
Player p = null;
VideoControl vc = null;
byte jbuffer[] = new byte[50000];
byte[] rawImage;
...
try {
    /* Using JPEG file as input data */
    InputStream in =
        this.getClass().getResourceAsStream("Pic2.jpg");
    length = in.available();
    int readLength = 0;
    int writeLength = 0;

    while ( writeLength != length )
    {
        readLength = in.read(jbuffer, writeLength, length - readLength);
        writeLength += readLength;
    }
    in.close();
}

```




```

byte[] barry = new byte[length];
System.arraycopy(jbuffer, 0, barry, 0, length);
ByteArrayInputStream bis = new ByteArrayInputStream(barry);

/* JPEG as InputStream */
p = Manager.createPlayer(bis, "image/jpeg");

if ( p.getState() == Player.UNREALIZED ) {
    p.realize();
}
if ( vc == null ) {
    vc = (VideoControl)p.getControl("VideoControl");
}
if ( init != true ) {
    // only want this to happen once.
    vc.initDisplayMode(VideoControl.USE_GUI_PRIMITIVE, null);
    init = true;
}

if ( p.getState() != Player.STARTED ) { p.start(); }

/* Resizing to VGA size */
rawImage = vc.getSnapshot("resize=640x480&quality=79
&encoding=jpeg");
...
} catch (MediaException pe) {
    System.out.println("Get MediaException here!");
}
...

```

2. Image Resizing (RGB as inputStream)

```

...
Player p = null;
VideoControl vc = null;
byte jbuffer[] = new byte[50000];
byte[] rawImage;
...
try {
    /* Using RGB565 as input data */
    InputStream in =
        this.getClass().getResourceAsStream("RGB565.txt");
    length = in.available();
    int readLength = 0;
    int writeLength = 0;

    while ( writeLength != length )
    {
        readLength = in.read(jbuffer, writeLength, length - readLength);
        writeLength += readLength;
    }
    in.close();

    byte[] barry = new byte[length];

```



```

System.arraycopy(jbuffer, 0, barry, 0, length);
ByteArrayInputStream bis = new ByteArrayInputStream(barry);

/* RGB as InputStream */
p = Manager.createPlayer(bis, "image/x-rgb565");

if ( p.getState() == Player.UNREALIZED ) {
    p.realize();
}
if ( vc == null ) {
    vc = (VideoControl)p.getControl("VideoControl");
}
if ( init != true ) {
    // only want this to happen once.
    vc.initDisplayMode(VideoControl.USE_GUI_PRIMITIVE, null);
    init = true;
}

if ( p.getState() != Player.STARTED ) {p.start(); }

/* Resizing to wallpaper size: quality = 89 */
rawImage = vc.getSnapshot("resize=176x220&
encoding=jpeg&quality=89");
...
} catch (MediaException pe) {
    System.out.println("Get MediaException here!");
}
...

```

3. Thumbnail Retrieving

```

...
Player p = null;
VideoControl vc = null;
byte[] rawImage;
...
try {
    /* files protocol: thumb.jpg is stored at Java RSC dir */
    p = Manager.createPlayer("file://thumb.jpg");

    if ( p.getState() == Player.UNREALIZED ) {
        p.realize();
    }
    if ( vc == null ) {
        vc = (VideoControl)p.getControl("VideoControl");
    }
    if ( init != true ) {
        //only want this to happen once.
        vc.initDisplayMode(VideoControl.USE_GUI_PRIMITIVE, null);
        init = true;
    }
    if ( p.getState() != Player.STARTED ) { p.start(); }

    /* Getting thumbnail in JPEG format */

```



```

rawImage = vc.getSnapshot("encoding=jpeg");

p.close();
p = null;
vc = null;
...
} catch (MediaException pe) {
    System.out.println("Get MediaException here!");
}
...

```

4. Screen Capturing (Full Code)

```

// Java Testing for Screen Capture

// Key6: show girlface.jpg (just JPEG decoding)
// Key7: Create Player
// Key8: Display the content of the Internal Screen Buffer

// Testing Process: Launch the MIDlet. First, press Key6 then
//                  Key7 and then Key8

// Input data: girlface.jpeg -- phone wallpaper size (176x220)

import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;
import javax.microedition.midlet.*;
import javax.microedition.media.*;
import javax.microedition.media.control.*;

public class Resizing extends MIDlet implements CommandListener {

    private static final int JPEG = 0;    // JPEG mode: display picture
    private static final int SCREEN = 1;  // SCREEN mode: screen capturing

    /**
     * The screen for this application
     */
    private MainCanvas myCanvas;

    Player p = null;

    /**
     * A reference to the Display
     */
    private Display myDisplay;

    Control c;

    // Testing time
    long t1, t2;

    // Control currentImage and oldImage

```



```

int yPos = 0;
int xPos = 0;

private Image currentImage = null;
private Image oldImage = null;

private int width;
private int height;
private int imageIndex;
private int mode = JPEG;
private boolean on = false;
private Font font = null;
private String[] jpegImages = {"/girlface.jpg"};

int counter;
/* parameters for getSnapshot() */
String[] snapshotURIs =
{
    "encoding=jpeg&resize=220x280&quality=75",
    "encoding=jpeg&resize=40x30&quality=76",
    "encoding=jpeg&resize=640x480&thumbnail=30x40&quality=77",
    "encoding=jpeg&resize=160x230&thumbnail=30x40&quality=78",
    "encoding=jpeg&resize=200x200&quality=79",
    "encoding=jpeg&quality=79",
    "encoding=jpeg&quality=79&thumbnail=30x40",
    "retrieve=yes",
};

byte[] rawImage;

/**
 * Create the Resizing
 */
public Resizing() {
    myCanvas = new MainCanvas();
    myDisplay = Display.getDisplay(this);
}

/**
 * Signals the MIDlet to start providing service and enter the
 * Active state.
 */
public void startApp() throws MIDletStateChangeException {
    myDisplay.setCurrent(myCanvas);
}

/**
 * Signals the MIDlet to stop providing service and enter the
 * Paused state. In the Paused state the MIDlet must stop
 * providing service, and might release all resources and become
 * quiescent.
 */
public void pauseApp() {

```



```

}

/**
 * Signals the MIDlet to terminate and enter the Destroyed state
 * Midlets should perform any operations required before being
 * terminated, such as releasing resources or saving preferences
 * or state.
 */
public void destroyApp(boolean unconditional) {
    if (p != null)
        p.close();
}

public void commandAction(Command c, Displayable s) {
    myDisplay.setCurrent(myCanvas);
    myCanvas.repaint();
}

/**
 * The frame class for this application
 */
class MainCanvas extends Canvas {

    Image myImage;

    int myColor = 0x00ff0000;
    VideoControl vc = null;
    boolean init = false;

    public MainCanvas() {
        width = getWidth();
        height = getHeight();
    }

    /**
     * Renders the Canvas
     * @param g graphics object to render the Canvas with
     */
    public void paint(Graphics g) {
        // clear the screen
        g.setColor(0xFFFFFFFF);           // white
        g.fillRect(0, 0, width, height);

        switch (mode) {
            case JPEG:
                g.setColor(0xff0000);
                g.drawString("JPEG 24-bit dithered", 0,
                    yPos, Graphics.LEFT | Graphics.TOP);
                break;

            case SCREEN:
                g.setColor(0x0000ff);
                g.drawString("SCREEN DISPLAY", 0,

```



```

        yPos, Graphics.LEFT | Graphics.TOP);
    break;
}

font = g.getFont();
int y = font.getHeight() + yPos - 1;
g.setColor(0x000000);           // black
g.drawLine(0, y, width, y);

if ( mode == JPEG )
{
    if (currentImage != null) {
        g.drawImage(currentImage, xPos, font.getHeight() + yPos,
            Graphics.TOP | Graphics.LEFT);
        //System.out.println("Draw current Image");
    }
}

if ( mode == SCREEN )
{
    if ( oldImage != null ) {
        g.drawImage(oldImage, xPos, font.getHeight() + yPos,
            Graphics.BOTTOM | Graphics.RIGHT);
        //System.out.println("Draw Internal Screen Buffer");
    }
}
System.gc();
}

public void keyPressed (int key) {

    myColor = 0x00ff0000;

    if (key == Canvas.KEY_NUM6)
    {
        mode = JPEG;

        try {
            currentImage = Image.createImage(jpegImages[imageIndex]);
        }
        catch (Exception e) {
            System.err.println(e);
        }
    }
    else if (key == Canvas.KEY_NUM7)
    {
        try {
            reset();
            p = Manager.createPlayer("capture://screen?display=internal");
        } catch (Exception e)
        {}
    }
    else if ( key == Canvas.KEY_NUM8 )

```



```

{
    mode = SCREEN;

    try {
        if (p.getState() == Player.UNREALIZED) { p.realize(); }

        if (vc == null) {
            vc = (VideoControl)p.getControl("VideoControl");
        }

        if (vc != null) {
            if (init != true) {
                // only want this to happen once.
                vc.initDisplayMode(VideoControl.USE_GUI_PRIMITIVE, null);
                init = true;
            }

            if (p.getState() != Player.STARTED) { p.start(); }

            rawImage = vc.getSnapshot(snapshotURIs[counter]);

            oldImage = Image.createImage(rawImage, 0, rawImage.length);
            xPos = xPos + 60;
            yPos = yPos + 40;
        }
    } catch (Exception e)
    {}
}

else if ( getGameAction(key) == Canvas.LEFT )
{
    xPos = xPos - 10;
}
else if ( getGameAction(key) == Canvas.RIGHT )
{
    xPos = xPos + 10;
}
else if ( getGameAction(key) == Canvas.UP )
{
    yPos = yPos - 10;
}
else if ( getGameAction(key) == Canvas.DOWN )
{
    yPos = yPos + 10;
}
else if ( key == -21 ) /* Soft KEY: change parameter */
{
    counter = (counter + 1 ) % snapshotURIs.length;
    System.out.println(snapshotURIs[counter]);
}
else
{
    System.out.println("other keys");
}

```



```
repaint();  
}  
  
private void reset()  
{  
    if (p != null) {p.close(); p = null;}  
    vc = null;  
    init = false;  
}  
} //end canvas  
} //end midlet
```




5

Telephony

5.1 Overview

The following sections describe the telephony management scheme for the iDEN Multi-Communication Device. This chapter will discuss the following:

- Interconnect, Private (Dispatch), and Selective Dynamic Group Call Initiation
- Call Receiving
- Recent Calls
- Java PIM Package
- Phonebook

5.2 Interconnect/Phone Call Initiation API

5.2.1 Overview

The Call Initiation API provides the ability to request interconnect, dispatch, and SDG calls. The API supports international and domestic phone numbers, “pause” dialing, and “wait” dialing.

The API does not actually make the call. Instead, it is designed to simply initiate a call request, wherefore then the end user must grant the request by pressing the Send key (also known as the Fire key in the Canvas Class). Since an application will be immediately suspended after calling the API, the employment of this interface must be from a separate thread other than the main thread. Upon successful call termination, the application will be resumed if the auto revert feature is enabled.

5.2.2 Class Description

The API for Call Initiation is located in package `com.motorola.iden.call`. The `GenericCall` class is the only class within the package needed to initiate calls and contains one static method to initiate the service calls.



```
java.lang.Object
|
+ -- com.motorola.iden.call.GenericCall
```

5.2.3 Method Description

5.2.3.1 GenericCall Method

5.2.3.1.1 makeCall(String number)

Initiates a call request, which the user must grant by pressing the Send key.

```
public static int makeCall(String number) throws Exception
```

All MIDlet threads are kept running after calling this function and establishing the call. Packet data activity is stopped while in the phone call. Currently, you should start a call request only while the phone is idle (not active in any type of service call). Otherwise, this method throws an exception.

This method's argument specifies the number to dial. Its format is as follows:

number ::= [<Prefix Tag>] <Id>

The optional prefix tag included within the string is case insensitive and the Id (which has a maximum of 64 characters) determines the number dialed.

makeCall () Argument Format

Prefix Tag	Id Value	Call Behavior
"phon" or "PHON" none*	Domestic Interconnect Call	
	(xxx)xxx-xxxx	-Dials xxxxxxxxxx
	XXXXXXXXXX	-Dials xxxxxxxxxx
	(xxx)xxx-xxxxPyyy or (xxx)xxx-xxxx,yyy	-Dials xxxxxxxxxx, connects, and generates yyy DTMF's
	xxxxxxxxxxPyyPzzPPz or xxxxxxxxxx,yy,zz,,z	-Dials xxxxxxxxxx, connects, generates yy DTMF's, pauses for 3 seconds, generates zz DTMF's, pauses for 6 seconds, and generates z DTMF.
	xxxxxxxxxxWyy	-Dials xxxxxxxxxx and connects. Once the user presses the Send (or Fire) key, yy DTMF's are generated.
	International Interconnect Call	
	+(xxx)xxx-xxxx	-Dials xxxxxxxxxx with international "type of number"
	+xxxxxxxxxxPyy	-Dials xxxxxxxxxx with international "type of number", connects, and generates yy DTMF's.
"Prvt" or "PRVT"	Dispatch Call	
	xxxxx	Make a dispatch call to xxxxx.

To place a domestic call, the string argument may be "(XXX) XXX-XXXX", "XXXXXXXXXX", "phonXXXXXXXXXX", or "PHONXXXXXXXXXX". To make an international phone call, a '+' must be placed between the tag and the number to be dialed, as in "+XXXXXXXXXX" or



“phon+XXXXXXXXXX”. To use pause dialing, insert a “P” (case sensitive) or ‘,’ (Comma) as a pause digit. The first instance of the pause digit separates the phone number to be dialed from the DTMF tones that will be generated after the call is connected. Any subsequent “pause digit” inserts a 3-second break into the DTMF string at the specified location (refer to Table 1). To use wait dialing, insert a “W” (case sensitive) as the wait digit. The first instance of the wait digit separates the phone number to be dialed from the DTMF. The DTMFs are not generated until the user presses the Send (or Fire) key after each wait digit. The length of the ID portion of the argument used in `makeCall()` (that is, the length of the string excluding the tag) should not exceed 64 characters. Any number of blank spaces in the number is ignored. If the number is null or contains any invalid characters, this method throws an `IllegalArgumentException`.

Once in the dialing screen, the end user must grant the request by pressing the Send key (also known as the Fire key in the Canvas Class). If the auto-revert feature is enabled and the end user does not grant the request (i.e. press the Send key) within 3 seconds, the application is resumed.

The `makeCall()` method can successfully process only one request at a time. For example, if a thread makes a call request and is waiting for the response, any other thread will receive an exception when attempting to initiate a call. The exception will be thrown for each call request until the system is no longer busy (i.e. the first thread receives a response).

Return values of the `makeCall()` method are:

- `GenericCall.CALL_RESPONSE_OK` if phone call request is placed successfully (user must grant the request)
- `GenericCall.CALL_RESPONSE_FAILURE` if request fails because the unit is not idle (e.g. in an active service call such as a private call)
- `GenericCall.CALL_RESPONSE_ERROR` if an error occurred during request because the keypad is disabled and the application was attempting to dial a number not in the phonebook
- `GenericCall.CALL_UNKNOWN_ERROR` if an unknown error took place while requesting

5.2.3.1.1 `makeCall(int type, String alias, String[] idList)`

Starts a Selective Dynamic Group Call.

```
public static int makeCall(int type, String alias, String[] idList)
throws Exception
```

When starting a SDGC call, the app should register the SDGC service first. An app starting a SDGC call without registering first will receive `IllegalAccessError`.

The participant list should contain at least 2 unique private ID; otherwise, the call will return `CALL_RESPONSE_ERROR`.



5.2.4 Code Examples

```
void makePhoneCall() {
    try {
        String number = "9555555555";
        int x = GenericCall.makeCall(number);
        if ( X == GenericCall.CALL_RESPONSE_OK) {
            // request to initiate has been successfully made
            // Note: does not mean phone call is finished

        }
        else {
            // Error occurred while making request
        }
    } catch (IllegalArgumentException e) {
    }
    catch (Exception e) {
    }
}
```

5.2.5 Compiling & Testing Interconnect Capable MIDlets

The stubbed `GenericCall` class is a non-functional class. The class is provided to build and run within any emulator. The class will make an attempt to display its method's behavior through `System.out` print statements when a method is called.

If the device is idle, `makeCall(9555555555)` displays the following:

```
Requesting to call to 9555555555
Waiting for request response
    The MIDlet pauses here. After the user presses the Send key,
    the phone call is started and the MIDlet remains paused.
CALL_RESPONSE_OK
```

If the keypad is disabled and 955555555 is not in the phonebook, `makeCall(955555555)` displays the following:

```
Requesting to call to 955555555
CALL_RESPONSE_ERROR
```

If the device is busy in another call, `makeCall(955555555)` displays the following:

```
Requesting to call to 955555555
CALL_RESPONSE_FAILURE
```



5.3 Call Receiving API

5.3.1 Overview



This API is only available on these handsets.

The Call Receiving API lets you answer incoming phone calls. The user can accept, reject, or end an incoming or connected call.

5.3.2 Class Descriptions

The API for the Call Receiving feature is located in package `com.motorola.iden.call`.

This is the class hierarchy for the `CallHandler` class:

```
java.lang.Object
|
+ - com.motorola.iden.call.CallHandler
|
+ - com.motorola.iden.call.Call
      |
      + - com.motorola.iden.call.InterconnectCall
      |
      + - com.motorola.iden.call.CallListener
      |
      + - com.motorola.iden.call.DispatchCall
      |
      + - com.motorola.iden.call.SdgcCall
```



`CallHandler` is a class that provides functionality to register for calls. The `Call` interface provides basic call functionality. `InterconnectCall` is an interface that extends the `Call` interface, and provides interconnect call functionality. The `CallListener` interface provides a listening mechanism for the different interfaces, such as `InterconnectCall`, if it is a registered type of call.



Similar to the InterconnectCall interface, the DispatchCall and SdgcCall interfaces extend the Call interface to provide dispatch and SDG call functionality, respectively.

5.3.3 Method Descriptions

5.3.3.1 CallHandler Methods

5.3.3.1.1 getInstance

Returns an instance of a Call.

```
public static com.motorola.iden.call.Call getInstance(int type,
com.motorola.iden.call.CallListener l)
```

This method returns an instance of a Call. If the method is called again, it returns the same instance of Call.

`type` specifies the type of call and `l` is the call listener, usually the MIDlet. The MIDlet must call this method once for each type of call it wants to listen to.

This method needs to be called from a thread. Calling it from the `startApp()` method of a class is the best way to implement the Call interface. The method needs to be called as soon as the application launches.

The following example illustrates how the Call instance can be retrieved in a MIDlet's `startApp()` method:

```
boolean isCallThreadRunning = false;
public void startApp(){
    if(!isCallThreadRunning){
        new Thread (new CallThread()).start(); // starting a new
                                                // thread
    }
}

//Thread implementation
public class CallThread extends Thread
{
    public CallThread(){
        isCallThreadRunning = true;
        try {
            call = (InterconnectCall)
                CallHandler.getInstance(CallHandler.
                    INTERCONNECT_CALL, this);
        } catch(IllegalArgumentException e) {
        }
    }
}
```



5.3.3.1.2 startCallListener

Starts your MIDlet's `callActionListener()` method when the call's state changes.

```
public void startCallListener()
```

After you call this method, your MIDlet's `callActionListener()` method is called whenever the call's state changes. Your MIDlet class must implement the `CallListener` interface as a thread that will listen for call state changes while it is running. If the phone receives a call type that hasn't been registered for via `getInstance()`, the listener's callback methods are not be called.

In the run method of a `CallThread`, this method can be called as shown in the following example:

```
public void run(){
    try {
        handle.startCallListener();
    } catch(Exception e) {
    }
}
```

5.3.3.2 CallListener Methods

5.3.3.2.1 callActionListener

Lets you know the state of the phone call.

```
public void callActionListener(int type, int state)
```

Implement this method if you want to be notified when the call changes state. When `type` is `INTERCONNECT_CALL` `state` is one of the following constants:

```
PHRX_INCOMING_STATE
PHRX_CALL_CONNECTED_STATE
PHRX_CALL_STOPPED_STATE
PHRX_CALL_REJECTED_STATE
PHRX_CALL_CANCELLED_STATE
PHRX_CALL_ENDED_STATE.
```



When type is DISPATCH_CALL state is one of the following constants:

```

PRVT_WAITING_RESPONSE
PRVT_INCOMING_STATE
PRVT_CALL_CANCELED_STATE
PRVT_CALL_STOPPED_STATE
PRVT_INHIBIT_RCVD_STATE
PRVT_PERMIT_RCVD_STATE
PRVT_INHIBIT_XMT_STATE
PRVT_PERMIT_XMT_STATE

```

i830e
only



If you want this method to be called, you must first call `startCallListener()`. State information is received only for registered call types.

```

public void callActionListener(int type, int state)
{
    // type will be the type of a call
    // state will be the kind of state the call is currently
    if ( (type == CallHandler.INTERCONNECT_CALL) &&
        (state == CallHandler.PHRX_CALL_CONNECTED_STATE) )
    {
        String s = call.getCallNumber();
        ...
    }
    else if((type == CallHandler.DISPATCH_CALL) &&
        (state == CallHandler.PRVT_INHIBIT_RCVD_STATE) )
    {
        //Receiver De-Keys the PTT button or stops talking.
    }
}

```



i830e
only



5.3.2.3 Call Methods

5.3.2.3.1 `getCallNumber`

Returns the phone number of the current call.

```
public String getCallNumber()
```

If the string returned by this method is equal to `Call.PHRX_NO_CALLER_ID_1` or `Call.PHRX_NO_CALLER_ID_2`, then the received call does not have caller ID information.

5.3.2.3.2 `getCallState`

Returns the current state of a call.

```
public int getCallState()
```

This method should be used to stop/start network activities, or can be useful for any application that supports call receiving functionality. For example, a racing game can also handle a call, but when a call is in progress, it should not display game functionality, but should display call related information.

The method can return any of the following constants.

```
PHRX_INCOMING_STATE
PHRX_CALL_CONNECTED_STATE
PHRX_CALL_STOPPED_STATE
PHRX_CALL_REJECTED_STATE
PHRX_CALL_CANCELLED_STATE
PHRX_CALL_ENDED_STATE
```



i830e
only

```
PRVT_WAITING_RESPONSE
PRVT_INCOMING_STATE
PRVT_CALL_CANCELED_STATE
PRVT_CALL_STOPPED_STATE
PRVT_INHIBIT_RCVD_STATE
PRVT_PERMIT_RCVD_STATE
PRVT_INHIBIT_XMT_STATE
PRVT_PERMIT_XMT_STATE
```

5.3.2.3.3 `getLineNumber`

The method returns the line number of an interconnect call.

```
public int getLineNumber()
```



5.3.2.4 InterconnectCall Methods

5.3.2.4.1 **interconnectAction**

Lets you accept, reject, or end an incoming call.

```
public void interconnectAction(int action_type, String phoneNumber)
    throws NumberFormatException, IllegalArgumentException
```

`action_type` must be one of the following:

`InterconnectCall.REJECT_CALL` rejects an incoming phone call.

`InterconnectCall.END_CALL` ends an incoming phone call.

`InterconnectCall.HOLD_CALL` holds a connected active call.

`InterconnectCall.RESUME_CALL` resumes a call that's been put on hold.

`phoneNumber` should be the number for the incoming call, such as returned by `getCallNumber()`.

5.3.2.4.2 **switchPhoneCall**

Switches between two active calls.

```
public void switchPhoneCall() throws IllegalStateException
```

If there are two active calls, your MIDlet should display a method for switching between them. If two calls are not active, this method throws `IllegalStateException`.

5.3.2.4.3 **enableMute**

Enables and disables muting in an active phone call.

```
public void enableMute(boolean state)
```

To enable muting, `state` must be true. To disable muting, `state` must be false.

The application should keep track of whether muting is enabled and should display a way for the user to turn it on or off.

5.3.2.4.4 **enableSpkrPhone**

Enables and disables the speaker phone.

```
public void enableSpkrPhone(boolean state)
```

To enable the speaker phone, `state` must be true. To disable the speaker phone, `state` must be false.

The application should keep track of whether the speaker phone is enabled and should display a way for the user to turn it on or off.



5.3.2.4.5 startDTMF/stopDTMF

Starts and stops sending DTMF tones.

```
public void startDTMF (char digit)
    throws IllegalStateException
```

```
public void stopDTMF () throws IllegalStateException
```

`startDTMF()` starts sending the DTMF tone specified by `digit`. `stopDTMF()` stops the tone.

If there is not a connected call, these methods throw an `IllegalStateException`.

5.3.2.4.6 playRinger

Plays the specified ringer.

```
public void playRinger (int index)
    throws IndexOutOfBoundsException
```

The ringer stops as soon as the interconnect call gets answered, rejected, or ended.

`index` is the index for a ringer stored on the phone. To play the default ringer, use `-1`. If there's no ringer for `index`, this method throws an `IndexOutOfBoundsException`.

Call this method when the phone receives an incoming call.

This is an example of using `playRinger()`:

```
public void callActionListener(int type, int state)
{
    // type will be the type of a call
    // state will be the kind of state the call is currently
    if ( (type == CallHandler.INTERCONNECT_CALL) &&
        (state==CallHandler.PHRX_CALL_CONNECTED_STATE) )
    {
        call.playRinger(-1);
        ...
    }
}
```

5.3.2.5 Dispatch and SDG Call methods



i830e
only

The APIs listed below can be used while during an active private or SDG call. By pressing the Hi/Lo key, the speaker will be turned on/off. The Up-Down Volume keys will work as it is while actively in either of these types of calls.



5.3.2.5.1 DispatchCall.dispatchAction

This method works differently depending upon if the MIDlet is signed or not. If the MIDlet is not signed, this method will be helpful only to end the active private call. In this case, the application will behave similar to phone ergonomics. User can press/resume PTT key in order to start/stop talking.

If the MIDlet is signed, then there are two action_types that can be used.

DispatchCall.PTT_PRESSED can be used in place of PTT key press/ hold during an active private call (user starts talking). DispatchCall.PTT_RELEASED can be used in place of PTT key released during an active private call (user stops talking).

```
public void dispatchAction(int action_type)
    throws NumberFormatException, IllegalArgumentException
```

action_type must be one of the following:

- DispatchCall.PTT_PRESS Start talking during active private call.
- DispatchCall.PTT_RELEASE Stop talking during active private call.
- DispatchCall.END_CALL ends a private call.

5.3.2.5.2 SdgcCall.sdgcAction

This method can be used with different action_type values to implement functionality for call related keys such as the PTT key and END key.

```
public void sdgcAction(int action_type)
    throws IllegalArgumentException
```

action_type must be one of the following:

- SdgcCall.PTT_PRESS Start talking during active SDG call.
- SdgcCall.PTT_RELEASE Stop talking during active SDG call.
- SdgcCall.END_CALL ends an SDG call.

5.3.2.5.3 DispatchCall.getCallAlias

The method returns the Alias name of the caller from the phonebook.

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

5.3.2.5.4 SdgcCall.getCallAlias

The method returns the SDG call alias of the current active call.

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

5.3.2.5.5 SdgcCall.getSdgcOriginatorID

The method returns the ID of the current SDG call's originator.



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

5.3.2.5.6 SdgcCall.getSdgcPartList

The method returns the participants of the current SDG call.

```
public java.lang.String[] getSdgcPartList()
```

5.3.2.5.7 SdgcCall.getSdgcPartStat

The method returns the status of the participants of the current SDG call.

```
public byte[] getSdgcPartStat()
```

5.3.2.5.8 SdgcCall.getSdgcTotalPartNumber

The method returns the received participant count of the current SDG call.

```
public int getSdgcTotalPartNumber()
```

5.3.2.5.8 SdgcCall.getSdgcSystemError

The method returns the last system error message of the current SDG call.

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

5.3.3 Code Examples

The following is a code example using the Call Receiving feature.

```
public class ExCall extends MIDlet implements CommandListener
{
    ExCall myMidlet;
    String pnumber = "9999999999";
    Form tScreen;
    Display myDisplay;
    ExternalDisplay ed;
    extCanvas exCan = new extCanvas(); // External Display Canvas
    CallHandler handle = new CallHandler();
    InterconnectCall call;

    public boolean isThreadRun = false;

    public ExCall()
    {
        myMidlet = this;
        myDisplay = Display.getDisplay(this);
    }
    public void startApp()
    {
        // good practice if thread is started just after following
        ed = ExternalDisplay.getDisplay(this);
        ed.setCurrent(exCan);

        if(!isThreadRun)
            new Thread(new CallThread()).start();
    }
}
```



```
}
public void pauseApp(){
}

}
public void destroyApp(boolean unconditional) {
}

public void commandAction(Command c, Displayable s)
{
    if(c == acceptCommand){
        try {
            call.interconnectAction(2, call.getCallNumber());
        }catch (Exception e){
        }
    }
    else if(c == rejectCommand) {
        try {
            call.interconnectAction(1, call.getCallNumber());
        } catch(Exception e){
        }
    }
}

}

public class CallThread extends Thread implements CallListener
{
    public CallThread()
    {
        isThreadRun = true;
        try {
            call = (InterconnectCall)CallHandler.getInstance(
                CallHandler.INTERCONNECT_CALL,this);
        }catch(Exception e) {
        }
    }

    public void run()
    {
        try {
            handle.startCallListener();
        } catch(Exception e) {
        }
    }

    public void callActionListener(int type, int state)
    {
        if(state == -1){
        }
    }
}
```



```

if (state == CallHandler.PHRX_INCOMING_STATE) {
    String pnumber = call.getCallNumber();

    myTick.setString(pnumber);
    call.playRinger(-1);
    call.getLineNumber();

}else if (state == CallHandler.PHRX_CALL_CONNECTED_STATE) {

}else if (state == CallHandler.PHRX_CALL_STOPPED_STATE) {
    System.out.println("in stopped state..");
}
}
}

```

The following is a code example using the SDG Call Receiving and Initiation feature.

```

import com.motorola.iden.call.*;
import java.io.PrintStream;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import com.motorola.iden.lcdui.ExternalDisplay;
import com.motorola.iden.lcdui.ExternalDisplayCanvas;
import com.mot.iden.multimedia.Lighting;

public class Sdgc extends MIDlet
    implements CommandListener
{
    class extCanvas extends ExternalDisplayCanvas
    {

        protected void paint(Graphics g)
        {
            String displayStr;
            g.setColor(0xffffffff);
            g.fillRect(0, 0, getWidth(), getHeight());
            g.setColor(0);
            Graphics _tmp = g;
            Graphics _tmp1 = g;
            if(callAlias != null) {
                displayStr = callAlias + " active";
            }else {
                displayStr = "";
            }
            g.drawString(displayStr, 0, 0, 0x10 | 0x4);
        }

        protected void showNotify()
        {
            try
            {
                (new Thread(new LightThread())).start();
                repaint();
            }
        }
    }
}

```



```
        System.out.println("external show notify");
    }
    catch(Exception exception) { }
}

public extCanvas()
{
}

}

class LightThread extends Thread
{

    public void run()
    {
        Lighting.javaOverRideLighting(true);
        do
        {
            try
            {
                Lighting.setLighting(4, 2);
                Thread.currentThread();
                Thread.sleep(250L);
                Lighting.setLighting(4, 7);
                Thread.currentThread();
                Thread.sleep(250L);
                Lighting.setLighting(4, 4);
                Thread.currentThread();
                Thread.sleep(250L);
            }
            catch(Exception exception) { }
        } while(true);
    }

    public LightThread()
    {
    }

}

class CallThread extends Thread
    implements CallListener
{

    public void run()
    {
        try
        {
            handle.startCallListener();
        }
        catch(Exception exception) { }
    }

    public void callActionListener(int i, int j)
    {
    }
}
```




```

        if(j == CallHandler.SDGC_END_CALL_EVENT) {
            siCallingNumber.setText("sdgc ended, the error is" +
sdgc.getSdgcSystemError());
            //ed.requestDisplay();
            //myDisplay.setCurrent(mainForm);
            callAlias = null;
        }
        if(j == CallHandler.SDGC_CALL_START_EVENT ) {
            siCallingNumber.setText("sdgc originate from" +
sdgc.getSdgcOriginatorID() + " now wait");
            myDisplay.setCurrent(fmAnswer);
            callAlias = sdgc.getSdgcCallAlias();
        }
        if(j == CallHandler.SDGC_LISTEN_PERMIT_EVENT) {
            siCallingNumber.setText("sdgc listen now");
            callAlias = sdgc.getSdgcCallAlias();
        }
        if(j == CallHandler.SDGC_TALK_PERMIT_EVENT) {
            siCallingNumber.setText("sdgc talk");
            callAlias = sdgc.getSdgcCallAlias();
        }
        if(j == CallHandler.SDGC_TALK_INHIBIT_EVENT ) {
            siCallingNumber.setText("talk over,come back to wait");
        }
        if(j == CallHandler.SDGC_LISTEN_INHIBIT_EVENT ) {
            siCallingNumber.setText("talk over,come back to wait");
        }
        if(j == CallHandler.SDGC_SYSTEM_TERMINATED_EVENT ) {
            siCallingNumber.setText("sdgc terminate");
            callAlias = null;
            myDisplay.setCurrent(mainForm);
            isEnd = true;
        }
    }

    public CallThread()
    {
        isThreadRun = true;
        try
        {
            //call = (InterconnectCall)CallHandler.getInstance(1,
this);

            sdgc = (SdgcCall)CallHandler.getInstance(3, this);
        }
        catch(Exception exception)
        {
            isThreadRun = false;
        }
    }
}
//private static boolean debug = true;
ExternalDisplay ed;

```



```

extCanvas exCan;
Form fmAnswer;
StringItem siCallingNumber;
CallHandler handle;
Command pttpressCommand;
Command pttreleaseCommand;
Command endCommand;
SdgcCall sdgc;
public boolean isThreadRun;
private Display myDisplay;
private Form mainForm;
TextField pid1;
TextField pid2;
TextField pid3;
TextField alias;
private TextBox exceptionTextBox;
private Command callCommand;
private Command exitCommand;
private String[] prvIDList;
private String callAlias;
private boolean isEnd;

public Sdgc()
{
    isEnd = true;
    isThreadRun = false;
    handle = new CallHandler();
    exCan = new extCanvas();
    myDisplay = Display.getDisplay(this);
    pttpressCommand = new Command("PRESS", 1, 1);
    pttreleaseCommand = new Command("RELEASE", 1, 2);
    endCommand = new Command("End", 1, 2);
    fmAnswer = new Form("Call Me");
    siCallingNumber = new StringItem("SDGC", "");
    fmAnswer.addCommand(pttpressCommand);
fmAnswer.addCommand(endCommand);
    fmAnswer.append(siCallingNumber);
    fmAnswer.setCommandListener(this);
    mainForm = new Form("SDGC");
    callCommand = null;
    exitCommand = null;
    callAlias = null;
    pid1 = new TextField("privateID1", "50002", 15, TextField.ANY);
    pid2 = new TextField("privateID2", "50003", 15, TextField.ANY);
    pid3 = new TextField("privateID3", "50004", 15, TextField.ANY);
    alias = new TextField("sdgc alias", "alias", 15, TextField.ANY);
    exceptionTextBox = new TextBox("Exception Caught!", "", 100, 0);
    mainForm.append(pid1);
    mainForm.append(pid2);
    mainForm.append(pid3);
    mainForm.append(alias);

    callCommand = new Command("CALL", 1, 2);

```



```

        exitCommand = new Command("EXIT", 1, 2);
        mainForm.addCommand(callCommand);
        mainForm.addCommand(exitCommand);
        mainForm.setCommandListener(this);
        myDisplay.setCurrent(mainForm);
    }

    public void startApp()
        throws MIDletStateChangeException
    {
        ed = ExternalDisplay.getDisplay(this);
        ed.setCurrent(exCan);
        if(!isThreadRun)
            (new Thread(new CallThread())).start();
    }

    public void pauseApp()
    {
        if(ed.getFlipState()) {
            ed.requestDisplay();
        }
    }

    public void destroyApp(boolean flag)
    {
    }

    public void commandAction(Command command, Displayable displayable)
    {
        if(command == callCommand) {
            prvIDList = new String[3];
            try {
                prvIDList[0] = pid1.getString();
                prvIDList[1] = pid2.getString();
                prvIDList[2] = pid3.getString();
                callAlias = alias.getString();
                int i = GenericCall.makeCall(3, callAlias, prvIDList);
                if(i == 0) {

System.out.println("GenericCall.CALL_RESPONSE_OK");
                    myDisplay.setCurrent(fmAnswer);
                }else {
                    if(i == 1)

System.out.println("GenericCall.CALL_RESPONSE_FAILURE");
                    else
                        if(i == 2)

System.out.println("GenericCall.CALL_RESPONSE_ERROR");
                    else
                        if(i == -1)

```



```

System.out.println("GenericCall.CALL_UNKNOWN_ERROR");
    }
}
catch(Exception ex) {
    myDisplay.setCurrent(exceptionTextBox);
    exceptionTextBox.setString(ex.toString());
}
} else {
if(command == exitCommand) {
    destroyApp(false);
    notifyDestroyed();
}
}
if(command == pttpressCommand) {
    try {
        sdgc.sdgcAction(SdgcCall.PTT_PRESS);
    } catch(Exception exception) {
        notifyDestroyed();
    }
    fmAnswer.removeCommand(pttpressCommand);
    if(isEnd) {
        fmAnswer.removeCommand(endCommand);
    } else {
        fmAnswer.removeCommand(exitCommand);
    }
    fmAnswer.addCommand(pttreleaseCommand);
    if(isEnd) {
        fmAnswer.addCommand(endCommand);
    } else {
        fmAnswer.addCommand(exitCommand);
    }
}
if(command == pttreleaseCommand)
{
    try {
        sdgc.sdgcAction(SdgcCall.PTT_RELEASE);
    } catch(Exception exception) {
        notifyDestroyed();
    }
    fmAnswer.removeCommand(pttreleaseCommand);
    if(isEnd) {
        fmAnswer.addCommand(endCommand);
    } else {
        fmAnswer.removeCommand(exitCommand);
    }
    fmAnswer.addCommand(pttpressCommand);
    if(isEnd) {
        fmAnswer.addCommand(endCommand);
    } else {
        fmAnswer.addCommand(exitCommand);
    }
}
}

```



```

if(command == endCommand)
{
    try {
        sdgc.sdgcAction(SdgcCall.END_CALL);
    }catch(Exception exception) {
        notifyDestroyed();
    }
    isEnd = false;
    fmAnswer.removeCommand(endCommand);
    fmAnswer.addCommand(exitCommand);
}
}
}

```

5.3.4 Tips /

- The `getInstance()`, `startCallListener()`, and `callActionListener()` methods must be called from a thread separate from the MIDlet's main thread, so that they do not block the MIDlet from running. These methods block while waiting for new call state information.
- The `getInstance()` method should be used as early as possible in the application as this method actually notifies the ergonomics of a phone that the MIDlet will handle the incoming call. If the application does not call this method within 3-4 seconds, the ergonomics will assume that the application is taking too long to handle the incoming phone call, and will pass the incoming callback to the native ergonomics. This will be considered as a failure of an application. Three failures will select the native ergonomics of the phone to handle calls from then on. User can always go back to the settings to select the application again, which will set the failure count to zero again.
- For External Display support, use the ExternalDisplay API. It is a good practice if the application does `setCurrent()` on an ExternalDisplay canvas before it starts its listener thread. This allows the application to render itself on the external display when the flip is closed and allows for user interaction without opening the flip.
- The `getCallNumber()` method returns the phone number as a string. The application can use the PhoneBook API in order to get the person's name and the type of call (home/office/cell/other). Other APIs that can be used are Lighting and RecentCalls APIs.
- If the permission of a phone receiving application is set to `Ask` or `Never`, the phone ergonomics will handle the call as we do not want to ask for permission while user receives a phone call. So, it is recommended to have the permission set to `Always`.

5.3.5 Compiling & Testing Call Receiving MIDlets

The following tag should be listed into the JAD file of the MIDlet in order to register the application as a Call Receiving Application.

```
iDEN-MIDlet-Phone:
```

The path of a MIDlet to identify a particular MIDlet as a call receiving capable should be mentioned after the colon.

Here is a sample .JAD file.



MIDlet-1: InCall, , com.motorola.iden.call.InCall

MIDlet-Jar-Size: 72575

MIDlet-Jar-URL: InCall.jar

MIDlet-Name: InCall

MIDlet-Vendor: Motorola Inc.

MIDlet-Version: 1.0

MicroEdition-Configuration: CLDC-1.0

MicroEdition-Profile: MIDP-1.0

iDEN-MIDlet-Phone: com.motorola.iden.call.InCall

The lines above in **bold** have the same class path. This JAD file specifies the InCall MIDlet as a Call Receiving application.

After installing the Call Receiving MIDlet, the user will have to access the phone's settings via the main menu and set the newly installed application as the Call Receiving MIDlet. If the user does not select the MIDlet in the Java App Control menu, then the previously selected application (Phone Ergonomics if there is no other Call Receiving MIDlets) will handle the incoming phone call. The Java App Control option is not available until at least one Call Receiving MIDlet has been installed on the phone.

For Dispatch Calls:

The following tag should be listed into the JAD file of the MIDlet in order to register the application as a Call Receiving Application.

iDEN-MIDlet-Prvtcall:

The path of a MIDlet to identify a particular MIDlet as a call receiving capable should be mentioned after the colon.

Here is a sample .JAD file.

MIDlet-1: DspchCall, , com.motorola.iden.call.DspchCall

MIDlet-Jar-Size: 72575

MIDlet-Jar-URL: DspchCall.jar

MIDlet-Name: DspchCall

MIDlet-Vendor: Motorola Inc.

MIDlet-Version: 1.0



5.4 RecentCalls API

5.4.1 Overview



This API is available on these handsets only.

The RecentCalls API lets you access the phone's recent calls data. It lets you read and remove recent call entries. However, it does not let you add to the recent calls list.

5.4.2 Class Descriptions

The API for the RecentCalls is located in package `com.motorola.iden.recentcalls`.

Following is the class Hierarchy of RecentCalls API.

```
java.lang.Object
|
+ - com.motorola.iden.recentcalls.RecentCalls
|
+ - com.motorola.iden.recentcalls.RecentCallsEntry
```

Following is the Interface Hierarchy of the RecentCalls API.

```
com.motorola.iden.recentcalls.RCLListener
```

5.4.3 Method Descriptions

5.4.3.1 RecentCalls Methods

5.4.3.1.1 **entryAt**

Returns the RecentCallsEntry at the specified index.

```
public RecentCallsEntry entryAt(int index)
    throws IllegalArgumentException
```

Be sure to call `refreshList()` before using this function, to ensure that the RecentCalls list is up-to-date.



`index` is the number of the `RecentCallsEntry` to return. Note that the first entry is at index 0.

If `index` is greater than the index for the last `RecentCallsEntry`, this method throws `IllegalArgumentException`.

If this application does not have the right permissions to read the native recent calls list, this method throws a `SecurityException`.

5.5.3.1.2 `getStsMsg`

Returns the status message string associated with the specific alert.

```
Public String getStsMsg ()
```

The code example is as follows:

```
System.out.println(
    "The status message of the call for last Entry is "
    + myEntry.getStsMsg ());
```

5.4.3.1.3 `firstEntry`

Returns the first `RecentCallsEntry` if the `RecentCalls` list is not empty, null if the list is empty

```
public RecentCallsEntry firstEntry()
```

Be sure to call `refreshList()` before using this function, to ensure that the `RecentCalls` list is up-to-date.

If this application does not have the right permissions to read the native recent calls list, this method throws a `SecurityException`.

5.4.3.1.4 `lastEntry`

Returns the last `RecentCallsEntry` if the `RecentCalls` list is not empty, null if the list is empty.

```
public RecentCallsEntry lastEntry()
```

Be sure to call `refreshList()` before using this function, to ensure that the `RecentCalls` list is up-to-date.

If this application does not have the right permissions to read the native recent calls list, this method throws a `SecurityException`.

5.4.3.1.5 `refreshList`

Synchronizes this `RecentCalls` list with the phone's native list of recent calls.

```
public boolean refreshList()
```

This method synchronizes the `RecentCalls` list with the phone's native list of recent calls, adding calls to the `RecentCalls` list that have been added to the native list and removing calls from the native list that have been removed from the `RecentCalls` list.



Be sure to call this method when the RecentCalls list is first created, or when you call `removeEntryAt()` or `removeAll()`. Additionally, it's a good idea to call this method before you access the RecentCalls list, to ensure that the list is up-to-date.

This method returns true if the operation was successful, false otherwise. If the operation fails, the list is left in the same condition it was in before the operation.

If this application does not have the right permissions to read the native recent calls list, this method throws a `SecurityException`.

5.4.3.1.6 capacity

Returns the maximum number of recent calls that this phone can store.

```
public int capacity()
```

5.4.3.1.7 doesContain

Returns true if the specified RecentCallsEntry is in this RecentCalls list; false, otherwise.

```
public boolean doesContain(RecentCallsEntry myEntry)
```

Be sure to call `refreshList()` before using this function, to ensure that the RecentCalls list is up-to-date.

If this application does not have the right permissions to read the native recent calls list, this method throws a `SecurityException`.

5.4.3.1.8 indexOf

Returns the index of the specified RecentCallsEntry.

```
public int indexOf(RecentCallsEntry myEntry)
```

Be sure to call `refreshList()` before using this function, to ensure that the RecentCalls list is up-to-date.

If this application does not have the right permissions to read the native recent calls list, this method throws `SecurityException`.

5.4.3.1.9 currentUsage

Returns the number of RecentCallsEntries in this RecentCalls list.

```
public int currentUsage()
```

Be sure to call `refreshList()` before using this function, to ensure that the RecentCalls list is up-to-date.

If this application does not have the right permissions to read the native recent calls list, this method throws a `SecurityException`.



5.4.3.1.10 **isListEmpty**

Returns true if this RecentCalls list contains no entries; false, otherwise.

```
public boolean isListEmpty()
```

Be sure to call `refreshList()` before using this function, to ensure that the RecentCalls list is up-to-date.

If this application does not have the right permissions to read the native recent calls list, this method throws a `SecurityException`.

5.4.3.1.11 **numIncomingCalls**

Returns the number of incoming calls in this RecentCalls list.

```
public int numIncomingCalls()
```

If this RecentCalls list is empty, this method returns -1.

Be sure to call `refreshList()` before using this function, to ensure that the RecentCalls list is up-to-date.

If this application does not have the right permissions to read the native recent calls list, this method throws a `SecurityException`.

5.4.3.1.12 **numOutgoingCalls**

Returns the number of outgoing calls in this RecentCalls list.

```
public int numOutgoingCalls()
```

If this RecentCalls list is empty, this method returns -1.

Be sure to call `refreshList()` before using this function, to ensure that the RecentCalls list is up-to-date.

If this application does not have the right permissions to read the native recent calls list, this method throws a `SecurityException`.

5.4.3.1.13 **numMissedCalls**

Returns the number of missed calls in this RecentCalls list.

```
public int numMissedCalls()
```

If this RecentCalls list is empty, this method returns -1.

Be sure to call `refreshList()` before using this function, to ensure that the RecentCalls list is up-to-date.

If this application does not have the right permissions to read the native recent calls list, this method throws a `SecurityException`.

5.4.3.1.14 **removeEntryAt**

Deletes the RecentCallsEntry at the specified index from this RecentCalls list.

```
public boolean removeEntryAt(int entryNumber)
```

```
throws IllegalArgumentException
```



This method returns true if the operation is successful, false otherwise. If the operation fails, check to see if the entry is still there and try again.

You must call `refreshList()` after calling this method, to ensure that the phone's native recent calls list matches this RecentCalls list.

`index` is the number of the RecentCallsEntry to remove. Note that the first entry is at index 0.

If `index` is greater than the index for the last RecentCallsEntry, this method throws an `IllegalArgumentException`.

If this application does not have the right permissions to read the native recent calls list, this method throws a `SecurityException`.

5.4.3.15 removeAll

Deletes every RecentCallsEntry in this RecentCalls list.

```
public boolean removeAll()
```

You must call `refreshList()` after calling this method, to ensure that the phone's native recent calls list matches this RecentCalls list.

If this application does not have the right permissions to read the native recent calls list, this method throws a `SecurityException`.

5.4.3.16 setRCLListener

Set the recent calls listener to be the specified object.

```
public void setRCLListener(RCLListener l)
```

The recent call listener notifies the application when the phone's native recent calls list has changed by sending the specified object the `RCLActionListener()` method.

5.4.3.2 RecentCallsEntry Methods

5.4.3.2.1 getServiceCallType

Returns the service call type for this RecentCallsEntry.

```
public int getServiceCallType()
```

This method returns `JAVA_SERVICE_CALL_TYPE_PHONE`, `JAVA_SERVICE_CALL_TYPE_PRIVATE`, `JAVA_SERVICE_CALL_TYPE_TALKGROUP`, or `JAVA_SERVICE_CALL_TYPE_CALLALERT`.

5.4.3.2.2 getCallType

Returns the call type for this RecentCallsEntry.

```
public int getCallType()
```

If the service call type for this RecentCallsEntry is `JAVA_SERVICE_CALL_TYPE_PHONE`, this method returns `JAVA_CALL_TYPE_INCOMING`, `JAVA_CALL_TYPE_OUTGOING`, or `JAVA_CALL_TYPE_MISSED`. Otherwise, this method returns `CALL_TYPE_NO_STATUS`.



5.4.3.2.3 **getPhoneNumber**

Returns the phone number for this RecentCallsEntry.

```
public String getPhoneNumber()
```

5.4.3.2.4 **getDuration**

Returns the duration of this call in seconds.

```
public int getDuration()
```

5.4.3.2.5 **getMinute**

Returns the minute at which this call was made.

```
public int getMinute()
```

5.4.3.2.6 **getHour**

Returns the hour at which this call was made.

```
public int getHour()
```

5.4.3.2.7 **getDay**

Returns the day in which the call was made, as integer from 1 to 31.

```
public int getDay()
```

5.4.3.2.8 **getMonth**

Returns the month in which the call was made, as an integer from 1 to 12.

```
public int getMonth()
```

5.4.3.3 RCLListener Method

5.4.3.3.1 **RCLActionListener**

Called when the phone's native recent calls list is changed.

```
public void RCLActionListener()
```

You should implement this method to be notified when the RecentCalls list changes. There can be only one recent calls listener per MIDlet.

5.4.4 Code Examples

The following is the code example of RecentCalls API:

```
RecentCalls RCL =new RecentCalls();  
RecentCallsEntry myEntry = new RecentCallsEntry;  
  
if(RCL.isEmpty()){  
    System.out.println("This Recent Calls List is empty");  
} else {  
    try {
```



```

if(RCL.refreshList()){
    try {
        int currentUsage = RCL.currentUsage();
    } catch(Exception e) {
        System.out.println(
            "Exception thrown in currentUsage()" + e);
    }
    try {
        for(int i = 0; i < currentUsage; i++){

            myEntry = RCL.entryAt(i);
            System.out.println("Phone number for Entry " +
                i + " is " + myEntry.getPhoneNumber());
            System.out.println("Call Type for Entry " + i +
                " is " + myEntry.getCallType());
            System.out.println("Service Call Type for Entry "
                + i + " is " + myEntry.getServiceCallType());
            System.out.println(
                "The time of the call for Entry " + i +
                " is " + myEntry.getHour() + ":" +
                myEntry.getMinute());
            System.out.println(
                "The date of the call for Entry " + i +
                " is " + myEntry.getMonth() + "-" +
                myEntry.getDay());
            System.out.println(
                "The duration of the call for Entry " +
                i + " is " + myEntry.getDuration());
            try{
                if(RCL.contains(myEntry)){
                    System.out.println(
                        "contains() returned true");
                }else {
                    System.out.println(
                        "contains() returned false");
                }
            } catch(Exception e) {
                System.out.println(
                    "Exception thrown in contains()" + e);
            }
        }
    } catch(Exception e) {
        System.out.println("Exception thrown in entryAt()" + e);
    }
    try {
        myEntry = RCL.firstEntry();
        System.out.println("Phone number for first Entry is "
            + myEntry.getPhoneNumber());
        System.out.println("Call Type for first Entry is " +
            myEntry.getCallType());

        System.out.println(
            "Service Call Type for first Entry is " +

```



```

        myEntry.getServiceCallType());
System.out.println(
    "The time of the call for first Entry is " +
    myEntry.getHour() + ":" + myEntry.getMinute());
System.out.println(
    "The date of the call for first Entry is " +
    myEntry.getMonth() + "-" + myEntry.getDay());
System.out.println(
    "The duration of the call for first Entry is "
    + myEntry.getDuration());
} catch(Exception e) {
    System.out.println("Exception thrown in firstEntry()" + e);
}
try {
    myEntry = RCL.lastEntry();
    System.out.println("Phone number for last Entry is "
        + myEntry.getPhoneNumber());
    System.out.println("Call Type for last Entry is " +
        myEntry.getCallType());
    System.out.println(
        "Service Call Type for last Entry is " +
        myEntry.getServiceCallType());
    System.out.println(
        "The time of the call for last Entry is " +
        myEntry.getHour() + ":" + myEntry.getMinute());
    System.out.println(
        "The date of the call for last Entry is " +
        myEntry.getMonth() + "-" + myEntry.getDay());
    System.out.println(
        "The duration of the call for last Entry is "
        + myEntry.getDuration());
} catch(Exception e) {
    System.out.println("Exception thrown in lastEntry()" + e);
}
try {
    System.out.println("The number of incoming calls are " +
        RCL.numIncomingCalls());
} catch(Exception e) {
    System.out.println(
        "Exception thrown in numIncomingCalls()" + e);
}
try {
    System.out.println("The number of outgoing calls are " +
        RCL.numOutgoingCalls());
} catch(Exception e) {
    System.out.println(
        "Exception thrown in numOutgoingCalls()" + e);
}
try {
    System.out.println("The number of missed calls are " +
        RCL.numMissedCalls());
} catch(Exception e) {
    System.out.println(

```



```
        "Exception thrown in numMissedCalls() " + e);
    }
    try {
        for(int i = 0; i < 5 ; i++){
            RCL.removeEntryAt(i);
        }
    } catch(Exception e) {
        System.out.println(
            "Exception thrown in removeEntryAt() " + e);
    }
    try {
        RCL.removeAll();
    } catch(Exception e) {
        System.out.println("Exception thrown in removeAll() "
            + e);
    }
}
else {
    System.out.println("RefreshList returned false");
}
}
```



5.5 PhoneBook

5.5.1 Overview

The Java-based PhoneBook APIs let you access the user's phonebook data. The methods support such functionality as opening a phonebook, reading phonebook entries, creation a phonebook entry, importing a phonebook entry, removing specified phonebook entries, deleting all phonebook entries, determining available storage and so on.

5.5.2 Class Descriptions

The APIs for Phonebook are all located in package class `com.motorola.iden.udm`.

The following will be the Class Hierarchy for the UDM API:

```

java.lang.Object
|
+- com.motorola.iden.udm.UDM
|
+- com.motorola.iden.udm.PhoneBook
|
+- com.motorola.iden.udm.PhoneBookEntry
|
+- java.lang.Throwable
    |
    +- java.lang.Exception
        |
        +- com.motorola.iden.udm.UDMException
  
```

The following will be the Interface Hierarchy for the UDM and PhoneBook API:

```

com.motorola.iden.udm.UDMEntry
com.motorola.iden.udm.UDMList
  
```

5.5.3 Class Methods

5.5.3.1 UDM Method

Class for accessing the UDM databases on a device.

5.5.3.1.1 openPhoneBook

Returns a PhoneBook with the phone's native phonebook entries.

```
public static PhoneBook openPhoneBook(int mode) throws UDMException
```

This method returns a PhoneBook, sorted by name. `mode` must be either `READ_ONLY` or `READ_WRITE`. If you call this method and the phone's native phonebook is not ready (e.g. the SIM reads have not been completed), it throws a `UDMException`.



Calling this method is equivalent to calling `openPhoneBook(mode, NAME_SORT)`.

```
public static PhoneBook openPhoneBook(int mode, int sort)
    throws UDMException
```

This method returns a `PhoneBook` with the phone's phonebook entries, sorted either by name or speed dial number. `mode` must be either `READ_ONLY` or `READ_WRITE`. Sort must be either `NAME_SORT` or `SPEED_NUM_SORT`. Otherwise, this method throws an `IllegalArgumentException`. Note that if you sort by speed dial number, you may not be able to retrieve entries without a speed dial number. If you call this method and the phone's phonebook is not ready (e.g. the SIM reads have not been completed) it throws a `UDMException`.

The first time a MIDlet calls this method, it creates a new `PhoneBook` object with all the entries from the device's native phonebook. When a MIDlet calls it subsequently, it returns the same `PhoneBook` object, after repopulating the object with the entries from the native phonebook. Note that if your MIDlet has changed any `PhoneBookEntries` and hasn't committed them (with the `PhoneBookEntry.commit()`), those changes are lost.

To determine whether your application has modified a `PhoneBookEntry` without committing the change (with `PhoneBookEntry.commit()`) use `PhoneBookEntry.isModified()`. To determine whether the native `PhoneBook` database has been changed since a `PhoneBook` was created, use `PhoneBook.isCurrent()`.

5.5.3.2 PhoneBookEntry Methods

5.5.3.2.1 commit

Writes the data in the `PhoneBookEntry` to the phone's native phone book.

```
public void commit() throws UDMException
```

This method locks the native phone book, writes the data, and then unlocks the phonebook.

If this `PhoneBookEntry` contains only a name without a phone number, IP address, group ID, or email address, this method throws a `UDMException` with the string "Number Required". If this `PhoneBookEntry` lacks a name, this method throws a `UDMException` with the string "Name Required".

If the phone database is busy, this method throws a `UDMException` with the string "Native DB is busy". This often occurs after an application calls `deleteAllPhoneBookEntries()`. When this happens, try to sleep for a period of time and try again later. It takes approximately 30 seconds to clear the phone book.

5.5.3.2.2 isModified

Returns true if any of this element's fields have been modified since the element was retrieved or last committed.

```
public boolean isModified ()
```



5.5.3.2.3 getAvailSpeedNum

Returns the next or last available speed dial number.

```
public int getAvailSpeedNum(boolean reverseOrder)
    throws UDMException
```

Use this method to generate default values for the `SPEED_NUM` field. If `reverseOrder` is false, this method returns the lowest unused speed dial number. If `reverseOrder` is true, it returns the highest unused speed dial number.

If the SIM card type is GSM SIM or ENDEAVOR SIM and `reverseOrder` is true, this method throws a `UDMException` with the string "PhoneBook does not support reverse order".

5.5.3.2.4 getFieldDataType

Returns the data type for the given field ID

```
public int getFieldDataType(int fieldID) throws UDMException
```

Use this method to find the data types for field IDs that may have different types of data in each element. This table lists the data types for the fields in a PhoneBook entry:

Field ID	Field Data Type
TEL, SPEED_NUM, PRIV	UDMEntry.TYPED_STRING
REVISION	UDMEntry.DATE
EMAIL, FORMATTED_NAME, GRP, IP, HUB*	UDMEntry.STRING
RINGER	UDMEntry.INT

*FieldID HUB applies to the i325

5.5.3.2.5 getInt

Returns the value of the specified integer field.

```
public int getInt(int fieldID) throws UDMException
```

If `fieldID` is not `RINGER`, this method throws a `UDMException` with the string "Not supported field ID". To read the available ringers, use `com.motorola.iden.call.CallReceive.playRinger(int index)`. The value for `RINGER` is an integer from 0 to the 250, which maps to one of the ringers stored on the phone. The value of the default ringer is `0xff`.

5.5.3.2.6 setInt

Sets the value of the specified integer field.

```
public void setInt(int fieldID, int value) throws UDMException
```

If `fieldID` is not `RINGER`, this method throws a `UDMException` with the string "Not supported field ID". The value for `RINGER` is an integer from 0 to the 250 that maps to one of the ringers stored on the phone. The value of the default ringer is `0xff`.



5.5.3.2.7 getString

Returns the value of the specified string field.

```
public String getString(int fieldID) throws UDMException
```

If `fieldID` is not `FORMATTED_NAME`, `GRP`, `IP` or `EMAIL`, this method throws a `UDMException` with the string "Not supported field ID".



The `fieldID` for this handset is expanded to include `HUB`.

5.5.3.2.8 setString

Sets the value of the specified string field.

```
public void setString(int fieldID, String value)  
    throws UDMException
```

If `fieldID` is not `FORMATTED_NAME`, `GRP`, `IP` or `EMAIL`, this method throws a `UDMException` with the string "Not supported field ID".



The `fieldID` for this handset is expanded to include `HUB`

The `HUB` value can contain three or fewer digits that represent a value between 1 and 255.

Keep these pointers in mind when you set the value:

- The valid values for the `FORMATTED_NAME` field depend on the phone's SIM type. If SIM type is `FALCON` SIM and the name contains no Unicode characters, the maximum length of the name is 20 characters. If the name contains Unicode characters, the maximum length of the name is 10 characters.
- If the SIM type is any other SIM and the name contains no Unicode character, the maximum length of the name is 11 characters. If the name contains Unicode characters, the maximum length of the name is 5 characters.
- The `GRP` field can contain three or fewer digits that represent a value between 1 and 255.
- For the `IP` field, the value should be a valid IP address.
- For the `EMAIL` field, the value should be a valid email address.



5.5.3.2.9 getDate

Returns the value of the specified date field.

```
public long getDate(int fieldID) throws UDMException
```

If `fieldID` is not `REVISION`, this method throws a `UDMException` with the string "Not supported field ID".

5.5.3.2.10 setDate

Sets the value of the specified date field.

```
public void setDate(int fieldID, long value) throws UDMException
```

If `fieldID` is not `REVISION`, this method throws a `UDMException` with the string "Not supported field ID". The value should not be less than the date offset in milliseconds from January 1, 1970, to January 1, 1999.

5.5.3.2.11 setTypedString

Sets the value of the specified typed string field.

```
public void setTypedString(int fieldID, int typeID, String value)
throws UDMException
```

If `fieldID` is not `TEL`, `SPEED_NUM`, or `PRIV`, this method throws a `UDMException` with the string "Not supported field ID". If `typeID` is not a type supported by the field, this method throws a `UDMException`. A list of fields and their supported types is at the `PhoneBook` method `getSupportedTypes()`.

For the `TEL` field, the value should be a valid Phone Number and contain only the values in this character set: "0123456789+pwPW*#" . The maximum length of the number depends on the SIM type. If the SIM type is `FALCON`, the maximum length is 64 characters. Otherwise, the maximum length is 20 characters.

"P" or "p" inserts a three-second pause into the DTMF string. "W" or "w" stops sending DTMF tones until the user presses the Send key.

For the `PRIV` field, the value must be a valid Private Number that contains only digits. The maximum length is 18 characters.

5.5.3.3 PhoneBook Methods

5.5.3.3.1 importPhoneBookEntry

Adds the specified `PhoneBookEntry` to this `PhoneBook`.

```
public PhoneBookEntry importPhoneBookEntry(PhoneBookEntry element)
throws UDMException
```

If you opened the `PhoneBook` in read-only mode, this method throws a `UDMException`.

5.5.3.3.2 isSupportedField

Returns true if this `PhoneBook` supports the given field.

```
public boolean isSupportedField(int fieldID) throws UDMException
```



Here are the fields that this phone supports:

Fields	Supported or Not
PhoneBookEntry.TEL, PhoneBookEntry.SPEED_NUM, PhoneBookEntry.FORMATTED_NAME, PhoneBookEntry.REVISION	Supported.
PhoneBookEntry.NAME_FAMILY, PhoneBookEntry.NAME_GIVEN, PhoneBookEntry.NAME_OTHER, PhoneBookEntry.NAME_PREFIX, PhoneBookEntry.NAME_SUFFIX, PhoneBookEntry.NICKNAME, PhoneBookEntry.VOICE_NAME	Not supported.
PhoneBookEntry.PRIV, PhoneBookEntry.GRP	If SIM Type is SIM_GSM, it's supported. Otherwise, it's not supported.
PhoneBookEntry.IP	If SIM Type is SIM_CONDOR or SIM_FALCON, it's supported. Otherwise, it's not supported.
PhoneBookEntry.EMAIL, PhoneBookEntry.RINGER	If SIM Type is SIM_FALCON, they're supported. Otherwise, they're not supported.

5.5.3.3.3 isCurrent

Returns true if a PhoneBookEntry object has been created since the last native phonebook update.

```
public static boolean isCurrent()
```

5.5.3.3.4 getSupportedTypes

Returns an array of the supported types for the given field.

```
public int[] getSupportedTypes(int fieldID) throws UDMEException
```

Before you call this method, call `isSupportedField(int fieldID)` to make sure the field is supported.



Here are the fields that have types and which types they support:

fieldID	Supported types
PhoneBookEntry.TEL, PhoneBookEntry.SPEED_NUM	<p>If SIM type is FALCON, the types are PhoneBookEntry.TYPE_OTHER, PhoneBookEntry.TYPE_HOME, PhoneBookEntry.TYPE_MOBILE, PhoneBookEntry.TYPE_PAGER, PhoneBookEntry.TYPE_WORK_1, PhoneBookEntry.TYPE_WORK_2, and PhoneBookEntry.TYPE_FAX</p> <p>If SIM type is CONDOR, the types are PhoneBookEntry.TYPE_OTHER, PhoneBookEntry.TYPE_HOME, PhoneBookEntry.TYPE_MOBILE, PhoneBookEntry.TYPE_PAGER, PhoneBookEntry.TYPE_WORK_1, PhoneBookEntry.TYPE_FAX, and PhoneBookEntry.TYPE_MAIN.</p> <p>Otherwise, the only supported type is PhoneBookEntry.TYPE_MAIN</p>
PhoneBookEntry.PRIV	PhoneBookEntry.TYPE_PRIVATE

5.5.3.3.5 removePhoneBookEntry

Removes the specified PhoneBookEntry from the PhoneBook.

```
public void removePhoneBookEntry(PhoneBookEntry element)
```

throws UDMException

If the PhoneBookEntry is not in this PhoneBook, this method throws a UDMException with "PhoneBookEntry is not in PhoneBook".

If you opened the PhoneBook in read-only mode, this method throws a UDMException with the string "PhoneBook is Read only".

If the native phone database DB is busy, this method throws a UDMException with the string "Native DB is busy". This often occurs after an application calls `deleteAllPhoneBookEntries()`. When this happens, try to sleep for a period of time and try again later. It takes approximately 30 seconds to clear the phone book.

5.5.3.3.6 deleteAllPhoneBookEntries

Removes all PhoneBookEntries from the list.

```
public void deleteAllPhoneBookEntries() throws UDMException
```

If you opened the PhoneBook in read-only mode, this method throws a UDMException with the string "PhoneBook is Read only"

If the native phone database DB is busy, this method throws a UDMException with the string "Native DB is busy". This often occurs after an application calls



`deleteAllPhoneBookEntries()`. When this happens, try to sleep for a period of time and try again later. It takes approximately 30 seconds to clear the phone book.

5.5.3.3.7 `getAvailableStorage`

Returns an array listing the number of slots available in the native database.

```
public int[] getAvailableStorage() throws UDMException
```

The numbers returned depend on the type of SIM card in the device. For example, a device with an Endeavor SIM returns an array of three numbers representing the number of available phone number slots, private number slots, and talk group slots. A device with a Condor SIM would return an array with one number representing the total number of slots available. This table shows what's returned depending on the SIM card:

SIM Type	Total Available	Phone Available	Private Available	Talkgroup Available
i2000 GSM	N/A	100	100	30
Standard GSM	N/A	100	N/A	N/A
32K SIM	250	N/A	N/A	N/A
64K SIM	600	N/A	N/A	N/A

5.5.4 Code Examples

The following is the code example of PhoneBook:

```
/**
 * Demo program of Motorola iDEN SDK PhoneBook APIs
 * Filename: MyPhoneBook.java
 * <p></p>
 * <hr/>
 * <b>MOTOROLA and the Stylized M Logo are registered trademarks of
 * Motorola, Inc. Reg. U.S. Pat. & Tm. Off.<br>
 * &copy; Copyright 2003 Motorola, Inc. All Rights Reserved.</b>
 * <hr/>
 *
 * @version iDEN Phonebook demo 1.0
 * @author Motorola, Inc.
 */

import com.motorola.iden.udm.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.Enumeration;
import java.lang.Thread;

public class MyPhoneBook extends MIDlet implements CommandListener {

    private Form textform;
    private Command exitCommand, checkCommand;
```



```

private PhoneBook contacts;
private PhoneBookEntry contact;
private StringItem username;
int[] type;

/**
 * Print all contacts in a phonebook.
 * <p></p>
 * @param pbk Phonebook to be read
 */
public void printList(PhoneBook pbk)
{
    contacts = pbk;

    try
    {
        for (Enumeration v = contacts.elements(); v.hasMoreElements();)
        {
            /* Get one contact from phonebook */
            contact = (PhoneBookEntry)v.nextElement();
            type = contact.getFields();

            /* Get contact's name */
            username = new StringItem("name",
                contact.getString(PhoneBookEntry.FORMATTED_NAME));
            textform.append(username);

            for (int j= 0; j<type.length; j++)
            {
                /* Get String labels for the given field IDs */
                System.out.print("Fields "+type[j] + " , " +
                    contact.getFieldLabel(type[j]) + " ,");

                /* Get an integer array containing the supported type
                 * IDs for the given field ID
                 */
                int [] alltype = contacts.getSupportedTypes(type[j]);

                /* Get 3 types String fields from PhoneBookEntries. */
                if (alltype.length == 0)
                {
                    if (contact.getFieldDataType(type[j]) ==
                        UDMEEntry.STRING)
                        System.out.print(contact.getString(type[j]) +
                            " ,");
                    else if (contact.getFieldDataType(type[j]) ==
                        UDMEEntry.DATE)
                        System.out.print(contact.getDate(type[j])+
                            " ,");
                    else if (contact.getFieldDataType(type[j]) ==
                        UDMEEntry.INT)
                        System.out.print(contact.getInt(type[j])+

```




```

        " ,");
    }
    else
    {
        /* Get String fields with specific types
         * in the PhoneBookEntry.
         */
        for (int ii =0; ii<alltype.length; ii++ )
        {
            System.out.print(
                contact.getTypedString(type[j],
                    alltype[ii])+ " ,");
        }
    }
    System.out.println("\n");
}
System.out.println("\n");
}
}
catch (Exception ex)
{
    ex.printStackTrace();
}
}

public MyPhoneBook() {

    textform = new Form("Hello, PhoneBook!");
    exitCommand = new Command("exit", Command.EXIT, 2);
    checkCommand = new Command("check", Command.OK, 1);
    textform.addCommand(exitCommand);
    textform.addCommand(checkCommand);
    textform.setCommandListener(this);

    int[] type;
    Enumeration v;

    String title;
    try
    {
        /* Creates a PhoneBook by read and write mode */
        contacts = UDM.openPhoneBook(UDM.READ_WRITE);
        if (contacts != null)
        {
            /* Get the amount of entries (not individual numbers)
             * in the list
             */
            int no = contacts.getNumOfEntries();
            System.out.println("Number of entries is" + no);
        }

        /* Get an integer array the amount of slots available
         * on the native database.
         */
        int[] slots = contacts.getAvailableStorage();
    }
}

```



```
for (int i = 0; i < slots.length; i++)
{
    System.out.println(slots[i]);
}

int index =0;
printList(contacts);

/* Removes a specific PhoneBookEntry from the list. */
Enumeration e;
e = contacts.elements();
contact = (PhoneBookEntry)e.nextElement();
contacts.removePhoneBookEntry(contact);

/* Create a PhoneBookEntry for this PhoneBookEntry list. */
contact = contacts.createPhoneBookEntry();

contact.setString(PhoneBookEntry.FORMATTED_NAME,
    "abcdefghijklmopqrstu");
contact.setTypedString(PhoneBookEntry.TEL,
    PhoneBookEntry.TYPE_HOME, "6795588");

contact.setString(PhoneBookEntry.EMAIL,
    "someone@somesite.com");
contact.setString(PhoneBookEntry.IP, "127.0.0.1");
contact.setInt(PhoneBookEntry.RINGER, 2);
contact.setTypedString(PhoneBookEntry.TEL,
    PhoneBookEntry.TYPE_WORK_1, "1234567");
contact.commit();

Thread.sleep(200);

slots = contacts.getAvailableStorage();
for (int i= 0; i < slots.length; i++)
{
    System.out.println(slots[i]);
}

}
catch (Exception ex)
{
    ex.printStackTrace();
}

}

public void startApp()
{
    Display.getDisplay(this).setCurrent(textform);
}

public void pauseApp()
{
}
```



```
public void destroyApp(boolean unconditional)
{
}

public void commandAction(Command c, Displayable d) {
    if(c == exitCommand) {
        try {
            contacts.close();
        }
        catch(Exception t) {
        }
        notifyDestroyed();
    }
    else if (c == checkCommand)
    {
        System.out.println(PHONEBOOK.isCurrent());
    }
}
}
```

5.5.5 Compiling & Testing PhoneBook MIDlets

- Method `PhoneBook.isCurrent()` always returns true since there is no native support for this method.
- Method `PhoneBook.getAvailableStorage()` always returns an empty array since there is no native support for this method.
- Method `PhoneBookEntry.getAvailSpeedNum()` always returns 1 since there is no native support for this method.



5.6 Java PIM Package

5.6.1 Overview



i275
only

This API is only available on these handsets.

Package:

```
javax.microedition.pim  
com.motorola.iden.pim
```

Classes:

```
javax.microedition.pim.Contact.java  
javax.microedition.pim.ContactList.java  
javax.microedition.pim.Event.java  
javax.microedition.pim.EventList.java  
javax.microedition.pim.FieldEmptyException.java  
javax.microedition.pim.FieldFullException.java  
javax.microedition.pim.PIM.java  
javax.microedition.pim.PIMException.java  
javax.microedition.pim.PIMItem.java  
javax.microedition.pim.PIMList.java  
javax.microedition.pim.RepeatRule.java  
javax.microedition.pim.ToDo.java  
javax.microedition.pim.ToDoList.java  
javax.microedition.pim.UnsupportedFieldException.java  
com.motorola.iden.pim.PIMExtension
```

Note that this API is dependent upon the SIM card used on the handset. The following table lists the types of SIM cards available and the handsets on which they ship with. However, SIM cards are transferable so the actual SIM card on a handset can vary.



SIM type	Ships With Handset
64K	i730, i710, i830, i830e, i275, i285, i325, i355, i860, i605
32K	i85s, i88s, i30sx, i80s, i90c, i95cl
i2000 GSM	i2000
Standard GSM	Does not ship with iDEN handsets.

5.6.2 Package Description

The API for the PIM feature is located in package `javax.microedition.pim`.

The Java PIM package implementation supports Contact and Event in the JSR. No ToDo functionality is supported.

Class Hierarchy

- class `java.lang.Object`
 - class `javax.microedition.pim.PIM`
 - class `javax.microedition.pim.RepeatRule`
 - class `java.lang.Throwable`
 - class `java.lang.Exception`
 - class `javax.microedition.pim.PIMException`
 - class `java.lang.RuntimeException`
 - class `javax.microedition.pim.FieldEmptyException`
 - class `javax.microedition.pim.FieldFullException`
 - class `javax.microedition.pim.UnsupportedFieldException`

Interface Hierarchy

- interface `javax.microedition.pim.PIMItem`
 - interface `javax.microedition.pim.Contact`
 - interface `javax.microedition.pim.Event`
 - interface `javax.microedition.pim.ToDo`
- interface `javax.microedition.pim.PIMList`
 - interface `javax.microedition.pim.ContactList`
 - interface `javax.microedition.pim.EventList`
 - interface `javax.microedition.pim.ToDoList`



5.6.2.1 Class Description

5.6.2.1.1 javax.microedition.pim.PIM

```
public PIMItem[] fromSerialFormat(InputStream is, String enc)
```

This method creates and fills one or more PIM items from data provided in the given `InputStream` object where the data is expressed in a valid data format supported by this platform.

Tip: Parameter must be one of null, "UTF-8", "UTF_8", "ISO10646_1", "US_ASCII", "ISO10646", "ISO8859_1", "ISO_8859_1". Otherwise, `java.io.UnsupportedEncodingException` will be thrown out.

```
public String[] listPIMLists(int pimListType)
```

This method returns a list of all PIM List names for the given PIM list type.

Tip: If `pimListType` is `PIM.TODO_LIST`, a zero-length string array will be returned.

```
public String[] supportedSerialFormats(int pimListType)
```

This method returns the supported data formats for items used when converting a `PIMItem`'s data to and from data streams.

Tips: The implementation supports "VCARD/2.1" and "VCARD/3.0" for contact list.

The implementation supports "VCALENDAR/1.0" for event list.

```
public void toSerialFormat(PIMItem item, OutputStream os, String enc, String dataFormat)
```

This method writes the data from the given item to the given `OutputStream` object as Unicode characters in a format indicated by the `String` parameter.

Tip: Parameter must be one of null, "UTF-8", "UTF_8", "ISO10646_1", "US_ASCII", "ISO10646", "ISO8859_1", "ISO_8859_1". Otherwise, `java.io.UnsupportedEncodingException` will be thrown out.

5.6.2.1.2 javax.microedition.pim.ContactList

```
public void addCategory(java.lang.String category)
```

This method adds the provided category to the PIM list.

Tip: Categories are only supported on 64K SIM for `ContactList`.

Categories are case sensitive in the underlying system. I.e. "Work" and "WORK" are two different categories. The maximum length of a category is 16 Unicode characters.

`ContactList` supports up to 64 categories.

```
public int getFieldDataType(int field)
```

This method returns an `int` representing the data type of the data associated with the given field.

Following are the data types for extended fields in `ContactList`:




Extended field name	Data type
PIMExtension.MAILER	PIMItem.STRING
PIMExtension.TIMEZONE	PIMItem.STRING
PIMExtension.GEO	PIMItem.STRING
PIMExtension.ROLE	PIMItem.STRING
PIMExtension.SORT_STRING	PIMItem.STRING
PIMExtension.PRODID	PIMItem.STRING
PIMExtension.RINGER	PIMItem.INT

```
public int[] getSupportedFields()
```

This method gets all fields that are supported in this list.

The support fields and their corresponding native entries for each SIM type are listed below:

64K SIM:

Field	Native Entry	
Contact.TEL	Attribute	Native Entry
	Contact.ATTR_HOME	Home #
	Contact.ATTR_MOBILE	Mobile #
	Contact.ATTR_PAGER	Pager #
	Contact.ATTR_FAX	Fax #
	Contact.ATTR_OTHER	Other #
	Contact.ATTR_WORK	Work1 #
	PIMExtension.ATTR_WORK2	Work2 #
PIMExtension.PRIV	Private #	
PIMExtension.IP	IP	
PIMExtension.EMAIL	Email	
PIMExtension.GRP	Talk group	
 PIMExtension.SDG	SDG	
Contact.FORMATTED_NAME	Phonebook entry Name	
Contact.FORMATTED_ADDR	Unknown type	
Contact.NICKNAME	Unknown type	
Contact.NOTE	Unknown type	



Contact. ORG	Unknown type
Contact. TITLE	Unknown type
Contact. UID	Unknown type
Contact. URL	Unknown type
Contact. PHOTO_URL	Unknown type
Contact. PUBLIC_KEY_STRING	Unknown type
Contact. NAME	Unknown types
Contact. ADDR	Unknown type
Contact. BIRTHDAY	Unknown type
Contact. REVISION	Phonebook entry revision
Contact. PHOTO	Phonebook entry picture
PIMExtension CLASS	Unknown type
PIMExtension.TIMEZONE	Unknown type
PIMExtension GEO	Unknown type
PIMExtension ROLE	Unknown type
PIMExtension SORT_STRING	Unknown type
PIMExtension PRODID	Unknown type
PIMExtension RINGER	Ring tone index



32K SIM:

Field	Native Entry	
	Attribute	Native Entry
Contact.TEL	Contact.ATTR_HOME	Home #
	Contact.ATTR_MOBILE	Mobile #
	Contact.ATTR_PAGER	Pager #
	Contact.ATTR_FAX	Fax #
	Contact.ATTR_OTHER	Other #
	Contact.ATTR_WORK	Work1 #
	PIMExtension.ATTR_MAIN	Work2 #
PIMExtension.PRIV	Private #	
PIMExtension.IP	IP	
PIMExtension.GRP	Talk group	
Contact.FORMATTED_NAME	Phonebook entry Name	
Contact. REVISION	Phonebook entry revision	

i2000 GSM SIM:

Field	Native Entry
Contact.TEL	Telephone #
PIMExtension.PRIV	Private #
PIMExtension.GRP	Talk group
Contact.FORMATTED_NAME	Phonebook entry Name
Contact. REVISION	Phonebook entry revision

Standard GSM SIM:

Field	Native Entry
Contact.TEL	Telephone #
Contact.FORMATTED_NAME	Phonebook entry Name
Contact. REVISION	Phonebook entry revision



```
public boolean isSupportedAttribute(int field, int attribute)
```

This method indicates whether or not the given attribute is supported in this PIM list for the indicated field.

Tip:

- Fields other than Contact.TEL only support Contact.ATTR_NONE.

Below are the supported attributes for field Contact.TEL in ContactList:

SIM type	Supported attributes for Contact.TEL
64K	Contact.ATTR_HOME, Contact.ATTR_MOBILE, Contact.ATTR_PAGER, Contact.ATTR_FAX, Contact.ATTR_OTHER, Contact.ATTR_WORK, PIMExtension.ATTR_WORK2, Contact.ATTR_PREFERRED
32K	Contact.ATTR_HOME, Contact.ATTR_MOBILE, Contact.ATTR_PAGER, Contact.ATTR_FAX, Contact.ATTR_OTHER, Contact.ATTR_WORK, PIMExtension.ATTR_MAIN, Contact.ATTR_PREFERRED
i2000 GSM	PIMExtension.ATTR_MAIN
Standard GSM	PIMExtension.ATTR_MAIN

```
public int maxCategories()
```

This method returns the maximum number of categories that this list can have.

It returns 64 for 64K SIM, 0 for other SIM types.

```
public int maxValues(int field)
```

This method indicates the total number of data values that a particular field supports in this list.

Tips:

For 64K SIM and 32K SIM, it returns 7 for Contact.TEL, 1, for other fields.

For i2000 GSM SIM and Standard GSM SIM, it always returns 1.



5.6.2.1.3 javax.microedition.pim.Contact

```
public void commit()
```

This method persists the data in the item to its PIM list.

If this contact belongs to a category that are not present in its contact list, the category will be added to its contact list. This scenario happens when the contact is imported from a contact that was retrieved from a vcard stream.

```
public byte[] getBinary(int field, int index)
```

This method gets a binary data value for a field from the item.

Tip: For Contact.PHOTO field on 64K SIM, if the picture is forward locked in the MRM database, a zero-length byte[] will be returned.

```
public int maxCategories()
```

This method returns the maximum number of categories that this item can be assigned to.

It returns 1 for 64K SIM, 0 for others.

5.6.2.1.4 javax.microedition.pim.EventList

```
public void addCategory(java.lang.String category)
```

This method adds the provided category to the PIM list.

Categories are case sensitive in the underlying system. I.e. "Work" and "WORK" are two different categories. The maximum length of a category is 16 Unicode characters.

EventList supports up to 64 categories.

```
public int getFieldDataType(int field)
```

This method returns an int representing the data type of the data associated with the given field.

Below is the data type for extended fields in EventList:

Extended field name	Data type
PIMExtension.MIDLET	PIMItem.STRING
PIMExtension.MIDLET_SUITE	PIMItem.STRING
PIMExtension.STYLE	PIMItem.STRING
PIMExtension.RINGER	PIMItem.INT



```
public int[] getSupportedFields()
```

This method gets all fields that are supported in this list.

Following are the support fields:

Field
PIMExtension.STYLE
PIMExtension.MIDLET_SUITE
PIMExtension.MIDLET
PIMExtension.RINGER
Event.ALARM
Event.END
Event.LOCATION
Event.REVISION
Event.START
Event.SUMMARY
Event.UID

```
public boolean isSupportedAttribute(int field, int attribute)
```

This method indicates whether or not the given attribute is supported in this PIM list for the indicated field.

Fields only support `Event.ATTR_NONE`.

```
public int maxCategories()
```

This method returns 64 categories, the maximum number of categories this list can have.

5.6.3 Code Examples

The following is a code example using the PIM API:

```
void fromSerialFormatTest()
{
    byte[] vCardCorrect = new String("BEGIN:vCard
\r\nTEL;CELL;WORK;HOME:5555555\r\nTEL;TYPE=VOICE;WORK;HOME:5555555\r
\nORG;\r\n
ENCODING=BASE64;HOME;WORK:Loon:\r\n\nVERSION:2.1\r\nN:Doe;John\r\n
\r\nEND:vCard
BEGIN:vCard\r\nFN:a16551\r\nTEL;PAGER:5555555\r\nEND:vCard").getBytes();

    InputStream is;
```



```
try
{
    is = (InputStream )new ByteArrayInputStream(vCardCorrect);
    PIM.getInstance().fromSerialFormat(is, null);
}
catch(UnsupportedEncodingException e)
{
}
catch(Exception e)
{
}
}

void itemsTest()
{
    try
    {
        ContactList theList = (ContactList
)PIM.getInstance().openPIMList(PIM.CONTACT_LIST, PIM.READ_WRITE);

        int num = 0;
        Enumeration enum = theList.items();
        while(enum.hasMoreElements())
        {
            Contact c = (Contact )enum.nextElement();
            if(c.getString(Contact.FORMATTED_NAME, 0).equals("First
PERSON") ||
                c.getString(Contact.FORMATTED_NAME,
0).equals("Second Guy") ||
                c.getString(Contact.FORMATTED_NAME, 0).equals("Third
Person"))
            {
                ++num;
            }
        }
        theList.close();
    }
}
```



```
    }
    catch(Exception e)
    {
    }
}

void removeValueTest()
{
    int i;
    try
    {
        ContactList theList = (ContactList
)PIM.getInstance().openPIMList(PIM.CONTACT_LIST, PIM.READ_WRITE);
        Enumeration enum = theList.items();
        Contact c = (Contact )enum.nextElement();
        int[] fields1 = c.getFields();
        for(i=c.countValues(Contact.FORMATTED_NAME); i>0; i--)
        {
            c.removeValue(Contact.FORMATTED_NAME, 0);
        }
        for(i=c.countValues(Contact.TEL); i>0; i--)
        {
            c.removeValue(Contact.TEL, 0);
        }
        theList.close();
    }
    catch(Exception e)
    {
    }
}

void removeEventTest()
{
    try {
```



```
        EventList eventList =
(EventList) PIM.getInstance().openPIMList(PIM.EVENT_LIST,
PIM.READ_WRITE);

        Enumeration rmevents = eventList.items();
        Event rmevent = eventList.createEvent();
        while(rmevents.hasMoreElements())
        {
            rmevent = (Event)rmevents.nextElement();
            eventList.removeEvent(rmevent);
            rmevents = eventList.items();
        }
        catch(Exception pe)
        {
        }
    }
```



6

File System and Storage

6.1 Overview

- Record Management System
- MIDP 2.0 File Input / Output
- Secure File Input / Output
- File Connection
- Java ZIP

6.2 MIDP 2.0 Record Management System (RMS)

6.2.1 Overview

The most common mechanism for persistently storing data on a MIDP device is through RMS. RMS lets a MIDlet store variable length records on the device. Those records are accessible to any MIDlet in the MIDlet suite, and also to MIDlets outside of the MIDlet suite if permission is given when the record is created. The RMS implementation on iDEN handsets is MIDP 2.0 compliant.

MIDlets within a suite can access each other's record stores directly. New APIs in MIDP 2.0 let you explicitly share record stores if the MIDlet creating the record store chooses to give such permission. Sharing is accomplished through the ability to name a record store created by another MIDlet suite.

You define access controls when you create a record store that's to be shared. Access controls are enforced when RecordStores are opened. The access modes allow private use or shareable with any other MIDlet suites.

6.2.2 Class Description

The API for the RecordStore is located in the package `javax.microedition.rms`.



6.2.3 Code Examples

The following simple code example opens a record store. If any exception occurs it is caught.

```
try {
    System.out.println("Opening RecordStore " + rsName + " ...");

    //try to open a record Store
    recordStore = RecordStore.openRecordStore(rsName, true);

    //keep a note for the last modified time for record store
    Date d = new Date(recordStore.getLastModified());
    System.out.println(recordStore.getName()+"modified last time: " +
        d.toString());
}
catch (RecordStoreException rse) {
    //process the IOException
}
```

The following simple code example will open (and possibly create) a record store that can be shared with other MIDlet suites. The record store is owned by the current MIDlet suite. The authorization mode is set when the record store is created, as follows:

AUTHMODE_PRIVATE allows only the MIDlet suite that created the record store to access it. This case behaves identically to `openRecordStore(recordStoreName, createIfNecessary)`.

AUTHMODE_ANY allows any MIDlet to access the record store. Note that this makes your record store accessible by any other MIDlet on the device. This could have privacy and security issues depending on the data being shared. Please use carefully.

```
try {
    System.out.println("Opening RecordStore " + rsName + " ...");

    //try to open a record store
    recordStore = RecordStore.openRecordStore(rsName,true,
        (byte)RecordStore.AUTHMODE_ANY, true);

    //keep a note for the last modified time for record store
    Date d = new Date(recordStore.getLastModified());
    System.out.println(recordStore.getName()+"modified last time: " +
        d.toString());
} catch (RecordStoreException rse) {
    //process the IOException
}
```

6.2.4 Tips /

- It is much faster to read and write in big chunks than it is to do so in small chunks. The optimal size for reading and writing is 512 bytes.
- Whenever you close a record store, `close()` does not return until all the pending writes have been written. A successful `close()` call guarantees that the data was written. It is



then safe to power off the phone. Because of this, `close()` may take a while to return. Therefore, if a record store is opened and closed for every write, performance will slow down greatly.

6.2.5 Caveats

- iDEN handsets support a maximum of 2048 record stores. If there is no file space available, you cannot create extra record stores or records. Once the phone contains 2048 record stores, you cannot create more. MIDI ringers, voice notes, wallpapers, PNG images included with a MIDI are all files. If a MIDlet has many images, such as sprites used in animations, it may be advantageous to have them all in one image file and use clipping to display only what you need.
- A record store can be of any size as long as there is file space available. A zero byte record store is also allowed.
- Each MIDlet suite is guaranteed to be able to open at least 5 files or record stores simultaneously.
- There is an additional pool of 16 files and record stores that can be opened. This pool is shared among all MIDlet suites, giving a MIDlet suite the potential to simultaneously open 21 files or record stores.

6.2.6 Compiling and Testing RMS MIDlets

This is a standard MIDP 2.0 package so there is no need for stub classes to compile the MIDlet with RMS APIs.



6.3 MIDP 2.0 File I/O and Secure File I/O

6.3.1 Overview

The objective of the File I/O and secure File I/O API is to provide a generic platform for the Java developer to use to open, read, write, append and delete a file sequentially. The goal is to provide UNIX-like file access APIs, as a simple alternative to Record Management System (RMS). This lets MIDlets save information between invocations; this is called "persistent storage." Examples include:

- Saving data such as notes, phone numbers, tasks, and so on.
- Keeping a history of recent URLs



Secure File I/O API provides a generic platform for the Java developer to protect the persistent storage with password protection.



Handsets that do not provide Secure File I/O provide all functionality specified for unsecured File I/O.

6.3.2 Class Description

The File I/O and Secure File I/O APIs are located in package `javax.microedition.io`.

6.3.3 Method Description

6.3.3.1 Connector Method

6.3.3.1.1 `open`

Opens or deletes the specified file.

```
public static Connection open(String name)
    throws IOException
```

```
public static Connection open(String name, int mode)
    throws IOException
```

```
public static Connection open(String name, int mode,
    boolean timeouts) throws IOException
```



Opening a file gives your application exclusive access to that particular file until it is explicitly closed or the program ends. Opening a secure file gives your application password-protected access to that particular file until it is explicitly closed or the program ends.

`name` is a URL that contains the name of the file to open, and can also include keywords that specify the mode in which to open it. Here are some examples:

- `"file://temp.txt"` specifies that file is to be opened in the default mode, which is `READ_WRITE`.
- `"file://temp.txt;APPEND"` specifies that file is to be opened in an `APPEND` mode.
- `"sfile://temp.txt;PASSWORD=313"` specifies that the file is a secure file to be opened with the password 313 and in the default mode, `READ_WRITE`.
- `"sfile://temp.txt;PASSWORD=313;APPEND"` specifies that the file is a secure file to be opened with the password 313 and in an `APPEND` mode.

You can also delete a file with the `DELETE` keyword. Note that all the `InputStreams`, `OutputStreams` and `StreamConnections` associated with a file should be closed before deleting the file. If a file cannot be deleted, these methods throw an `IOException`. Here are some examples of name parameters that delete a file:

`"file://temp.txt;DELETE"` deletes `temp.txt`.

`"sfile://temp.txt;PASSWORD=313;DELETE"` deletes the secure file `temp.txt` with the password 313.

`mode`, if included, must have one of these three values: `READ`, `WRITE`, or `READ_WRITE`.

`timeout` has no effect on the method call and is ignored.

These are five basic steps for reading and writing a file:

1. Open the file using the `open()` method of `Connector` class. This returns a `StreamConnection` object for file. Otherwise, an `IOException` is thrown.
2. Get the output stream using the `openOutputStream()` method of `OutputConnection`.
3. Get the input stream using the `openInputStream()` method of `InputConnection`.
4. Once the connection has been established, simply use the normal methods of any input or output stream to read and write data.
5. Close the file using the `close()` method of `Connection`.
6. For more information, see the Javadocs.



6.3.4 Code Examples

6.3.4.1 Example # 1 (File/Secure File Snippet)

The following example shows how to open a file, write bytes to the file and read the same number of bytes.

```
StreamConnection sc = null;
InputStream is = null;
OutputStream os = null;

//For regular file
String name = "file://temp.txt";

//For secure file
String name = "sfile://temp.txt;PASSWORD=4509";
try {
    // open a file, default mode: READ_WRITE
    sc = (StreamConnection)Connector.open(name);

    // get OutputStream
    os = sc.openOutputStream();

    // get InputStream
    is = sc.openInputStream();
    String b = "Hello World";

    // write the bytes
    os.write(b.getBytes());
    int dataAvailable = is.available();
    byte [] b1 = new byte[dataAvailable];

    // read the bytes
    is.read(b1);
} finally {
    if (sc != null)
        sc.close();
    if (is != null)
        is.close();
    if (os != null)
        os.close();
}
```



6.3.4.2 Example # 2 (Complete File MIDlet Code)

The following example is a simple MIDlet that will provide the overall operation of the file I/O interface and how most of the APIs can be used. The MIDlet also shows a simple alternative to RMS to store data as a persistent storage.

```
import javax.microedition.io.*;
import java.io.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class Example2 extends MIDlet implements CommandListener{
    /**
     * List of available tests
     */
    StreamConnection sc;
    String[] testList = {"file to w/r", "setData",
        "write/append/read", "delete"};
    TextBox tf1;
    TextBox tf2;

    /**
     * Reference to Display object associated with this Display
     */
    Display myDisplay;

    /* default file name */
    String fileURL = "temp.txt";

    /*default amount of data*/
    int dataNum = 0;

    /*default string to write in file*/
    String stringNum ="Hello World";

    /**
     * The output screen
     */
    Form myOutput;

    /**
     * The list of tests
     */
    List myList;

    /**
     * Ok command to indicate a test was selected
     */
    Command okCommand;

    /**
     * Create NetTests
     */
}
```



```

public Example2( ) {

}

/**
 * Start running
 */
protected void startApp() {
    myDisplay = Display.getDisplay(this);
    myOutput = new Form("Results");
    myList = new List("Select test:", List.IMPLICIT, testList,
        null);
    okCommand = new Command("OK", Command.OK, 1);
    myOutput.addCommand(okCommand);
    myList.addCommand(okCommand);
    myOutput.setCommandListener(this);
    myList.setCommandListener(this);
    tf1 = new TextBox("file to w/r", fileURL, 28, TextField.ANY);
    tf1.addCommand(okCommand);
    tf1.setCommandListener(this);
    tf2 = new TextBox("Set Data to Send", stringNum, 28,
        TextField.ANY);
    tf2.addCommand(okCommand);
    tf2.setCommandListener(this);
    myDisplay.setCurrent(myList);
}

/**
 * Stop running
 */
protected void pauseApp() {
}

/**
 * Destroy App
 */
protected void destroyApp(boolean unconditional) {
}

/**
 * Handle ok command
 */
public void commandAction(Command c, Displayable s) {
    if ((s == tf1) || (s == tf2)) && (c == okCommand) {
        if(s==tf1) fileURL = tf1.getString();
        if(s==tf2) {
            /* data in the string form */
            stringNum = tf2.getString();
            /* convert the string into the integer form */
            dataNum = stringNum.length();
        }
    }
    if (s == myList) {
        switch (((List)s).getSelectedIndex()) {

```



```

        case 0:
            myDisplay.setCurrent(myOutput);
            setFileName();
            break;
        case 1:
            myDisplay.setCurrent(myOutput);
            setData();
            break;
        case 2:
            myDisplay.setCurrent(myOutput);
            readWrite();
            break;
        case 3:
            myDisplay.setCurrent(myOutput);
            deleteFile();
            break;
    }
} else {
    myDisplay.setCurrent(myList);
}
}

private void setFileName() {
    myDisplay.setCurrent(tf1);
}

private void setData() {
    myDisplay.setCurrent(tf2);
}

private void readWrite() {
    int length = dataNum;
    byte[] message = new byte[length];
    message = stringNum.getBytes();
    OutputStream os = null;
    InputStream is = null;
    try {
        //open a file in the mode APPEND
        sc = (StreamConnection)Connector.open("file://" +
            fileURL + ";" + "APPEND");

        //get OutputStream
        os = sc.openOutputStream();

        //get InputStream
        is = sc.openInputStream();

        //write the bytes to the file
        os.write(message);
        myOutput.append("write/append done");

        //create an array to store available data
        // from the file
        byte [ ] b1 = new byte[is.available()];

```




```
//read the bytes
is.read(b1);
String readString = new String(b1);

//printout the data in the phone screen
myOutput.append(readString);
myOutput.append("read finished");

//close all the opened streams
if (is != null)
    is.close();
if (os != null)
    os.close();
if (sc != null)
    sc.close();
} catch (Exception e) {
    System.out.println("Exception: " + e.getMessage());

    try {
        if (is != null)
            is.close();
        if (os != null)
            os.close();
        if (sc != null)
            sc.close();
    }
    catch(Exception ex) {
    }
}

private void deleteFile() {
    try {

        //open a file in the delete mode
        //by doing this existing file is eventually delete
        sc = (StreamConnection)Connector.open("file://" +
            fileURL + ";" + "DELETE");

        //close the stream
        if (sc != null)
            sc.close();
        myOutput.append("file deleted");
    } catch (Exception ex1 ) {
        System.out.println("Exception: " + ex1.getMessage());
        try {
            if (sc != null)
                sc.close();
        }
        catch(Exception ex) {
        }
    }
}
```



```
    }
}
```

6.3.4.3 Example # 3 (Complete Secure File MIDlet Code)

The following example is a simple MIDlet that provides the overall operation of the secure file I/O interface and how most of the APIs can be used. The MIDlet also shows a simple alternative to RMS to store data as a persistent storage with password protection.

```
import javax.microedition.io.*;
import java.io.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class Example4 extends MIDlet implements CommandListener{

    /**
     * List of available tests
     */
    StreamConnection sc;
    String[] testList = {"file to w/r","setData","write/append/read","delete"};
    TextBox tf1;
    TextBox tf2;

    /**
     * Reference to Display object associated with this Display
     */
    Display myDisplay;

    /* default file name */
    String fileURL = "temp.txt;PASSWORD=1413";

    /*default amount of data*/
    int dataNum = 0;

    /*default string to write in file*/
    String stringNum ="Hello World";

    /**
     * The output screen
     */
    Form myOutput;

    /**
     * The list of tests
     */
    List myList;

    /**
     * Ok command to indicate a test was selected
     */
    Command okCommand;
```



```

/**
 * Create NetTests
 */
public Example4( ) {

}

/**
 * Start running
 */
protected void startApp() {
    myDisplay = Display.getDisplay(this);
    myOutput = new Form("Results");
    myList = new List("Select test:", List.IMPLICIT, testList,
        null);
    okCommand = new Command("OK", Command.OK, 1);
    myOutput.addCommand(okCommand);
    myList.addCommand(okCommand);
    myOutput.setCommandListener(this);
    myList.setCommandListener(this);
    tf1 = new TextBox("file to w/r", fileURL, 28, TextField.ANY);
    tf1.addCommand(okCommand);
    tf1.setCommandListener(this);
    tf2 = new TextBox("Set Data to Send", stringNum, 28,
        TextField.ANY);
    tf2.addCommand(okCommand);
    tf2.setCommandListener(this);
    myDisplay.setCurrent(myList);
}

/**
 * Stop running
 */
protected void pauseApp() {
}

/**
 * Destroy App
 */
protected void destroyApp(boolean unconditional) {
}

/**
 * Handle ok command
 */
public void commandAction(Command c, Displayable s) {
    if ((s == tf1) || (s == tf2)) && (c == okCommand) {
        if(s==tf1) fileURL = tf1.getString();
        if(s==tf2) {
            /* data in the string form */

```



```

        stringNum = tf2.getString();
        /* convert the string into the integer form */
        dataNum = stringNum.length();
    }
}
if (s == myList) {
    switch (((List)s).getSelectedIndex()) {
    case 0:
        myDisplay.setCurrent(myOutput);
        setFileName();
        break;
    case 1:
        myDisplay.setCurrent(myOutput);
        setData();
        break;
    case 2:
        myDisplay.setCurrent(myOutput);
        readWrite();
        break;
    case 3:
        myDisplay.setCurrent(myOutput);
        deleteFile();
        break;
    }
} else {
    myDisplay.setCurrent(myList);
}
}
private void setFileName() {
    myDisplay.setCurrent(tf1);
}
private void setData() {
    myDisplay.setCurrent(tf2);
}
private void readWrite() {
    int length = dataNum;
    byte[] message = new byte[length];
    message = stringNum.getBytes();
    OutputStream os = null;
    InputStream is = null;
    try {
        //open a file in the mode APPEND
        sc = (StreamConnection)Connector.open("sfile://" +
            fileURL + ";" + "APPEND");

        //get OutputStream
        os = sc.openOutputStream();

        //get InputStream
        is = sc.openInputStream();

        //write the bytes to the file
        os.write(message);
    }
}

```



```
myOutput.append("write/append done");

//create an array to store available data from the file
byte [ ] b1 = new byte[is.available()];

//read the bytes
is.read(b1);
String readString = new String(b1);

//printout the data in the phone screen
myOutput.append(readString);
myOutput.append("read finished");

//close all the opened streams

if (is != null)
    is.close();
if (os != null)
    os.close();
if (sc != null)
    sc.close();
} catch (Exception e) {
    System.out.println("Exception: " + e.getMessage());
    try {
        if (is != null)
            is.close();
        if (os != null)
            os.close();
        if (sc != null)
            sc.close();
    }
    catch(Exception ex) {
    }
}

private void deleteFile() {
    try {
        //open a file in the delete mode
        //by doing this existing file is eventually delete
        sc = (StreamConnection)Connector.open("sfile://" +
            fileURL + ";" + "DELETE");

        //close the stream
        if (sc != null)
            sc.close();
        myOutput.append("file deleted");
    } catch (Exception ex1 ) {
        System.out.println("Exception: " + ex1.getMessage());
        try {
            if (sc != null)
                sc.close();
        }
    }
}
```



```
        catch(Exception ex) {  
        }  
    }  
}
```

6.3.5 Tips /

- Like RMS, it is much faster to read and write in big chunks than it is to do so in small chunks. The optimal size for reading and writing is 512 bytes.
- File I/O is a simple alternative to Record Management System (RMS). When used effectively, direct file I/O can speed up storing and retrieving data.
- After creating a file from a MIDlet suite, the file is associated with the current MIDlet suite only. No other MIDlet suite can access it.
- If a MIDlet suite is updated to another version, then the file(s) associated with the current version of MIDlet suite can be maintained for the new version to use. The user is prompted to keep the old data, or delete it.
- If a MIDlet suite is deleted, all files associated with it are deleted.
- It is a MIDlet's responsibility to coordinate the use of multiple threads to access a file since unintended consequences may result.
- Whenever you close a file, the `close()` command will not return until all the pending writes have been completed; thus closing a file guarantees that all of the data is written. It is then safe to power off the device. One consequence is that the `close()` command may take a while to return. Therefore, if you open and close the file every time a write is required, performance will be greatly affected.
- Secure File I/O API has all the functionality of regular File I/O, but in addition it provides password protection to the persistent storage.

6.3.6 Caveats

- This File Access System is a sequential system. This means once you write a particular chunk of data to the file, you can't go back and manipulate it.
- Theoretically, the maximum number of files supported is 2048. If there is no file space available, one cannot create extra files. And once the phone contains 2048 files, it will not be able to create more. MIDI ringers, voice notes, wallpapers, PNG images included with a MIDlet are all files. If a MIDlet has many images, such as sprites used in animations, it may be advantageous to have them all in one image file and use clipping to display only what you need.
- The file name can contain up to 32 alphanumeric characters.
- A file can be of any size as long as file space is available.
- A zero-byte file is not allowed. Unwanted behavior may occur when a file is opened and nothing is written into it before closing it.
- It is recommended that only 21 files remain open at one time. Exceeding the maximum number of opened files can result in unintended behavior.



- The `InputStream` method `markSupported()` returns true only if the file open mode is `READ` or `APPEND`. This means that in any other file open mode, the `mark()` and `reset()` methods do not work.
- In the `InputStream` method `mark()`, the `readlimit` argument tells the input stream to allow that many bytes to be read before the mark position gets invalidated. Since this operation is on a file, “remembering” the entire contents of stream/file does not incur any type of cost, so the `readlimit` parameter is ignored, preventing mark position invalidation.
- A secure file can only be opened with the correct password. A wrong password cannot open the file and will throw an exception.
- A password can have length up to 32 alphanumeric characters.

6.3.7 Compiling and Testing File/Secure File MIDlets

The file I/O APIs and secure file I/O APIs are based off of generic `Connector.Open()` APIs, so there is no need of any stub classes to compile the MIDlet with these APIs.



6.4 FileConnection

6.4.1 Overview

The primary goal of the FileConnection APIs is to provide access to file systems on devices and/or mounted memory cards. File system connectivity through the Generic Connection Framework may be supported by an implementation if the target device has the necessary underlying operating system and hardware support for file systems. Connections to a file system may be opened to file systems located either on memory cards or in a device's memory, depending on device and operating system limitations.

6.4.2 Package javax.microedition.io.file

APIs for FileConnection are all located in package `javax.microedition.io.file`.

Class Summary	
FileSystemRegistry	The FileSystemRegistry is a central registry for file system listeners interested in the adding and removing (or mounting and unmounting) of file systems on a device.
ConnectionClosedException	Represents an exception thrown when a method is invoked on a file connection but the method cannot be completed because the connection is closed.
IllegalModeException	Represents an exception thrown when a method is invoked requiring a particular security mode (e.g. READ or WRITE), but the connection opened is not in the mode required. The application does pass all security checks, but the connection object is in the wrong mode.
Interface Summary	
FileConnection	This interface is intended to access files or directories that are located on file system on a device.
FileSystemListener	This class is used for receiving status notification when adding or removing a file system root.

6.4.2.1 Package Tree

6.4.2.1.1 Class Hierarchy

The following will be the Class Hierarchy for the FileConnection API:

```
interface javax.microedition.io.Connection
interface javax.microedition.io.InputConnection
```




```
interface javax.microedition.io.StreamConnection (also extends
javax.microedition.io.OutputConnection)

interface javax.microedition.io.file.FileConnection

interface javax.microedition.io.OutputConnection

interface javax.microedition.io.StreamConnection (also extends
javax.microedition.io.InputConnection)

interface javax.microedition.io.file. FileConnection

interface javax.microedition.io.file.FileSystemListener
```

6.4.2.2 CLASS **FileSystemRegistry**

6.4.2.2.1 **addFileSystemListener**

```
static boolean addFileSystemListener
(javax.microedition.io.file.FileSystemListener listener);
```

This method is used to register a **FileSystemListener** that is notified in case of adding and removing a new file system root.

6.4.2.2.2 **removeFileSystemListener**

```
public static boolean removeFileSystemListener
(javax.microedition.io.file.FileSystemListener listener)
```

This method is used to remove a registered **FileSystemListener**. If file systems are not supported on a device, false will be returned from the method.

6.4.2.2.3 **listRoots**

```
public static java.util Enumeration listRoots()
```

This method returns the currently mounted root file systems on a device

Tip: Only one root “.” is supported.

6.4.2.3 **FileConnection** interface

6.4.2.3.1 **openInputStream ()**

```
public java.io.InputStream openInputStream() throws
java.io.IOException
```

This method opens and returns an input stream for a connection.

Tip: **Connector.open()** will be used first and the connection shall be checked by **fileconnection.exists()**. If a connection does not exist, an exception will be thrown.



6.4.2.3.2 `openDataInputStream ()`

```
public java.io.DataInputStream openDataInputStream() throws  
java.io.IOException
```

This method opens and returns a data input stream for a connection.

Tip: `Connector.open()` will be used first and the connection shall be checked by `fileconnection.exists()`. If a connection does not exist, an exception will be thrown.

6.4.2.3.3 `openOutputStream ()`

```
public java.io.OutputStream openOutputStream() throws  
java.io.IOException
```

This method opens and returns an output stream for a connection.

Tip: `Connector.open()` will be used first and the connection shall be checked by `fileconnection.exists()`. If the connection does not exist an exception will be thrown.

6.4.2.3.4 `openDataOutputStream ()`

```
public java.io.DataOutputStream openDataOutputStream() throws  
java.io.IOException
```

This method opens and returns a data output stream for a connection.

Tip: `Connector.open()` will be used first and the connection shall be checked by `FileConnection.exists()`. If the connection does not exist an exception will be thrown.

6.4.2.3.5 `openOutputStream ()`

```
public java.io.OutputStream openOutputStream(int byteOffset) throws  
java.io.IOException
```

This method opens an output stream and positions it at the indicated byte offset in the file.

Tips:

- The `Connector.open()` will be used first and the connection shall be checked by `FileConnection.exist()`. If the connection does not exist an exception will be thrown.
- The `byteOffset` can't access the connection length. If the `byteOffset` accesses the range of the file length, an exception will be thrown.

6.4.2.3.6 `totalSize ()`

```
public long totalSize()
```

This method determines the total size of the file system on which the connection's target resides.



6.4.2.3.7 `availableSize ()`

```
public long availableSize()
```

This method determines the free memory that is available on the file system on which the file or directory resides.

6.4.2.3.8 `usedSize ()`

```
public long usedSize()
```

This method determines the used memory of a file system on which the connection's target resides.

6.4.2.3.9 `directorySize (boolean includeSubDirs)`

```
public long directorySize(boolean includeSubDirs) throws  
java.io.IOException
```

This method determines the size in bytes on a file system of all of the files that are contained in a directory.

6.4.2.3.10 `fileSize ()`

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

This method determines the size of a file.

Tip: only used for file size; if used for directory size, an exception will be thrown out.

6.4.2.3.11 `canRead ()`

```
public boolean canRead()
```

This method checks if the file or directory is readable.

Tip: This method is not supported and returns true on iDEN handsets.

6.4.2.3.12 `canWrite ()`

```
public boolean canWrite()
```

This method checks if the file or directory is writeable.

Tip: This method is not supported and returns true on iDEN handsets.

6.4.2.3.13 `isHidden ()`

```
public boolean isHidden()
```

This method checks if the file or directory is hidden.

Tip: This method is not supported and returns false on iDEN handsets.

6.4.2.3.14 `setReadable(boolean readable)`

```
public void setReadable(boolean readable) throws java.io.IOException
```

This method sets the file or directory readable attribute to the indicated value

Tip: This method is not supported and does nothing on iDEN handsets.



6.4.2.3.15 setWritable(boolean writable)

public void **setWritable**(boolean writable) throws java.io.IOException

This method sets the file or directory writable attribute to the indicated value.

Tip: This method is not supported and does nothing on the iDEN phone.

6.4.2.3.16 setHidden(boolean hidden)

public void **setHidden**(boolean hidden) throws java.io.IOException

This method sets the file or directory hidden attribute to the indicated value.

Tip: This method is not supported and does nothing on iDEN handsets.

6.4.2.3.17 list()

public java.util.Enumeration **list**() throws java.io.IOException

This method gets a list of all files and directories contained in a directory.

6.4.2.3.18 list(java.lang.String filter, boolean includeHidden)

public java.util.Enumeration **list**(java.lang.String filter,
boolean includeHidden) throws java.io.IOException

This method gets a list of all files and directories contained in a directory.

6.4.2.3.19 create

public void **create**() throws java.io.IOException

This method creates a file corresponding to the file string provided in the `Connector.open()` method for this `FileConnection`.

Tip: This method is only used to create a file; if a new directory is needed, use `mkdir()`.

6.4.2.3.20 mkdir()

public void **mkdir**() throws java.io.IOException

This method creates a directory corresponding to the directory string provided in the `Connector.open()` method.

6.4.2.3.21 exists()

public boolean **exists**()

This method checks if the file or directory specified in the URL passed to the `Connector.open()` method exists.

6.4.2.3.22 isDirectory()

public boolean **isDirectory**()

This method checks if the URL passed to the `Connector.open()` is a directory.



6.4.2.3.23 delete ()

```
public void delete() throws java.io.IOException
```

This method deletes the file or directory specified in the `Connector.open()` URL.

6.4.2.3.24 rename ()

```
public void rename(java.lang.String newName) throws  
java.io.IOException
```

This method renames the selected file or directory to a new name in the same directory.

6.4.2.3.25 truncate ()

```
public void truncate(int byteOffset) throws java.io.IOException
```

This method truncates the file, discarding all data from the given byte offset to the current end of the file.

6.4.2.3.26 setFileConnection (java.lang.String fileName)

```
public void setFileConnection(java.lang.String fileName) throws  
java.io.IOException
```

This method resets this `FileConnection` object to another file or directory.

6.4.2.3.27 getName ()

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

This method returns the name of a file or directory excluding the URL schema and all paths.

6.4.2.3.28 getPath ()

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

This method returns the path excluding the file or directory name and the "file" URL schema and host from where the file or directory specified in the `Connector.open()` method is opened.

6.4.2.3.29 getURL ()

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

This method returns the full file URL including the scheme, host, and path from where the file or directory specified in the `Connector.open()` method is opened

6.4.2.3.30 lastModified ()

```
public long lastModified()
```

This method returns the time that the file denoted by the URL specified in the `Connector.open()` method was last modified.

Tip: This method is not supported and returns 0 on the IDEN phone.



6.4.2.3.31 isOpen ()

```
public Boolean isOpen()
```

This method returns an indication of whether the file connection is currently open or not.

6.4.2.4 Class FileSystemListener

6.4.2.4.1 rootChanged ()

```
public void rootChanged(int state, java.lang.String rootName)
```

This method is invoked when a root on the device has changed state.

Tip: Only one root “/” is supported on the IDEN phone.

6.4.2.5 Compatibility

6.4.2.5.1 getProperty ()

When System.getProperty() is called with key “microedition.io.file.FileConnection.version” the version string “1.0” shall be returned if FCOP is supported on the handset.

6.4.2.6 File Connection Code Example

```
import javax.microedition.midlet.*;
import javax.microedition.io.*;
import javax.microedition.io.file.*;
import java.io.*;
import java.util.Enumeration;

public class DemoFC extends MIDlet
{
    protected void startApp() throws MIDletStateChangeException
    {
        version();
        create();
        write_read();
        attrdemo();
        renamedemo();
        getdemo();
        setFC();
        truncate_size_demo();
    }
}
```



```
        list();
        notifyDestroyed();
    }
    private void version()
    {

        String v = System.getProperty(
"microedition.io.file.FileConnection.version");
        if( v !=null)
        {
            System.out.println("FCOP is available "+v);
        }
        else
        {
            System.out.println("FCOP is not available");
        }
        Enumeration roots = FileSystemRegistry.listRoots();
        for (; roots.hasMoreElements() ;)
        {
            System.out.println("Roots:" + roots.nextElement());
        }
        FileSystemListener File_listen = (FileSystemListener) new
FileListener();
        System.out.println("add one FileSystemListener:
"+FileSystemRegistry.addFileSystemListener(File_listen));
        System.out.println("add one FileSystemListener:
"+FileSystemRegistry.removeFileSystemListener(File_listen));
    }

    private void create()
    {
        try {
            FileConnection fconn =
(FileConnection)Connector.open("file:///./a1b1c2file");
            if(!fconn.exists())
            {
                fconn.create();
            }
        }
    }
}
```



```
    }
    fconn.close();
    fconn =
(FileConnection)Connector.open("file:///./a1b1c2dir");
    if(!fconn.exists())
    {
        fconn.mkdir();
    }
    fconn.close();
} catch (Exception ioe) {}
}

private void list()
{
    try {
        FileConnection fconn =
(FileConnection)Connector.open("file:///./");
        Enumeration a = fconn.list();
        for(int i=0;a.hasMoreElements();i++)
        {
            System.out.println(a.nextElement());
        }
        System.out.println("list(d*,false)");
        a = fconn.list("d*",false);
        for(int i=0;a.hasMoreElements();i++)
        {
            System.out.println(a.nextElement());
        }
        System.out.println("list(*,true)");
        a = fconn.list("*",true);
        for(int i=0;a.hasMoreElements();i++) {
            System.out.println(a.nextElement());
        }
    } catch (IOException ioe) {
        System.err.println(ioe.toString());
    }
}
```




```
}

private void write_read()
{
    try {
        FileConnection fconn =
        (FileConnection)Connector.open("file:///./f_abcde");

        if(!fconn.exists())
        {
            fconn.create();
        }
        OutputStream os = fconn.openOutputStream();
        os.write(11);
        os.write(22);
        os.write(33);
        os.close();
        InputStream is = fconn.openInputStream();
        System.out.println("OUT:" + is.read());
        System.out.println("OUT:" + is.read());
        System.out.println("OUT:" + is.read());
        is.close();
        fconn.close();
        fconn =
        (FileConnection)Connector.open("file:///./f_abcde");
        if(!fconn.exists())
        {
            fconn.create();
        }
        DataOutputStream os_data = fconn.openDataOutputStream();
        os_data.write(77);
        os_data.write(99);
        os_data.write(66);
        os_data.close();
        DataInputStream is_data = fconn.openDataInputStream();
        System.out.println("OUT:" + is_data.read());
    }
}
```



```
        System.out.println("OUT:" + is_data.read());
        System.out.println("OUT:" + is_data.read());
        is_data.close();
        fconn.close();
    } catch (IOException ioe) {
        System.err.println(ioe.toString());
    }
}

private void attrdemo() {
    try {
        FileConnection fconn =
        (FileConnection)Connector.open("file:///./dir_1");
        if(!fconn.exists())
        {
            fconn.mkdir();
        }
        System.out.println("dir_1 exist:"+fconn.exists());
        System.out.println("dir_1
isDirectory:"+fconn.isDirectory());
        System.out.println("dir_1 isHidden:"+fconn.isHidden());
        System.out.println("dir_1 canWrite:"+fconn.canWrite());
        System.out.println("dir_1 canRead:"+fconn.canRead());

        fconn.setHidden(true);
        System.out.println("dir_1
isHidden(true):"+fconn.isHidden());
        fconn.setHidden(false);
        System.out.println("dir_1
isHidden(false):"+fconn.isHidden());
        fconn.setWritable(false);
        System.out.println("dir_1
canWrite(false):"+fconn.canWrite());
        fconn.setWritable(true);
        System.out.println("dir_1
canWrite(true):"+fconn.canWrite());
        fconn.setReadable(false);
```



```
        System.out.println("dir_1
canWrite(false):"+fconn.canRead());

        fconn.setReadable(true);

        System.out.println("dir_1
canWrite(true):"+fconn.canRead());

        fconn.close();

    } catch (IOException ioe) {
        System.out.println("IOException"+ioe);
    }
}

private void renamedemo()
{
    try {
        FileConnection fconn =
(FileConnection)Connector.open("file:///./rename");
        if(!fconn.exists()) {
            fconn.mkdir();
        }
        fconn.close();

        fconn =
(FileConnection)Connector.open("file:///./rename/f1");
        if(!fconn.exists())
        {
            fconn.create();
            OutputStream os = fconn.openOutputStream();
            os.write(11);
            os.write(22);
            os.write(33);
            os.close();

        }
        fconn.close();
    }
```



```
fconn =
(FileConnection)Connector.open("file:///./rename/f2");
if(fconn.exists())
{
    fconn.delete();
}
fconn.close();

fconn =
(FileConnection)Connector.open("file:///./rename/f1");
fconn.rename("f2");
fconn.close();

fconn =
(FileConnection)Connector.open("file:///./rename/dir1");
if(!fconn.exists())
{
    fconn.mkdir();
}
fconn.close();

fconn =
(FileConnection)Connector.open("file:///./rename/dir2");
if(fconn.exists()) {
    System.out.println("dir2 exists and delete");
    fconn.delete();
}
fconn.close();

fconn =
(FileConnection)Connector.open("file:///./rename/dir1");
fconn.rename("dir2");
fconn.close();
System.out.println("show rename result");

fconn =
(FileConnection)Connector.open("file:///./rename");
Enumeration a = fconn.list();
for(int i=0;a.hasMoreElements();i++) {
```



```
        System.out.println(a.nextElement());
    }

    fconn.close();

} catch (IOException ioe)
{
    System.out.println(ioe);
}

private void getdemo() {
    try {
        FileConnection fconn =
        (FileConnection)Connector.open("file:///./getxxx");
        if(!fconn.exists())
        {
            fconn.mkdir();
        }

        System.out.println("getxxx/getURL:"+fconn.getURL());
        System.out.println("getxxx/getName:"+fconn.getName());
        System.out.println("getxxx/getPath:"+fconn.getPath());
        fconn.close();

        fconn =
        (FileConnection)Connector.open("file:///./getxxxfile");
        if(!fconn.exists())
        {
            fconn.create();
        }

        System.out.println("getxxxfile/getURL:"+fconn.getURL());

        System.out.println("getxxxfile/getName:"+fconn.getName());
```



```
System.out.println("getxxxfile/getPath:"+fconn.getPath());
    fconn.close();

    } catch (IOException ioe) {
        System.out.println(ioe);
    }
}

private void setFC() {
    try {
        FileConnection fconn =
(FileConnection)Connector.open("file:///./BB");
        if(!fconn.exists())
        {
            fconn.mkdir();
        }
        fconn.close();

        fconn = (FileConnection)Connector.open("file:///./BB");
        fconn.setFileConnection("..");
        System.out.println("file:///./ URL:" + fconn.getURL());
        System.out.println("file:///./ Name:" + fconn.getName());
        System.out.println("file:///./ Path:" + fconn.getPath());
        fconn.close();
    }
    catch (Exception ioe) {
        System.err.println(ioe.toString());
    }
}

private void truncate_size_demo()
{
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    DataOutputStream outputStream = new DataOutputStream(baos);
```



```
byte[] ba ;
OutputStream os;
FileConnection fconn ;

try {

    fconn =
(FileConnection)Connector.open("file:///./file_to_truncate");

    if(!fconn.exists())
    {
        fconn.create();
    }

    os = fconn.openOutputStream();
    long a_length = 1000;

    for(long i=0; i<a_length;i++)
    {
        try
        {
            outputStream.writeByte(0);
        } catch (IOException e) {}
    }
    ba = baos.toByteArray();
    os.write(ba, 0, ba.length);
    os.close();
    fconn.close();

    fconn =
(FileConnection)Connector.open("file:///./file_to_truncate");
    System.out.println("Total size before
truncate:"+fconn.totalSize());

    System.out.println("Avail size before
truncate:"+fconn.availableSize());

    System.out.println("Used size before
truncate:"+fconn.usedSize());
```



```

        System.out.println("File size before
truncate:"+fconn.fileSize());

        fconn.close();

        fconn =
(FileConnection)Connector.open("file:///./file_to_truncate");

        fconn.truncate(900);

        System.out.println("truncate "+ 900+ "size for
file_to_truncate");

        System.out.println("Total size after
truncate:"+fconn.totalSize());

        System.out.println("Avail size after
truncate:"+fconn.availableSize());

        System.out.println("Used size after
truncate:"+fconn.usedSize());

        System.out.println("File size after
truncate:"+fconn.fileSize());

        fconn.delete();

        fconn.close();

        fconn = (FileConnection)Connector.open("file:///./");

        System.out.println("root directory size
includeSubDirs:"+fconn.directorySize(true));

        System.out.println("root directory size not
includeSubDirs:"+fconn.directorySize(false));

        fconn.close();

    } catch (IOException ioe) {}

}

/**
 * Pause the MIDlet
 */
protected void pauseApp() {
}

/**
 * Called by the framework before the application is unloaded
 */

```




```
protected void destroyApp(boolean unconditional)
    throws MIDletStateChangeException {
}

public class FileListener implements FileSystemListener
{
    public FileListener()
    {

    }

    public void rootChanged(int state, java.lang.String rootName)
    {

    }

}
}
```



6.5 Java ZIP

6.5.1 Overview

The Java Zip API has been included as an enhancement especially well suited for a limited bandwidth data device such as an iDEN phone. It allows for file deflation before sending data via the network and inflation after receiving data from the network to best use the bandwidth available. Downloading a zipped file and then decompressing on the device is usually much faster than downloading uncompressed content.

The Java ZIP API consists of ZipEntry, ZipInputStream, ZipOutputStream, and ZipException. This package provides classes for reading and writing data in the standard ZIP (WinZip archive) format.

The Java ZIP API is compatible with the Sun's J2SE™ v1.4 ZIP API (java.util.zip). Please refer to the following web page for details: <http://java.sun.com/j2se/1.4.2/docs/api/index.html>.

6.5.2 Class Description

The API for the ZIP is located in the com.mot.iden.zip package.

```
java.lang.Object
|
+ - com.mot.iden.zip.ZipEntry
```

6.5.3 Method Descriptions

Please refer to the relevant Javadocs (java.util.zip).

6.5.4 Code Example

6.5.4.1 Get ZipEntry information

```
public static void print(ZipEntry e)
{
    PrintStream err = System.err;
    err.print("added " + e.getName());
    if ( e.getMethod() == ZipEntry.DEFLATED ) {
        long size = e.getSize();
        if (size > 0) {
            long csize = e.getCompressedSize();
            long ratio = ((size-csize)*100) / size;
            err.println(" (deflated " + ratio + "%)");
        } else {
            err.println(" (deflated 0%)");
        }
    } else {
        err.println(" (stored 0%)");
    }
}
```



6.5.4.2 ZipOutputStream/ZipInputStream

```
try {
    ByteArrayOutputStream gis = new ByteArrayOutputStream(1024);

    // (1) Compression: Define ZIPOutputStream with
    //      ByteArrayOutputStream
    ZipOutputStream os = new ZipOutputStream(gis);

    ZipEntry zipentry = new ZipEntry("TEST1");
    /* set the 1st entry name */
    os.putNextEntry(zipentry);

    // (2) Writes the string to the ZIPOutputStream
    os.write("This chapter covers how to configure a system "+
        "without a name service. Administration is ...".getBytes());

    zipentry = new ZipEntry("TEST2");
    /* set the 2nd entry name */
    os.putNextEntry(zipentry);

    // (3) Writes the string to the ZIPOutputStream
    os.write("The document you requested is not found. " +
        "It may have expired or moved.".getBytes());

    os.close();

    // (4) Decompression: Get input compressed data
    //      from the gis stream
    ByteArrayInputStream gis1 =
        new ByteArrayInputStream(gis.toByteArray());

    // (5) Define ZIPInputStream with ByteArrayInputStream
    ZipInputStream os1 = new ZipInputStream(gis1);

    byte[] buf1 = new byte[2048]; /* Decompressed buffer */
    int ch;
    ZipEntry entry;

    // (6) Reads the compressed stream to the decompressed buffer
    while ((entry = os1.getNextEntry()) != null) {
        System.out.println("Extracting: " + entry);

        while ((ch = os1.read(buf1, 0, buf1.length - 1)) >= 0) {
            System.out.println(new String(buf1, 0, ch));
        }
    }

    os1.close();
} catch (Exception e) {
    e.printStackTrace();
}
```



7

Networking and Security

7.1 Overview

This section will present an in-depth explanation with examples of the following networking and security features:

- J2ME™ Networking
- Push registry
- Wireless Messaging
- WMA over MMS
- MIDP 2.0 Security
- Cryptography
- JAXP
- JAX-RPC

7.2 J2ME™ Networking

7.2.1 Overview

iDEN handsets provide the following protocols specified in MIDP 2.0:

- HTTP
- HTTPS
- TCP Sockets
- SSL Secure Sockets
- Server Sockets
- UDP Sockets
- Serial Port Access



7.2.2 Timeouts




The timeout period for the TCP implementation is 40 seconds for an open operation. The timeout period for read/write operations is about 120 seconds if the timeout flag is set to true, and about 180 seconds if the timeout flag is set to false. The lingering time for closing sockets is 10 seconds, so if a new socket is requested within this time frame and the maximum number of sockets opened has been reached, an `IOException` is thrown.

Applications requesting a network resource for any protocol must use one of these three methods:

```
Connector.open(String URL) - default READ_WRITE, no timeout
Connector.open(String URL, int mode) - defaults to no timeout
Connector.open(String URL, int mode, Boolean timeout)
```

The URL is the distinguishing argument that determines the difference between HTTP, UDP, Serial, and so on. The following chart details the prefixes that should be used for the supported protocols.

Supported Networking Protocols

Protocol	URL Format
HTTP	http://
HTTPS	https://
TCP Sockets	socket://host:port
SSL Secure Sockets	ssl://host:port or ssocket://host:port
Server Sockets	socket://:port or serversocket://:port
UDP Sockets	datagram://
Serial Port	comm:com0 or comm:0;
 RFCOMM	btsp://host or btsp://localhost:UUID
 L2CAP	bt12cap://host or bt12cap://localhost:UUID
 OBEX	btgoep://host:UUID or tcpobex://localhost:port or tcpobex://localhost or tcpobex://

7.2.3 Protocols

7.2.3.1 HTTP

The HTTP implementation follows the MIDP 2.0 standard. The `Connector.open()` methods return an `HttpConnection` object that is then used to open streams for reading and writing. The following is a code example:

```
HttpConnection hc =
    (HttpConnection)Connector.open("http://www.motorola.com");
```



In this particular example, the standard port 80 is used, but you can specify this parameter as in the following example:

```
HttpConnection hc =  
(HttpConnection)Connector.open("http://www.motorola.com:8080");
```

The other static Connector methods work in the same manner, but they provide the application additional control in dealing with the properties of the connection. By default, HTTP 1.1 persistency is used to increase efficiency while requesting multiple pieces of data from the same server. In order to disable persistency, set the "Connection" property of the HTTP header to "close".

7.2.3.2 HTTPS

The HTTPS implementation follows the MIDP 2.0 standard, except for the security aspects. The `Connector.open()` methods return an `HttpsConnection` object that is then used to open streams for reading and writing. The following is a code example:

```
HttpsConnection hc =  
(HttpsConnection)Connector.open("https://www.motorola.com");
```

In this particular example, the standard port 443 is used, but you can specify this parameter as in the following example:

```
HttpsConnection hc =  
(HttpsConnection)Connector.open("http://www.motorola.com:8888");
```

The other static Connector methods work in the same manner, but they provide the application additional control in dealing with the properties of the connection.

Note that only VeriSign security certificates are supported. The following is a list of supported features:

- SSL 3.0
- TLS 1.0
- Server Authentication

7.2.3.3 TCP Sockets

The low-level socket used to implement the higher-level HTTP protocol is exposed to applications via the Generic Connection Framework. The use is similar to the examples above; however, a `SocketConnection` is returned by the `Connection.open()` method, as in the following example:

```
SocketConnection sc =  
(SocketConnection)Connector.open("socket://www.motorola.com:8000");
```

Although similar to HTTP, notice the required port number at the end of the remote address. In the previous protocols, those ports are well known and registered so they are not required, but in the case of low level sockets, this value is not defined. The port number is a required parameter for this protocol stack.



7.2.3.4 SSL Secure Sockets

The low-level socket used to implement the higher-level HTTPS protocol is also exposed to applications via the Generic Connection Framework. The usage is similar to the examples above:

```
SecureSocketConnection sc =  
(SecureSocketConnection)Connector.open("ssl://www.motorola.com:8000")  
;
```

As with non-secure sockets, the port number is a required parameter for this protocol stack.

7.2.3.5 Server Sockets

In addition to acting as a data requestor, some applications may act as data providers or servers. In order to accomplish this without workarounds or polling, a server socket is required. This functionality is provided via the Generic Connection Framework. Opening a `ServerSocket` with the `Connector` object returns a `ServerSocketConnection`. Unlike the other networking protocols, the `ServerSocketConnection` does not contain any accessor methods to retrieve data, but rather only one method to accept and open a `SocketConnection`. This method blocks until a `Socket` connection is available, at which time it returns a `ServerSocketConnection` object. The following example illustrates this:

```
ServerSocketConnection scn =  
(ServerSocketConnection)Connector.open("socket://:8000");  
  
ServerSocketConnection sc =  
(ServerSocketConnection)scn.acceptAndOpen();
```

The URL parameter passed in is similar to that used for TCP sockets, with the exception of the target address. In this particular instance, the target address is left blank, assuming the socket is to be opened on the local device. The port number however, is still required. The `acceptAndOpen()` method of the `ServerSocketConnection` object is a blocking call, so applications that utilize the particular protocol, should take this into consideration.

Note that to close the socket, you must close the associated `ServerSocketConnection`.

7.2.3.6 UDP Sockets

If networking efficiency is of greater importance than reliability, datagram (UDP) sockets are also available to the application in much the same manner as other networking protocols. The `Connector` object in this case returns an `UDPDatagramConnection` object, as is shown in the following example:

```
UDPDatagramConnection dc =  
(UDPDatagramConnection)Connector.open(  
"datagram://70.69.168.167:8000");
```

Much like low-level sockets, accessing UDP requires both a target address and a port number. iDEN handsets support a maximum outgoing and incoming payload of 1472 bytes and 2944 bytes, respectively.



7.2.3.7 Serial Port Access

Applications using the bottom connector (serial port) to communicate with a variety of devices are given exclusive access to the port until either the application voluntarily releases the port or the application is terminated. Much like any other networking connection, opening a serial port is not guaranteed and an exception may be thrown. If another application—native or Java—is using the port, or a cable is not attached to the device, an `IOException` is thrown. In the normal usage scenario, the `Connector` object in this instance returns a `CommConnection`, as is shown in the following example:

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

```
CommConnection cc = (CommConnection)
Connector.open("comm:com0;baudrate=19200");
```

Both old connection optional parameters from iDEN OEM Connection implementation and new connection parameters from MIDP 2.0 are allowed. The new optional parameters are recommended, as these are specified in MIDP 2.0.

These are the old parameters:

Old Connection Optional Parameters

Parameter	Syntax	Options	Default
baud rate	baud rate = x	[300,1200,2400,4800,9600,19200,38400,57600,115200]	19200
Data bits	data bits = x	[8,7]	8
Stop bits	stop bits = x	1	1
parity with mapping	parity = x	[n,o,e,s,m] n=none, o = odd, e=even, s=space, m=mark	n
Flow control	flowcontrol = outflow/inflow	[n, s, h] / [n, s, h] n=none, s=software, h=hardware	N/n

Note - The following combinations of properties are not supported.

- 7 databits with none parity
- 8 databits with mark parity
- 8 databits with space parity
- 8 databits with odd parity
- 8 databits with even parity.

`IOException` will be thrown while trying to use any of the unsupported combinations in `Connector.open()`.



And here are the new parameters:

New Connection Optional Parameters

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



7.2.3.8 RFCOMM, L2CAP, and OBEX

These connections take advantage of communications using Bluetooth technology. iDEN handsets provide these connections only when equipped with Bluetooth hardware. For more information on these connection types and the Bluetooth APIs provided see section 7.10, Java™ APIs for Bluetooth™ Wireless Technology.

7.2.4 Implementation Notes I

As stated in the previous sections, a vast array of networking options is supported. The networking options, however, are limited by both memory and bandwidth, which place hard restrictions on the applications. These limitations manifest themselves mainly in the number of simultaneous connections that can be opened. Boundary conditions for each networking stack can be found in Appendix A:

Specification Sheets

page 525.

7.2.5 Tips /

- Keep in mind the blocking nature of many `javax.microedition.io` and `java.io` object methods. It's recommended to spawn another thread specifically dedicated to retrieving data in order to keep the user interface interactive. If a single thread is used to retrieve data on a blocking call, the user interface becomes inactive with the end-user perceiving the application as dead.
- When the length of the data is known, reading from an `InputStream` using an array is faster than reading byte by byte. For example, if the content length is provided in the header of the `HttpConnection`, then an array of the specified size can be used to read the data.
- The `InputStream` and `OutputStream` as well as the `Connection` object need to be completely closed.
- An application in the suspended state can still continue to actively use the networking facilities of the handset.
- The platform does not support simultaneous voice and data transmissions.
- Only one serial port is available. If you try to open two concurrent serial port connections, an exception is thrown.



7.3 Wireless Messaging

7.3.1 Overview



The Wireless Messaging API allows a MIDlet to open a connection and send or receive messages through this connection. It supports both SMS and datagram as the underlying protocol.

Developers SHOULD read the JSR before reading this guide.

Not all classes and methods are addressed in this developer guide. For those classes and methods, please refer to JSR 205 document found at <http://www.jcp.org>.

7.3.2 Method Descriptions

7.3.2.1 Connector Method

7.3.2.1.1 open

Returns a connection to the specified phone number or IP address.

```
public static Connection open(String name) throws IOException
```

Equals to `Connector.open(name, READ_WRITE)`.

If you're using the iDEN SMS protocol, `name` should contain "1" plus the phone number or "+1" plus the phone number. For example, either

```
Connector.open("sms://19545555555:5000") or
```

```
Connector.open("sms://+19545555555:5000")
```

 opens a connection to the device with the phone number 954-555-5555 at SMS port 5000.

If you're using the iDEN datagram protocol, `name` should contain the IP address. For example, `Connector.open("udp://120.2.12.20:5000")` opens a connection to the device at the IP address 120.2.12.20 at port 500.



7.3.2.2 MessageAddress Method

7.3.2.2.1 MessageAddress

Creates a new message address.

```
public MessageAddress(String address)
throws IllegalArgumentException
```

`address` consists of two parts; the host and the port fields, separated by a colon. The host field must contain either 0–9 digits or '+' followed by 0–9 digits. The port field must be a number between 0 and the GSM maximum port number 65,535. If `address` doesn't follow this rule, this method throws an `IllegalArgumentException`.

For a `Message` that has been sent using the Datagram as the underlying protocol, this method will return the IP address associated with this message rather than a phone number like in WMA over SMS.

7.3.2.2.3 Protocol Methods

7.3.2.2.4 newMessage

Creates a new text or binary message object.

```
public Message newMessage(String type)
```

`type` must be either `MessageConnection.BINARY_MESSAGE` or `MessageConnection.TEXT_MESSAGE`. `type` should be the default address from the original open.

7.3.2.2.5 send

Sends a `Message` over the connection.

```
public void send(Message msg) throws IOException
```

Before using this method, be sure the connection is still open. Otherwise, this message throws an `IOException`. The message and host must not be nulls.

7.3.2.2.6 receive

Returns a `Message`, created from the bytes sent over the connection.

```
public Message receive() throws IOException
```

Before using this method, be sure the connection is still open. Otherwise, this method throws an `IOException`. You cannot receive a message on a connection that was opened in client mode.



7.3.2.2.7 numberOfSegments

Returns how many segments would be required to send the specified message.

```
public int numberOfSegments(Message msg)
```

If `msg` is null or is not a Message object, this method returns 0.

Neither of the two protocols that this phone supports (WMA over SMS and WMA over Datagram) let you send messages that require more than one segment. Use this method before sending a message to make sure it fits in one segment.

7.3.3 Caveat for WMA over SMS

iDEN's WMA implementation supports both SMS send and receive. It does not support CBS.

iDEN's SMS implementation partly follows GSM 03.40, so iDEN's WMA over SMS implementation has the following restrictions:

- It supports only 8-bit encoding scheme.
- The maximum size of one message is 140 bytes without a destination port, and 132 bytes with a destination port.
- It does not support concatenation.
- If you do not specify a port number when sending a message, the message is routed to the destination unit's native SMS application.

The port number can be any number from 0 to 16999, except for restricted numbers specified in Appendix A of WMA 1.1 Specification.

This API is intended for sending SMS within the Nextel network only due to the iDEN's Caveat for WMA over Datagram.

iDEN's WMA implementation supports send and receive over UDP datagram with the following restrictions:

- It supports only 8-bit encoding scheme.
- The maximum length is 1467 bytes for a text message and 1465 bytes for a binary message.
- It does not support concatenation.



7.4 WMA over MMS



This API is only available on these handsets.

The Wireless Messaging API over MMS allows a MIDlet to open a connection and send or receive messages with multimedia contents through this connection.

Not all classes and methods are addressed in this developer guide. For those classes and methods, please refer to JSR 205 document.

7.4.1 Package Description

APIs for WMA over MMS are all located in package `javax.microedition.io` and `javax.wireless.messaging`.

7.4.2 Package Tree

7.4.2.1 Class Hierarchy

The following will be the Class Hierarchy for the WMA over MMS API:

```

java.lang.Object
|
+--javax.microedition.io.Connector
|
+-- javax.wireless.messaging.MessagePart
|
+--java.lang.Throwable
|
+--java.lang.Exception
|
+--java.io.IOException
|
+-- javax.wireless.messaging.SizeExceededException

```



7.4.2.2 Interface Hierarchy

The following will be the Interface Hierarchy for the WMA over MMS API:

```
interface javax.wireless.messaging.Message
interface javax.wireless.messaging.BinaryMessage
interface javax.wireless.messaging.MultipartMessage
interface javax.wireless.messaging.TextMessage
interface javax.wireless.messaging.MessageConnection
interface javax.wireless.messaging.MessageListener
```

7.4.2.3 javax.microedition.io.Connector

7.4.2.3.1 open(String name)

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

Creates and opens a Connection.

Tips

- If you are using iDEN MMS protocol, name must start with "mms://", and contain an email address, phone number, or IP address. Shortcode addressing is not supported on iDEN handsets.
- A MIDlet can open no more than 7 concurrent MMS connections.
- The application id "com.mot.cldc.io.j2me.mms" is reserved. An application can't open a connection in server mode using this application ID. Any messages sent to an address with this application ID will be discarded.

7.4.2.3.2 open(String name, int mode)

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

```
throws IOException
```

Create and open a Connection with the specified access mode.

7.4.2.3.3 open(String name, int mode, boolean timeouts)

```
public static javax.microedition.io.Connection open(String name,
int mode,
```

```
boolean timeouts) throws IOException
```

Create and open a Connection with the specified access mode. The Connection is created with timeout exceptions if specified.



7.4.2.3.4 openDataInputStream(String name)

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

throws IOException

Create and open a connection input stream.

Tip: This function is not supported on MMS connections. When the name starts with "mms://" an IllegalArgumentException will be thrown.

7.4.2.3.5 openDataOutputStream(String name)

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

throws IOException

Creates and opens a connection output stream.

Tip: This function is not supported on MMS connections. When the name starts with "mms://" an IllegalArgumentException will be thrown.

7.4.2.3.6 openInputStream(String name)

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

throws IOException

Create and open a connection input stream.

Tip: This function is not supported on MMS connections. When the name starts with "mms://" an IllegalArgumentException will be thrown.

7.4.2.3.7 openOutputStream (String name)

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

throws IOException

Create and open a connection output stream.

Tip: This function is not supported on MMS connections. When the name starts with "mms://" an IllegalArgumentException will be thrown.

7.4.2.4 javax.wireless.messaging.TextMessage

7.4.2.4.1 setPayloadText(String data)

```
public void setPayloadText (String data)
```

Sets the payload data of this message.

Tip: The length of the payload data should not exceed 30K.



7.4.2.5 javax.wireless.messaging.MessageConnection

7.4.2.5.1 numberOfSegments(Message msg)

```
public int numberOfSegments (Message msg)
```

Returns the number of segments in the underlying protocol that would be needed for sending the specified Message.

Tip: If the length of a message exceeds 30K, this function will return 0, otherwise return 1.

7.4.2.5.2 newMessage(String type)

```
public javax.wireless.messaging.Message newMessage(String type)
```

Constructs a new Message object of a given type.

Tip: The type parameter should not be set to BINARY_MESSAGE. MMS servers for iDEN handsets do not support mime type for binary messages and will reject binary messages.

7.4.2.5.2 newMessage(String type, String address)

```
public javax.wireless.messaging.Message newMessage(String type,  
String address)
```

Constructs a new Message object of a given type and initializes it with the given destination address. The semantics related to the parameter type are the same as for the method signature with just the type parameter.

7.4.2.6 javax.wireless.messaging.MessagePart

7.4.2.6.1 MessagePart(byte[] contents, String mimeType, String contentId, String contentLocation, String enc)

```
public MessagePart(byte[] contents, String mimeType, String  
contentId, String contentLocation, String enc) throws  
SizeExceededException
```

Constructor of class MessagePart.

Tip: "text/plain" is often used for text content; "application/smil" is used for smil content.

An `IllegalArgumentException` should be thrown for unsupported mime types. The following table lists the supported mime types:



text/*	text/html	text/plain	text/x-hdml	text/x-ttml
text/x-vCalendar	text/x-vCard	text/vnd.wap.wml	text/vnd.wap.wmlscript	text/vnd.wap.wta-event
multipart/*	multipart/mixed	multipart/form-data	multipart/byteranges	multipart/alternative
application/*	application/java-vm	application/x-www-form-urlencoded	application/x-hdmlc	application/vnd.wap.wmlc
application/vnd.wap.wmlscriptc	application/vnd.wap.wta-eventc	application/vnd.wap.uaprof	application/vnd.wap.wtls-ca-certificate	application/vnd.wap.wtls-user-certificate
application/x-x509-ca-cert	application/x-x509-user-cert	image/*	image/gif	image/jpeg
image/tiff	image/png	image/vnd.wap.wbmp	application/vnd.wap.multipart.*	application/vnd.wap.multipart.mixed
application/vnd.wap.multipart.form-data	application/vnd.wap.multipart.byteranges	application/vnd.wap.multipart.alternative	application/xml	text/xml
application/vnd.wap.wbxm1	application/x-x968-cross-cert	application/x-x968-ca-cert	application/x-x968-user-cert	text/vnd.wap.si
application/vnd.wap.sic	text/vnd.wap.sl	application/vnd.wap.slc	text/vnd.wap.co	application/vnd.wap.coc
application/vnd.wap.multipart.related	application/vnd.wap.sia	text/vnd.wap.connectivity-xml	application/vnd.wap.connectivity-wbxm1	application/pkcs7-mime
application/vnd.wap.hashcd-certificate	application/vnd.wap.signed-certificate	application/vnd.wap.cert-response	application/xhtml+xml	application/wml+xml
text/css	application/vnd.wap.mms-message	application/vnd.wap.rollover-certificate	multipart/related	application/vnd.wap.wml+xml
text/x-wap-wta-wml	application/x-wap-wta-wmlc	text/vnd.wap.channel	application/smil	application/x-shockwave-flash
image/bmp	image/pjpeg	image/svg+xml	audio/amr	audio/GSM-EFR
audio/imelody	audio/mpeg	audio/midi	audio/mid	audio/pcma
audio/pcmu	audio/x-ms-wma	audio/x-wav	audio/sp-midi	audio/x-idenambe
video/mp4	video/h263	video/x-ms-wmv		

7.4.2.6.2 MessagePart(InputStream is, String mimeType, String contentId, String contentLocation, String enc)

```
public MessagePart(InputStream is, String mimeType, String
contentId, String contentLocation, String enc) throws
IOException, SizeExceededException
```

Constructs a `MessagePart` object from an `InputStream`. The contents of the `MessagePart` are loaded from the `InputStream` during the constructor call until the end of the stream is reached.



7.4.2.6.3 MessagePart(byte[] contents, int offset, int length, String mimeType, String contentId, String contentLocation, String enc)

```
public MessagePart(byte[] contents, int offset, int length, String
mimeType, String contentId, String contentLocation, String enc)
throws SizeExceededException
```

Constructs a `MessagePart` object from a subset of the byte array. This constructor is only useful, if the data size is small (roughly less than 10K). For larger content the `InputStream` based constructor should be used.

7.4.2.7 javax.wireless.messaging.MultipartMessage

7.4.2.7.1 addAddress(String type, String address)

```
public boolean addAddress(String type, String address)
```

Adds an address to the multipart message.

Tips:

- Shortcode address is not supported on iDEN handsets.
- The length of this address must be less than 256.
- iDEN handsets' MMS implementation does not support BCC addresses. Although an application can add BCC addresses, the message will not be sent to the BCC address.
- All addresses of receivers including "to" and "cc" combined either contain 0 or 1 application ID. This application ID will be used for all addresses.
- The maximum number of total receivers specified by "to", "cc" and "bcc" combined is 20.

7.4.2.7.2 addMessagePart(MessagePart part)

```
public void addMessagePart(javax.wireless.messaging.MessagePart
part)
```

```
throws SizeExceededException
```

Attaches a `MessagePart` to the multipart message.

Tip: The total length of this message should not exceed 30K.



7.4.3 Code Examples

The following is a code example demonstrating the usage of MMS-based messaging:

```
/**
 * Demo program of Motorola iDEN WMA on MMS
 * Filename: wmaTest.java
 *
 * @version 1.0
 * @author Motorola, Inc.
 */

import java.io.*;
import javax.microedition.midlet.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.wireless.messaging.*;

public class wmaTest extends MIDlet implements CommandListener
{
    Display display = Display.getDisplay(this);
    Displayable resumeScreen = null;
    Form backForm = new Form("Test WMA over MMS");
    TextField destinationTextField = new TextField("To", null, 20,
    TextField.ANY);

    Command startCommand = new Command("Start", Command.OK, 1);
    Command exitCommand = new Command("Exit", Command.EXIT, 2);
    Command viewCommand = new Command("View", Command.OK, 3);

    MessageConnection messconn = null;
    MultipartMessage msg;

    public wmaTest()
    {

```



```
/* Add UI items to form */
backForm.append(destinationTextField);
backForm.addCommand(startCommand);
backForm.addCommand(exitCommand);
backForm.setCommandListener(this);
resumeScreen = backForm;
}

public void sendMMS() throws IOException
{
    MessagePart mp;
    String to = destinationTextField.getString();

    msg =
(MultipartMessage) (messconn.newMessage(MessageConnection.MULTIPART_ME
SSAGE, to));
    msg.setSubject("Test MMS from JAVA!");

    /* Add a text content to MMS */
    String s = new String("\nPls enjoy this MMS!");
    byte[] buf = s.getBytes("ISO-8859-1");
    mp = new MessagePart(buf, "text/plain", "txt1", null, "ISO-
8859-1");
    msg.addMessagePart(mp);

    /* Add a jpeg content to MMS */
    InputStream in = this.getClass().getResourceAsStream("/2.jpg");
    mp = new MessagePart(in, "image/jpeg", "jpeg2", null, null);
    msg.addMessagePart(mp);
    in.close();

    /* Add a jpeg content to MMS */
    in = this.getClass().getResourceAsStream("/4.jpg");
    mp = new MessagePart(in, "image/jpeg", "jpeg4", null, null);
    msg.addMessagePart(mp);
    in.close();
}
```



```
messconn.send(msg);
backForm.append("\nSending MMS SUCCESS!");
msg = null;
}

public void receiveMMS()throws IOException
{
    msg = (MultipartMessage)(messconn.receive());
    backForm.append("\nReceive MMS SUCCESS!");
    backForm.addCommand(viewCommand);
}

public void viewMsg() throws UnsupportedEncodingException
{
    MessagePart mp;
    MessagePart[] mpArray;
    byte[] content;
    Image imageItem;
    String mime_type, s;

    if (msg == null)
    {
        return;
    }

    mpArray = msg.getMessageParts();
    backForm.deleteAll();
    backForm.append("\nSubject : " + msg.getSubject());
    backForm.append("\nFrom : " + msg.getAddress());
    for (int k = 0; k < mpArray.length; k++)
    {
        mp = mpArray[k];
        mime_type = mp.getMIMEType();
        content = mp.getContent();
    }
}
```



```
        if (mime_type.equals("image/jpeg"))
        {
            imageItem = Image.createImage(content, 0,
mp.getLength());
            backForm.append(imageItem);
        }
        else if (mime_type.equals("text/plain"))
        {
            backForm.append("\n");
            if (mp.getEncoding() != null)
            {
                s = new String(content, mp.getEncoding());
            }
            else
            {
                s = new String(content, "UTF-8");
            }
            backForm.append(s);
        }
    }
}

public void commandAction(Command c, Displayable s)
{
    try
    {
        if (c == exitCommand)
        {
            destroyApp(false);
            notifyDestroyed();
        }
        else if (c == startCommand)
        {
            if (0 == destinationTextField.size())
```



```
        {
            backForm.append("\nPls input the destination
address!");
        }
        else
        {
            messconn =
(MessageConnection)Connector.open("mms://:com.mot.oyye");
            sendMMS();
            receiveMMS();
        }
    }
    else if (c == viewCommand)
    {
        viewMsg();
    }
}
catch (Exception e)
{
    e.printStackTrace();
}
}
/**
 * startApp should return immediately to keep the dispatcher
 * from hanging.
 */
public void startApp()
{
    display.setCurrent(resumeScreen);
}

/**
 * Remember what screen is showing
 */
public void pauseApp()
```




```
{
    resumeScreen = display.getCurrent();
}

/**
 * Destroy must cleanup everything.
 * @param unconditional true if a forced shutdown was requested
 */
public void destroyApp(boolean unconditional)
{
    try
    {
        if (messconn != null)
        {
            messconn.close();
        }
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
```



7.5 MIPD 2.0 Push Registry

7.5.1 Overview

Push registration lets a MIDlet set itself to be launched automatically. The push registry allows for registering network and timing based activation and also manages the MIDlet activation process defined by MIPD 2.0 push registry.

The iDEN implementation of push registry supports all methods defined in the MIPD 2.0 PushRegistry class.

7.5.2 Network Launch

The iDEN implementation supports three network protocols: datagram (UDP), socket (TCP) and SMS. An application can be statically registered by defining a property in a descriptor file or it can register dynamically by calling the register connection API during run time. To register an application for static socket (TCP) connections, the device must have packet data service. To receive inbound messages, the device must have packet data or SMS service. This can require special provisioning by the carrier or service provider. iDEN handsets support a maximum of twelve push registrations; a MIDlet may have multiple push registrations.

The iDEN implementation buffers the first incoming datagram or SMS message before it launches the MIDlet. Once the MIDlet launches, the connection delivers this message, and all subsequent messages are delivered directly to the application. The MIDlet is of course responsible for opening the connection using the Generic Connection framework. For sockets, the MIDlet is launched when a TCP connection is established, and the connection is transferred to an application after it is launched.

The sections below describe device-specific information regarding registration. For more information on PushRegistry consult the MIPD 2.0 specification.

7.5.3 Time-based Launch

Time-based launch is accomplished using the `registerAlarm()` method detailed below. Each application only has access to one alarm and only one future event can be pending. The maximum number of alarms that are available at any one time is 32. An application is launched only if the phone is powered on. If the phone is powered off and an alarm goes off, the application will not be launched.

7.5.4 Class Description

The API for the PushRegistry is located in package `javax.microedition.io`.

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



7.5.5 Method Description

7.5.5.1 PushRegistry Method

7.5.5.1.1 registerAlarm

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

You can delete a previously registered alarm by setting the time parameter to zero. The registered time must be local time. The time must be a minimum of two minutes in the future from the current time.

7.5.6 Tips /

- It's recommended that you open the connection immediately in a separate thread in the MIDlet's `startApp()` and read the received message.
- Applications should not use any reserved ports as defined by IANA; for example FTP, Telnet, or HTTP.



7.6 MIDP 2.0 Security API

7.6.1 Overview

The MIDP 2.0 Security API consists of `HttpsConnection`, `SecureConnection`, `SecurityInfo`, `Certificate` and `CertificateException`.

The `HttpsConnection` class defines the necessary methods and constants to establish a secure network connection. The URL that specifies `HTTPS` when passed to `Connector.open` will return an `HttpsConnection`.

The `SecureConnection` class defines the secure socket stream connection. A secure connection is established using `Connector.open()` and a URL that specifies `SSL`. The secure connection is established before the `open` method returns. If the secure connection cannot be established due to errors related to certificates, a `CertificateException` is thrown. A secure socket is accessed using a generic connection string with an explicit host and port number. The host may be specified as a fully qualified host name or IPv4 number. For example, `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 `SecurityInfo` class 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. that is in use.

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` 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.

7.6.2 Class Descriptions

The API for the `HttpsConnection`, `SecureConnection`, and `SecurityInfo` is located in package `java.microedition.io`. The API for the `Certificate` and `CertificateException` is located in package `java.microedition.pki`.

```
java.lang.Object
|
+ - java.microedition.io.HttpsConnection
|
+ - java.microedition.io.SecureConnection
|
+ - java.microedition.io.SecurityInfo
|
+ - java.microedition.pki.Certificate
|
+ - java.microedition.pki.CertificateException
```



7.6.3 Method Descriptions

Please refer to MIDP 2.0 Javadocs.

7.6.4 Code Examples

7.6.4.1 **HttpsConnection**

The following is the code example of `HttpsConnection`; open a HTTPS connection, set its parameters, then read the HTTP response.

```
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();
    }
}
```



7.6.4.2 SecureConnection

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

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

sc.setSocketOption(SocketConnection.LINGER, 5);

InputStream is = sc.openInputStream();
OutputStream os = sc.openOutputStream();

os.write("\r\n".getBytes());
int ch = 0;
while(ch != -1) {
    ch = is.read();
}

is.close();
os.close();
sc.close();
```

7.6.4.3 SecurityInfo

```
HttpsConnection c = null;
InputStream is = null;

c = (HttpsConnection) Connector.open("https://www.bellsouth.com/",
                                     Connector.READ_WRITE, true);
is = c.openInputStream();

try {
    secuInfo = c.getSecurityInfo();
} catch(Throwable t) {
    t.printStackTrace();
}

System.out.println(" ProtocolVersion "+secuInfo.getProtocolVersion());
System.out.println(" ProtocolName " + secuInfo.getProtocolName());
System.out.println(" CipherSuite " + secuInfo.getCipherSuite());
```

7.6.4.4 Certificate

```
Certificate cer = secuInfo.getServerCertificate();

System.out.println(" CA Type " + cer.getType());
System.out.println(" CA Version " + cer.getVersion());
System.out.println(" CA NotAfter " + cer.getNotAfter());
System.out.println(" CA NotBefore " + cer.getNotBefore());
System.out.println(" CA Subject " + cer.getSubject());
```



```
System.out.println(" CA Issuer " + cer.getIssuer());  
System.out.println(" CA SerialNumber " + cer.getSerialNumber());
```

7.6.4.5 CertificateException

```
try {  
    c = (HttpsConnection)Connector.open("https://www.bellsouth.com/",  
                                       Connector.READ_WRITE, true);  
    is = c.openInputStream();  
    . . . . .  
  
} catch (CertificateException ce) {  
    System.out.println ("Unexpected CertificateException " + ce);  
}
```

7.6.5 Tips /

- HTTPS is the secure version of HTTP (IETF RFC2616), a request-response protocol in which the parameters of the request must be set before the request is sent.
- In addition to the normal IOExceptions that may occur during invocation of the various methods that cause a transition to the Connected state, CertificateException (a subtype of IOException) may be thrown to indicate various failures related to establishing the secure link. The secure link is necessary in the Connected state so the headers can be sent securely. The secure link may be established as early as the invocation of `Connector.open()` and related methods for opening input and output streams and failure related to certificate exceptions may be reported.
- MIDP 2.0 devices are expected to operate using standard Internet and wireless protocols and techniques for transport and security. The current mechanisms for securing Internet content is based on existing Internet standards for public key cryptography:
 - [RFC2437] - PKCS #1 RSA Encryption Version 2.0
 - [RFC2459] - Internet X.509 Public Key Infrastructure
 - [WAPCERT] - WAP-211-WAPCert-20010522-a - WAP Certificate Profile Specification
- Only VeriSign server security certificates are supported. Using other server certificates will cause a CertificateException to be thrown.



7.7 Cryptography APIs

7.7.1 Overview



This API is only available on these handsets.

To complement SSL/TLS/HTTPS and enrich secure Java applications, iDEN handsets include a set of lightweight cryptography APIs that provide flexible and customizable end-to-end application-layer security in the J2ME™ environment. A rich variety of cryptographic mechanisms and algorithms are incorporated into these APIs, thus providing confidentiality, integrity and authentication. Cryptographic algorithms and schemes supported include: message digest (MD5 and SHA-1), secure random number generator (FIPS186 RNG), ciphers (DES, DESede, AES, RC4, and others), digital signatures (ECDSA and others) and key agreement (DH and ECDH).

7.7.2 Class Descriptions

The Crypto APIs are located in the packages `com.motorola.iden.crypto` and `com.motorola.iden.security`.

```
java.lang.Object
|
+-com.motorola.iden.crypto.Cipher
|
+-com.motorola.iden.crypto.KeyAgreement
|
+-com.motorola.iden.security.MessageDigestSpi
|
+-com.motorola.iden.security.MessageDigest
|
+-com.motorola.iden.security.Signature
```

7.7.2.1 MessageDigest Description

`MessageDigest` is a one-way hash function that takes arbitrary-sized data and outputs a fixed-length hash value. All the information in the message is used to construct the message digest, but the message cannot be recovered from the hash. The message digest provides data integrity.

- Algorithms MD5 and SHA-1 are supported in this platform.
- The `MessageDigest` class provides applications with the functionality of a message digest algorithm, such as MD5 or SHA-1.
- SHA-1 is a basic hash function that takes an entire message (or several parts of a single message submitted in separate blocks) and produces a 160-bit message digest value.



- MD5 is a hash function that takes an entire message (or several parts of a single message submitted in separate blocks) and produces a 128-bit message digest value.

7.7.2.2 Cipher Description

- Encryption is a tool used to protect data. Typical uses are to protect files in a file system or to encrypt network communications.

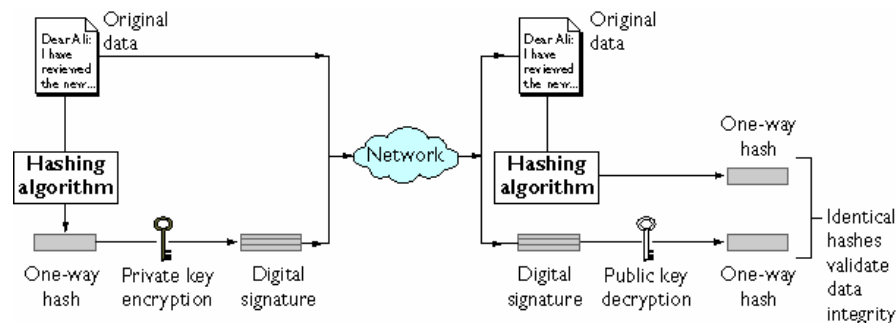
Two kinds of ciphers are supported:

- Symmetric Ciphers use a single secret key to encrypt and decrypt data.
- Asymmetric Ciphers use a pair of keys. One key is public and may be freely distributed. The other key is private and should be kept secret. Data encrypted with either key can be decrypted using the other key.

This class provides the functionality of a cryptographic cipher for encryption and decryption.

7.7.2.3 Signature Description

A digital signature is simply a message digest that has been processed with a signer's private key. The signature can be passed around with the data, providing proof that whoever signed the data had access to the private key.



The Signature class provides the functionality of signing and verifying a digital signature.

7.7.2.4 KeyAgreement Description

KeyAgreement can establish shared secrets without exchanging a secret key. KeyAgreement relies on public-public key pairs, just like asymmetric encryption. Your own private key and another party's public key generate the shared secret. The generated shared secret can be used as a key for symmetric encryption.

The class KeyAgreement that is located in package `com.motorola.iden.crypto` contains the method of generating and verifying a digital signature. Algorithms Diffie-Hellman (DH) and ECC Diffie-Hellman (ECDH) are supported. For algorithm DH, standard ANSI X9.63 KDF is followed and for ECDH, ANSI X9.42 KDF is followed. For ECDH, only curve WTLS-7 (160 bits) is supported.



7.7.3 Method Descriptions

7.7.3.1 MessageDigest Methods

7.7.3.1.1 getInstance

Creates a MessageDigest instance.

```
public static MessageDigest getInstance(String algorithm)
    throws NoSuchAlgorithmException
```

This method generates a MessageDigest instance, which implements one of the above algorithms.

`algorithm` is the name of the algorithm requested; for example, "MD5" or "SHA".

7.7.3.1.2 update

Updates the digest with the specified bytes.

```
public void update(byte[] input, int offset, int len)
```

This method updates the message digest with an array of bytes that represents one of several parts of a single message. A message can be submitted in separate blocks. This method can be called multiple times.

`input` is the array of bytes. `offset` is the offset to start from in the array of bytes. `len` is the number of bytes to use.

7.7.3.1.3 digest

Returns a completed digest, created from the parts specified with calls to `update()`.

```
public byte[] digest()
```

After you finish updating the entire message, this method finishes the operation and produces the digest.

7.7.3.2 Cipher Methods

7.7.3.2.1 getInstance

Creates a Cipher instance.

```
public static final Cipher getInstance(String transformation)
    throws NoSuchAlgorithmException, NoSuchPaddingException
```

This method generates a Cipher instance that represents a certain cipher algorithm and possible associated padding scheme.

`transformation` is the name of the transformation in the form "algorithm/mode/padding" or "algorithm"; for example, "DES/CBC/PKCS5Padding" or "DES".

If the transformation is specified by algorithm only, the mode and padding are set to the default values for the algorithm provider.

The following table lists all supported cipher algorithms, modes, and padding.



Supported Cipher Algorithms

Algorithm	Mode	Padding
DES	ECB;CBC;CFB;OFB	PKCS5Padding
DESede	ECB;CBC;CFB;OFB	PKCS5Padding
AES	ECB;CBC;CFB 128;OFB 128	PKCS5Padding
ARC4 (or RC4)	-----	-----

7.7.3.2.2 **init**

Initializes a Cipher instance with an operation mode, key, and algorithm specifications.

```
public final void init(int opmode, Key key,
    AlgorithmParameterSpec params)
    throws InvalidKeyException, InvalidAlgorithmParameterException
```

Before you perform any other operation on the Cipher, call this method to initialize it with the operation mode (encrypt or decrypt), a key, and the proper algorithm parameters (such as the initial vector).

`opmode` is the operation mode of this cipher (e.g. `ENCRYPT_MODE`, `DECRYPT_MODE`) .
`key` is the encryption key. `params` is the algorithm parameters.

7.7.3.2.3 **update**

Updates the cipher with the specified bytes.

```
public final byte[] update(byte[] input, int offset, int len)
    throws IllegalStateException
```

This method places information into the cipher to start or to continue a multiple-part encryption or decryption operation..

`input` is the input buffer. `offset` is the offset in `input` where the input starts . `len` is the input length.

7.7.3.2.4 **doFinal**

Returns a completed cipher, created from the parts specified with calls to `update()` .

```
public final byte[] doFinal()
    throws IllegalStateException, IllegalBlockSizeException,
    BadPaddingException
```

This method finishes a multiple-part encryption or decryption operation and produces the cipher (if the operation was encryption) or plain text (if the operation was decryption).



7.7.3.3 Signature Methods

7.7.3.3.1 getInstance

Creates a Signature instance.

```
public static Signature getInstance(String algorithm)
    throws NoSuchAlgorithmException
```

This method creates a Signature instance that implements the specified signature.

`algorithm` is the name of the algorithm, such as "ECDSA".

7.7.3.3.2 initSign

Initializes the Signature for signing with the specified key.

```
public final void initSign(PrivateKey privateKey)
    throws InvalidKeyException
```

Before you perform any signing operation, you must call this method to specify the private key.

`privateKey` is the private key of the identity whose signature is to be generated.

7.7.3.3.3 initVerify

Initializes the Signature for verification with the specified key.

```
public final void initVerify(PublicKey publicKey)
    throws InvalidKeyException
```

Before you perform any verification operation, you must call this method to specify the public key.

`publicKey` is the public key of the identity whose signature is going to be verified.

7.7.3.3.4 update

Updates the data to be signed or verified with the specified bytes.

```
public final void update(byte[] data, int offset, int len)
    throws SignatureException
```

This method updates the data to be signed or verified with the specified array of bytes.

`data` is the array of bytes. `offset` is the offset to start from in the array of bytes. `len` is the number of bytes to use, starting at `offset`.

7.7.3.3.5 sign

Returns the signature for the data specified with `update()`.

```
public final byte[] sign() throws SignatureException
```

This method returns the signature bytes of the input data. The format of the signature depends on the underlying signature scheme. Calling this method resets this signature object to the state it was in when initialized with `initSign()`.



7.7.3.3.6 **verify**

Returns true if the specified signature matches the data specified with `update()`.

```
public final boolean verify(byte[] signature)
    throws SignatureException
```

This method verifies that the specified signature is for the data specified with `update()`.

Calling this method resets this signature object to the state it was in when initialized with `initVerify()`.

7.7.3.4 **KeyAgreement Methods**

7.7.3.4.1 **getInstance**

Creates a `KeyAgreement` instance.

```
public static KeyAgreement getInstance(String algorithm)
    throws NoSuchAlgorithmException
```

This method generates a `KeyAgreement` object that implements the specified key agreement algorithm. In the current implementation, algorithm DH and ECDH are available.

`algorithm` is the name of the key agreement algorithm; that is "DH" or "ECDH".

7.7.3.4.2 **init**

Initializes the `KeyAgreement`.

```
public final void init(Key key, AlgorithmParameterSpec params)
    throws InvalidKeyException, InvalidAlgorithmParameterException
```

Before you can use the `KeyAgreement`, you must call this method to initialize it with the given key and set of algorithm parameters.

`key` is the party's private information. For example, in the case of the Diffie-Hellman key agreement, this would be the party's own Diffie-Hellman private key.

`params` is the key agreement parameters.

7.7.3.4.3 **doPhase**

Updates the `KeyAgreement` with a key received from one of the other parties involved in this key agreement.

```
public final Key doPhase(Key key, boolean lastPhase)
    throws InvalidKeyException
```

`key` is the key for this phase. For example, in the case of Diffie-Hellman between two parties, this would be the other party's Diffie-Hellman public key.

`lastPhase` is a Boolean flag that indicates whether this is the last phase of this key agreement. Currently, only one phase is supported so this argument should always be true. Using false causes this method to throw an exception.



7.7.3.4.4 generateSecret

Returns the shared secret based on the keys obtained from `init()` and `doPhase()`.

```
public final byte[] generateSecret()
```

This method resets this `KeyAgreement` instance, so that it can be reused for further key agreements. Unless this key agreement is reinitialized with one of the `init()` methods, the same private information and algorithm parameters are used for subsequent key agreements.

7.7.4 Example Code

7.7.4.1 MessageDigest Example #1

```
public CDemo1()
{
    byte[] message1 = new byte[25];
    byte[] message2 = new byte[250];
    byte[] digest;
    try{
        //get an Instance of MessageDigest whose algorithm
        //is MD5
        MessageDigest md = MessageDigest.getInstance ("MD5");

        //update message1 into MessageDigest context
        md.update(message1,0,25);

        //update part of message2 (start at element 2, length //125)
        // into MessageDigest context
        md.update(message2,2,125);

        //finalize and get MessageDigest
        digest = md.digest();
    } catch (NoSuchAlgorithmException) {}
}
```

7.7.4.2 MessageDigest Example #2

```
public CDemo2()
{
    byte[] message1 = new byte[25];
    byte[] message2 = new byte[250];
    byte[] digest;
    try{
        //get an Instance of MessageDigest whose algorithm
        //is SHA-1
        MessageDigest sha = MessageDigest.getInstance("SHA");

        //update message1 into MessageDigest context
        sha.update(message1,0,25);

        //update part of message2 (start at element 2, length //125)
```



```
// into MessageDigest context
sha.update(message2,2,125);

//finalize and get MessageDigest
digest = sha.digest();
} catch (NoSuchAlgorithmException) {}
}
```

7.7.4.3 Cipher Example

```
public cipherdemo1()
{
    //message needs to be encrypted
    String info = "Hello World!";

    //cipher
    byte[] cipher;

    //Decrypted message
    String output;

    try {
        //get a cipher instance for encryption
        Cipher A = Cipher.getInstance("DES/CBC/PKCS5Padding");

        //get a cipher instance for decryption
        Cipher B = Cipher.getInstance("DES/CBC/PKCS5Padding");

        //setup a des key
        byte key_input[] = {0,1,2,3,4,5,6,7};

        //key instance
        //DES_Key implements interface Key
        DES_Key key = new DES_Key(key_input);

        //initial vector
        byte [] iv;

        //init cipher A
        A.init(Cipher.ENCRYPT_MODE, key);

        //get generated IV
        iv = A.getIV();

        //encrypt
        cipher = A.doFinal(info.getBytes());

        //new IvParameterSpec for decryption
        IvParameterSpec ips = new IvParameterSpec(iv);

        //init cipher for decryption
        B.init(Cipher.DECRYPT_MODE, key, (AlgorithmParameterSpec) ips);
    }
}
```



```

        //decrypt the message
        byte out[] = B.doFinal(encrypted3);

        //get the decrypted info
        output = new String(out, 0, out.length);
    }
    catch (Exception e) {
    }
}

```

7.7.4.4 Signature Example

```

public CDemo3()
{
    Signature sig, verify;

    try {
        //get new Signature instance for signing.
        sig = Signature.getInstance("ECDSA");

        //setup ECDSAParameterSpec for initialization
        ECDSAParameterSpec ecdsaparameter = new
            ECDSAParameterSpec(Security.WTLS7,null);
        sig.setParameter(ecdsaparameter);

        //initialize for signing
        sig.initSign((ECC_PrivateKey)privatekey);

        //update the message to be signed
        sig.update("testtesttest".getBytes(),0,12);

        //get the signature (s-value)
        byte[] signature = sig.sign();

        //get the r-value and store it into ecdsaparameter
        ecdsaparameter = (ECDSAParameterSpec)sig.getParameter();

        //get new Signature instance for verifying
        verify = Signature.getInstance("ECDSA");

        //set ECDSAParameterSpec for verifying
        //setup both curve and r-value
        verify.setParameter(ecdsaparameter);

        //initialize for verifying
        verify.initVerify((ECC_PublicKey)publickey);

        //update the message to be verified
        verify.update("testtesttest".getBytes(),0,12);

        //verify
        boolean b = sig2.verify(signature);
    }
}

```




```
        catch (Exception e){
        }
    }
```

7.7.4.5 Key Address Example

```
public CDemo4()
{
    //initialize variables used in this Key Agreement
    KeyAgreement dh;
    KeyAgreement dh2;

    KeyPair keypair;
    KeyPair keypair2;

    DHParameterSpec dhspec;
    DHParameterSpec dhspec2;

    KeyPairGenerator dhgen;
    KeyPairGenerator dhgen2;

    PublicKey publickey;
    PublicKey publickey2;

    PrivateKey privatekey;
    PrivateKey privatekey2;

    byte[] BobS;
    byte[] AliceS;

    int i;
    try {
        //BOB
        //create dhspec
        dhspec = new DHParameterSpec(p,g,q);

        //create dhgen
        dhgen = KeyPairGenerator.getInstance("DH");

        //init dhgen
        dhgen.initialize(dhspec);

        //gen keypair
        keypair = dhgen.generateKeyPair();

        //get publickey and privatekey for dh
        publickey = keypair.getPublic();
        privatekey = keypair.getPrivate();

        //Alice
        //create dhspec
        dhspec2 = new DHParameterSpec(p,g,q);
```



```
//create dhgen
dhgen2 = KeyPairGenerator.getInstance("DH");

//init dhgen
dhgen2.initialize(dhspec2);

//gen keypair
keypair2 = dhgen2.generateKeyPair();

//get publickey and privatekey for dh
publickey2 = keypair2.getPublic();
privatekey2 = keypair2.getPrivate();

//get dh
dh = KeyAgreement.getInstance("DH");

//init dh
dh.init((DH_PrivateKey)privatekey,dhspec);

//doPhase
dh.doPhase((DH_PublicKey)publickey2,true);

//generate secret key using Bob's private key and Alice's
//public Key
BobS = dh.generateSecret();

//get dh
dh2 = KeyAgreement.getInstance("DH");

//init dh
dh2.init((DH_PrivateKey)privatekey2,dhspec2);

//doPhase
dh2.doPhase((DH_PublicKey)publickey,true);

//generate secret key using Alice's private key and Bob's
//public Key
AliceS = dh2.generateSecret();

} catch (Exception e) {
}
}
```



7.7.5 Tips /

- In order to use DES, DESede, AES, and ARC4, a MIDlet must implement the Key interface.
- DES supports 56-bit key (8 bytes, including parity). DES key parity check and weak key detection are not supported.
- DESede, also called 3DES ("triple DES"), supports 168-bit keys (24 bytes, including parity). Parity check and weak key detection are not supported.
- AES supports 128, 192 or 256-byte key.
- ARC4, also called RC4, supports a key size that is less than 256 bits.

7.7.6 Compiling & Testing Cryptography Enhanced MIDlets

This tip is only applicable to the stub classes for emulators: Instead of executing actual cryptographic operations, console messages are displayed for certain operations. This allows rudimentary debugging of applications without actual cryptographic operations.



7.8 JAXP

7.8.1 Overview



This API is only available on these handsets.

JAXP provides APIs that allow a J2ME application to access a XML parser and parse XML document using SAX. The detailed description of JAXP subset is defined in J2ME Web Services Specification 1.0.

- The `javax.xml.parsers`, `org.xml.sax` and `org.xml.sax.helpers` packages contain the basic classes needed to access XML parser to parse XML document.
- This feature supports obtaining and referencing a platform's given parser implementation. It includes the following classes in JSR172: `SAXParser`; `SAXParserFactory`, `ParserConfigurationException`, `FactoryConfigurationError`.
- This feature contains a subset of the SAX 2.0 API classes and interfaces. It includes `Attributes`, `Locator`, `InputSource`, `SAXException`, `SAXNotRecognizedException`, `SAXNotSupportedException`, `SAXParseException`.
- This feature supports applications to extend to receive parse events.

7.8.2 Package `javax.xml.parsers`

Class Summary	
<code>SAXParser</code>	Defines the API that represents a simple SAX parser.
<code>SAXParserFactory</code>	Defines a factory API that enables applications to configure and obtain a SAX based parser to parse XML documents.
Exception Summary	
<code>ParserConfigurationException</code>	Indicates a serious configuration error. It will be thrown when a parser cannot be created which satisfies the request configuration when invoking <code>SAXParserFactory.newSAXParser()</code> .
Error Summary	
<code>FactoryConfigurationError</code>	Thrown when a problem with configuration with the Parser Factories exists. This exception will be thrown when the class of a parser factory specified in the system properties cannot be found or instantiated.



7.8.3 Package org.xml.sax

Interface Summary	
Attributes	Interface for a list of XML attributes.
Locator	Interface for associating a SAX event with a document location.
Class Summary	
InputSource	A single input source for an XML entity.
Exception Summary	
SAXException	Encapsulates a general SAX error or warning. It is extended by the following three exceptions:
SAXNotRecognizedException	Exception class for an unrecognized identifier. A DefaultHandler will throw this exception when it finds an unrecognized feature or property identifier
SAXNotSupportedException	Exception class for an unsupported operation. A DefaultHandler will throw this exception when it recognizes a feature or property identifier, but cannot perform the requested operation (setting a state or value).
SAXParseException	Encapsulates a XML parse error or warning. This exception is passed to DefaultHandler's error(), fatalError(), warning() as parameters to report the information when an error occurs in the original XML document.

7.8.4 Package org.xml.sax.helpers

Class Summary	
DefaultHandler	Default base class for SAX2 event handlers.



7.8.5 Package Tree

7.8.5.1 Class Hierarchy

The following will be the Class Hierarchy for the JAXP API:

- o class java.lang.Object
 - o class org.xml.sax.helpers.DefaultHandler
 - o class org.xml.sax.InputSource
 - o class javax.xml.parsers.SAXParser
 - o class javax.xml.parsers.SAXParserFactory
 - o class java.lang.Throwable
 - o class java.lang.Error
 - o class javax.xml.parsers.FactoryConfigurationError
 - o class java.lang.Exception
 - o class javax.xml.parsers.ParserConfigurationException
 - o class org.xml.sax.SAXException
 - o class org.xml.sax.SAXNotRecognizedException
 - o class org.xml.sax.SAXNotSupportedException
 - o class org.xml.sax.SAXParseException

7.8.5.2 Interface Hierarchy

The following will be the Interface Hierarchy for the JAXP API :

- o interface org.xml.sax.Attributes
- o interface org.xml.sax Locator

7.8.5.3 Class javax.xml.parsers.SAXParser

```
parse(InputStream is, DefaultHandler dh)
```

Parses the content of the given InputStream instance as XML using the specified DefaultHandler.

```
parse(InputSource is, DefaultHandler dh)
```

Parses the content given InputSource as XML using the specified DefaultHandler.

Tip. The implementation will use the InputSource object to determine how to read XML input. If there is a character stream available, the parser will read that stream directly; if not, the parser will use a byte stream, if available; if neither a character stream nor a byte stream is available, the parser will attempt to open a connection to the resource identified by the system identifier.



7.8.5.4 Class `javax.xml.parsers.SAXParserFactory`

7.8.5.4.1 `newInstance()`

```
public static SAXParserFactory newInstance() throws  
FactoryConfigurationError
```

Obtain a new instance of `SAXParserFactory`.

Tip: This static method creates a new factory instance of the platform default `SAXParserFactory` instance. Once an application has obtained a reference to a `SAXParserFactory` it can use the factory to configure and obtain parser instances.

7.8.5.4.2 `newSAXParser()`

Creates a new instance of a `SAXParser` using the currently configured factory parameters.

7.8.5.4.3 `setFeature(String name, boolean value)/getFeature(String name)`

Sets/gets the particular feature in the underlying implementation.

Tips:

- A list of the core features and properties can be found at <http://www.megginson.com/SAX/Java/features.html>
- NAMESPACES and NAMESPACE_PREFIXES are supported features. When processing VALIDATION a `SAXNotSupportedException` will be thrown. When processing any other features a `SAXNotRecognizedException` will be thrown.

7.8.5.4.4 Class `org.xml.sax.helpers.DefaultHandler`

Default base class for SAX2 event handlers. This class is available as a convenience base class for SAX2 applications: it provides default implementations for applications to extend. Application writers can extend this class when they need to implement only part of an interface.

7.8.5.4.5 Interface `org.xml.sax.Attributes`

This interface simply provides a list of XML attributes. The list can be accessed in three ways:

- by attribute index;
- by Namespace-qualified name; or
- by qualified (prefixed) name.

If the namespace-prefixes feature (see above) is *false*, access by qualified name may not be available.. If the `http://xml.org/sax/features/namespaces` feature is *false*, access by Namespace-qualified names may not be available.



7.8.6 Code Examples

The following is the code example of JAXP:

```
/**
 * Demo program of Motorola iDEN SDK JAXP APIs
 * Filename: MyJAXP.java
 * <p></p>
 * <hr/>
 * <b>MOTOROLA and the Stylized M Logo are registered trademarks of
 * Motorola, Inc. Reg. U.S. Pat. & Tm. Off.<br>
 * &copy; Copyright 2003 Motorola, Inc. All Rights Reserved.</b>
 * <hr/>
 *
 * @version iDEN JAXP demo 1.0
 * @author Motorola, Inc.
 */
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;

public class MyJAXP extends MIDlet
    implements CommandListener
{

    private Form textform;
    ParserApp parser;
    XMLDataStore xmlData;

    Command testCommand,exitCommand;

    public MyJAXP()
    {
        textform = new Form("XML Test Form!");
        testCommand = new Command("Test", 1, 1);
        exitCommand = new Command("Exit", 1, 2);
        textform.addCommand(testCommand);
        textform.addCommand(exitCommand);
        textform.setCommandListener(this);
    }

    public void startApp()
    {
        Display.getDisplay(this).setCurrent(textform);
        parser = new ParserApp();
        xmlData = new XMLDataStore();
    }

    public void pauseApp()
    {
    }
}
```




```

public void destroyApp(boolean flag)
{
}

public void commandAction(Command command, Displayable
displayable)
{
    if (command == testCommand)
    {
        String s = null;
        s = "fire.xml";
        xmlData.resetXMLDataStore();
        parser.startParsing(xmlData, new String(s));
        Display.getDisplay(this).setCurrent(textform);
    }
    else if (command == exitCommand)
    {
        notifyDestroyed();
    }
}
}

/**
 * Demo program of Motorola iDEN SDK JAXP APIs
 * Filename: XMLDataStore.java
 * <p></p>
 * <hr/>
 * <b>MOTOROLA and the Stylized M Logo are registered trademarks of
 * Motorola, Inc. Reg. U.S. Pat. & Tm. Off.<br>
 * &copy; Copyright 2003 Motorola, Inc. All Rights Reserved.</b>
 * <hr/>
 *
 * @version iDEN JAXP demo 1.0
 * @author Motorola, Inc.
 */

import java.util.Vector;
import org.xml.sax.Attributes;

class XMLDataStore
{
    Vector elements;
    Vector attrList;

    public XMLDataStore()
    {
        elements = new Vector();
        attrList = new Vector();
    }

    public void resetXMLDataStore()

```



```

    {
        elements.removeAllElements();
        attrList.removeAllElements();
    }

    public void insertElement(String s, Attributes attributes)
    {
        Attributes attributes1 = attributes;
        elements.addElement(s);
        int i = attributes.getLength();
        Vector vector = new Vector(4);
        for(int j = 0; j < i; j++)
            vector.addElement(new String(attributes.getValue(j)));

        attrList.addElement(vector);
    }

    public String getElement(int i)
    {
        return (String)elements.elementAt(i);
    }

    public Vector getAttributes(int i)
    {
        return (Vector)attrList.elementAt(i);
    }

    public String getAttribute(int i, int j)
    {
        Vector vector = (Vector)attrList.elementAt(i);
        return vector.elementAt(j).toString();
    }

    public int getSize()
    {
        return elements.size();
    }
}

/**
 * Demo program of Motorola iDEN SDK JAXP APIs
 * Filename: ParserApp.java
 * <p></p>
 * <hr/>
 * <b>MOTOROLA and the Stylized M Logo are registered trademarks of
 * Motorola, Inc. Reg. U.S. Pat. & Tm. Off.<br>
 * &copy; Copyright 2003 Motorola, Inc. All Rights Reserved.</b>
 * <hr/>
 *
 * @version iDEN JAXP demo 1.0
 * @author Motorola, Inc.
 */
import java.io.PrintStream;

```



```
import java.io.Writer;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.Attributes;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.SAXParseException;

class ParserApp extends DefaultHandler
{
    SAXParserFactory factory;
    SAXParser myParser;
    DefaultHandler handler;
    private Writer out;
    int noOfElements;
    XMLDataStore xmlData;

    public ParserApp()
    {
        noOfElements = 0;
        xmlData = null;
    }

    public void startParsing(XMLDataStore xmldatastore, String s)
    {
        xmlData = xmldatastore;
        try
        {
            factory = SAXParserFactory.newInstance();
            myParser = factory.newSAXParser();
        }
        catch(Exception exception)
        {
            System.out.println("Couldn't create parser");
        }
        try
        {
            handler = this;
            java.io.InputStream inputstream = null;
            Class class1 = null;
            class1 = getClass();
            inputstream = class1.getResourceAsStream(s);
            myParser.parse(inputstream, handler);
        }
        catch(Exception exception1)
        {
            System.out.println("Some Exception occured");
            exception1.printStackTrace();
        }
    }

    public void startDocument()
    {

```



```
        System.out.println("start document here");
        emit("<?xml version=1.0 encoding=utf-8 standalone=yes
?>\n");
    }

    public void endDocument()
    {
        emit("\nEND OF DOCUMENT");
    }

    public void startElement(String s, String s1, String s2,
Attributes attributes)
    {
        System.out.println("a start element");
        Attributes attributes1 = attributes;
        xmlData.insertElement(s2, attributes1);
        emit("<" + s2);
        int i = 0;
        i = attributes.getLength();
        for(int j = 0; j < i; j++)
        {
            emit("\t" + attributes.getQName(j));
            emit("=" + attributes.getValue(j));
        }

        emit(">");
    }

    public void endElement(String s, String s1, String s2)
    {
        emit("</" + s2 + ">");
        System.out.println("a end element");
    }

    public void characters(char ac[], int i, int j)
    {
        String s = new String(ac, i, j);
        s.trim();
        emit(s);
    }

    public void fatalError (SAXParseException e)
    {
        System.out.println("fatal error here");
    }

    public void emit(String s)
    {
        System.out.println(s);
    }
}
```



7.8.7 Compiling & Testing JAXP MIDlets

- Method `SAXParserFactory.isValidating()` & `SAXParser.isValidating()` always returns false since the implementation supports a non-validating parser only.
- Method `SAXParserFactory.setValidating()` does nothing since the implementation supports a non-validating parser only.



7.9 JAX-RPC

7.9.1 Overview



This API is only available on this handset.

JAX-RPC provides an API that allows a J2ME application to dispatch Remote Procedure Call (RPC) to remote SOAP / XML based web services. The detailed description of the JAX-RPC subset is defined in J2ME Web Services Specification 1.0.

The `javax.microedition.xml.rpc`, `javax.xml.namespace`, `javax.xml.rpc` and `java.rmi` packages contain the basic classes needed to dispatch Remote Procedure Call (RPC) to remote SOAP / XML based web services.

7.9.2 Package `javax.microedition.xml.rpc`

Interface Summary	
FaultDetailHandler	Implemented by stubs that handle custom faults.
Class Summary	
ComplexType	Provides a special Type instance used to represent an <code>xsd:complexType</code> defined in a Web Service's WSDL definition.
Element	Provides a special Object used to represent an <code>xsd:element</code> defined in a Web Service's WSDL definition.
Operation	Corresponds to a <code>wsdl:operation</code> defined for a target service endpoint.
Type	Provides a type safe enumeration of allowable types that are used to identify simple types defined in a Web Service's WSDL definition.
Exception Summary	
FaultDetailException	Returns service specific exception detail values, and an associated QName, to a Stub instance.

7.9.3 Package `javax.xml.namespace`

Class Summary	
QName	Represents a qualified name as defined in the XML specifications: XML Schema Part2: Datatypes specification, Namespaces in XML, Namespaces in XML Errata.



7.9.4 Package javax.xml.rpc

Interface Summary	
Stub	Which is the interface for javax.xml.rpc.Stub, the common base interface for the stub classes.
Class Summary	
NamespaceConstants	Constants used in JAX-RPC for namespace prefixes and URIs
Exception Summary	
JAXRPCException	Which is thrown from the core JAX-RPC APIs to indicate an exception related to the JAX-RPC runtime mechanisms.

7.9.5 Package java.rmi

Interface Summary	
Remote	Serves to identify interfaces whose methods may be invoked from a non-local virtual machine.
Exception Summary	
MarshalException	This exception is thrown if a java.io.IOException occurs while marshalling the remote call header, arguments, or return value for a remote method call.
RemoteException	This exception is the common superclass for a number of communication-related exceptions that may occur during the execution of a remote method call.
ServerException	This exception is thrown as a result of a remote method invocation when a RemoteException is thrown while processing the invocation on the server, either while unmarshalling the arguments, executing the remote method itself, or marshalling the return value.



7.9.6 Class and Interface Hierarchy

The following is the Class Hierarchy for the JAX-RPC API:

- class java.lang.Object
 - class javax.microedition.xml.rpc.Operation
 - class javax.microedition.xml.rpc.Type
 - class javax.microedition.xml.rpc.Element
 - class javax.microedition.xml.rpc.ComplexType
 - class javax.xml.namespace.QName
 - class javax.xml.rpc.NamespaceConstants
 - class java.lang.Throwable
 - class java.lang.Exception
 - class javax.microedition.xml.rpc.FaultDetailException
 - class java.io.IOException
 - class java.rmi.RemoteException
 - class java.rmi.MarshalException
 - class java.rmi.ServerException
 - class java.lang.RuntimeException
 - class javax.xml.rpc.JAXRPCException

The following is the Interface Hierarchy for the JAX-RPC API:

- class java.lang.Object
 - interface javax.microedition.xml.rpc.FaultDetailHandler
 - interface javax.xml.rpc.Stub
 - interface java.rmi.Remote



7.9.7 Development Procedure

Web service developers do not need to use the above classes directly to develop web service applications. Instead, the application will call the functions of stubs and then the stubs call the functions provided by the above classes. The aim of this document is to instruct the developer how to write the client application to access the web services when receiving a WSDL file with the WSDL-to-java tool.

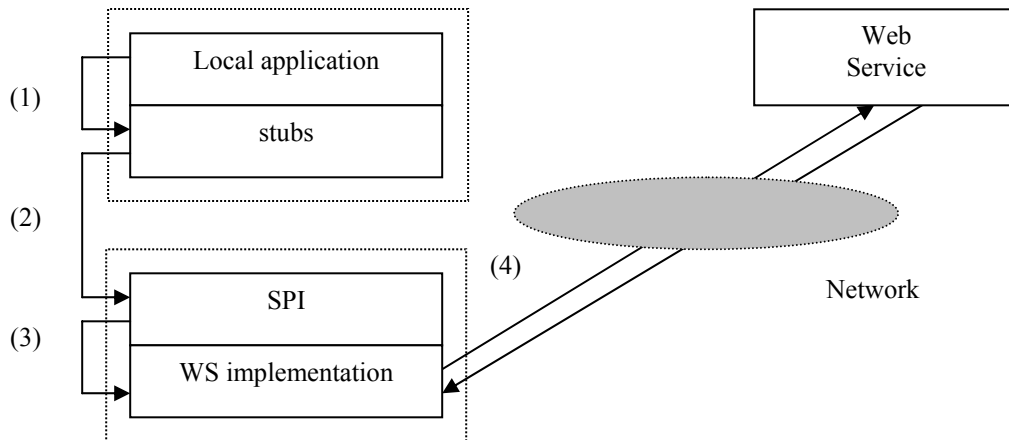
7.9.8 Background Knowledge

Web services enable local applications to call service procedures located on remote servers. The web services are designed to be platform and language independent, so a standard file is needed to describe the service – the WSDL file. WSDL files describe the operations, the request/response message format for each operation, the message format binding and transport binding and service URI. Appendix A gives a sample WSDL file.

In order to develop a client application for one kind of web service, client application developers should get the WSDL file for this web service, then use the WSDL-to-java tool (for example, Sun's WTK) to create stubs.

The stubs generated by a WSDL-to-java tool include: a) A public class for each complexType defined in the web service's WSDL file. b) A service endpoint interface - a public interface for the portType and binding operations defined in the web services's WSDL file. c) A stub file that implements the portType and binding class, for creating connections with the service endpoint and handling data streaming.

The interaction between the local application, the stubs, the Service Provider Interface, and the J2ME Web Services implementation is showed in Figure 1.



Typical RPC Scenario

(1) The local application makes calls to the stubs.

(2) The stub makes calls to the Service Provider Interface (SPI).

The SPI defines the interface between the stubs and the web service implementation. It ensures that any stub generated on any implementation of J2ME Web Services will work with any other implementation.

The SPI is used by the generated stub to execute RPC calls. The SPI is defined by the Type, Element, Complex Type and Operation classes. These classes are used by the stub to describe the input parameters and return type of an RPC. An object graph of these classes represents a description of the serialization of the values in a complex type.

(3) The SPI makes calls to the Java Web Services implementation.

The J2ME Web Services implementation uses the information received from the stub via the SPI to form SOAP message and open a network connection to send the message to remote service; After receiving the response SOAP message from the remote service, the J2ME Web Services implementation will parse it and return the result to the stubs via the SPI.



7.9.9 Development Steps

In this section, an example is given to explain how to develop a web service application.

- (1) Get the WSDL for one kind of web service. Web service providers publish the WSDL files on the Internet. The WSDL file in [appendix A](#) is used as an example WSDL. In this service, if the string "<username>!" is sent to server, the server will send the string "Hello <username>!" back.
- (2) Use the WSDL as the input file for the WTK or another WSDL-to-java tool to generate the stub code. In this case, 4 files are generated: HelloIF.java, HelloIF_Stub.java, SayHello.java and SayHelloResponse.java. They are put in [appendix B](#). HelloIF.java defines the service interface. HelloIF_Stub.java is the implementation class of interface HelloIF and interface javax.xml.rpc.Stub. SayHello.java and SayHelloResponse.java are classes for complexType defined in the web service's WSDL file.
- (3) After the stub code is generated, the application can invoke the remote procedure call (RPC). First, the application should create an instance of HelloIF_Stub: stub = new HelloIF_Stub(); Second, the application can call stub._setProperty() to set properties that are needed to invoke an RPC:

```
//Set the service address
```

```
stub._setProperty(javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,
    "http://localhost:8080/authhello-jaxrpc/authhello");
```

```
//Set the username and password if the service requires authentication
```

```
stub._setProperty(javax.xml.rpc.Stub.PASSWORD_PROPERTY,
    "xdl_1234");
stub._setProperty(javax.xml.rpc.Stub.USERNAME_PROPERTY,
    "tinaf");
```

Finally, the application can place the remote call stub.sayHello("Tina!").

7.9.10 Sample Application

7.9.10.1 Code Sample

The following is the whole application source code of the example mentioned above:

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;
import java.io.*;
import staticstub.*;
import java.rmi.*;
import javax.xml.rpc.*;
```



```
import javax.xml.namespace.*;

public class BasicAuth extends MIDlet
implements CommandListener
{
    private Form homeScr;
    private Form settingsScr;
    private StringItem anwser;
    TextBox errorScr;
    TextBox msgScr;
    Command exitC, goC, settingsC;
    Command backC, applyC;
    TextField stdEndpoint;
    TextField userName;
    TextField passWord;

    Display display; // The display for this MIDlet

    HelloIF_Stub stub;

    public BasicAuth()
    {
        display = Display.getDisplay(this);

        homeScr = new Form("web service Test Form!");
        homeScr.setCommandListener(this);

        settingsScr = new Form("Settings");
        settingsScr.setCommandListener(this);
        stdEndpoint =
        new TextField("Standard Endpoint", "http://", 255,
        TextField.ANY);
        settingsScr.append(stdEndpoint);
        userName = new TextField("User Name", null, 50,
        TextField.ANY |
        TextField.INITIAL_CAPS_WORD);
        passWord = new TextField("Pass Word", null, 50,
        TextField.ANY |
        TextField.INITIAL_CAPS_WORD);
        settingsScr.append(userName);
        settingsScr.append(passWord);

        errorScr = new TextBox("Error:", null, 500,
        TextField.ANY);
        errorScr.setCommandListener(this);
        msgScr = new TextBox("Message:", null, 500,
        TextField.ANY);
        msgScr.setCommandListener(this);
    }
}
```



```
exitC = new Command("Exit", Command.EXIT, 1);
settingsC = new Command("Settings", Command.SCREEN, 1);
goC = new Command("Go", Command.SCREEN, 2);
backC = new Command("Back", Command.BACK, 1);
applyC = new Command("Apply", Command.SCREEN, 1);

stub = new HelloIF_Stub();

//default

stub._setProperty(javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,
    "http://localhost:8080/authhello-
jaxrpc/authhello");

stub._setProperty(javax.xml.rpc.Stub.PASSWORD_PROPERTY,
    "xdl_1234");
stub._setProperty(javax.xml.rpc.Stub.USERNAME_PROPERTY,
    "tinaf");
}

public void startApp()
{
    showHomeScreen();
}

public void pauseApp()
{
}

public void destroyApp(boolean flag)
{
}

void showHomeScreen() {
    clearCommands(homeScr);
    homeScr.addCommand(exitC);
    homeScr.addCommand(goC);
    homeScr.addCommand(settingsC);
    display.setCurrent(homeScr);
}

void showSettingsScreen() {
    settingsScr.setTitle("Settings");
    clearCommands(settingsScr);
    settingsScr.addCommand(backC);
    settingsScr.addCommand(applyC);
    display.setCurrent(settingsScr);
}

void showErrorScreen(String text) {
    clearCommands(errorScr);
}
```



```
errorScr.setString(text);
errorScr.addCommand(backC);
display.setCurrent(errorScr);
}

void showMsgScreen(String text) {
clearCommands(msgScr);
msgScr.setString(text);
msgScr.addCommand(backC);
display.setCurrent(msgScr);
}

void clearCommands(Displayable d) {
d.removeCommand(exitC);
d.removeCommand(settingsC);
d.removeCommand(backC);
d.removeCommand(applyC);

d.removeCommand(goC);
}

boolean setSettings() {

stub._setProperty(
javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,
stdEndpoint.getString());
stub._setProperty(javax.xml.rpc.Stub.PASSWORD_PROPERTY,
passWord.getString());
stub._setProperty(javax.xml.rpc.Stub.USERNAME_PROPERTY,
userName.getString());

return true;
}

public void commandAction(Command command, Displayable s)
{
    if (command == goC)
    {
        System.out.println("Web service Test start
here...");

        try {

            showMsgScreen(stub.sayHello("Tina!"));

        }
        catch (java.rmi.RemoteException re) {
            showErrorScreen(re.getMessage());
            re.printStackTrace();
        }
        catch (Throwable t) {
            showErrorScreen(t.getMessage());
        }
    }
}
```



```

        }
        return;
    }

    if (command == settingsC) {
        //have a chance to change the endpoint
        showSettingsScreen();
        return;
    }
    if (command == applyC) {
        if (setSettings()) {
            showHomeScreen();
        } else {
            settingsScr.setTitle("Settings: Retry");
        }
        return;
    }

    if (command == backC) {
        if (s == settingsScr || s == msgScr || s == errorScr)
        {
            showHomeScreen();
        }

        return;
    }

    if (command == exitC)
    {
        notifyDestroyed();
    }

}

}

```

7.9.10.2 WSDL Sample

```

<?xml version="1.0" encoding="UTF-8"?>

<definitions name="MyHelloService"
targetNamespace="urn:Foo"
xmlns:tns="urn:Foo"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
    <types>
<schema targetNamespace="urn:Foo"
xmlns:tns="urn:Foo"
xmlns:soap11-enc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://www.w3.org/2001/XMLSchema">

```



```

    <import
namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
    <complexType name="sayHello">
        <sequence>
            <element name="String_1" type="string"
nillable="true"/>
        </sequence>
    </complexType>
    <complexType name="sayHelloResponse">
        <sequence>
            <element name="result" type="string" nillable="true"/>
        </sequence>
    </complexType>
    <element name="sayHello" type="tns:sayHello"/>
</schema>
</types>
<message name="HelloIF_sayHello">
    <part name="parameters" element="tns:sayHello"/></message>
<message name="HelloIF_sayHelloResponse">
    <part name="result"
element="tns:sayHelloResponse"/></message>
<portType name="HelloIF">
    <operation name="sayHello">
        <input message="tns:HelloIF_sayHello"/>
        <output
message="tns:HelloIF_sayHelloResponse"/></operation></portType>
    <binding name="HelloIFBinding" type="tns:HelloIF">
        <operation name="sayHello">
            <input>
                <soap:body use="literal"/></input>
            <output>
                <soap:body use="literal"/></output>
            <soap:operation soapAction=""/></operation>
        </binding>
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    </binding>
    <service name="MyHelloService">
        <port name="HelloIFPort" binding="tns:HelloIFBinding">
            <soap:address location="REPLACE_WITH_ACTUAL_URL"/></port>
        </service>
    </definitions>

```

7.9.10.3 Generated stub code

7.9.10.3.1 HelloIF.java

```

// This class was generated by 172 StubGenerator.
// Contents subject to change without notice.
// @generated

```

```
package staticstub;
```

```
public interface HelloIF extends java.rmi.Remote {
```




```

    public java.lang.String sayHello(java.lang.String string_1)
    throws java.rmi.RemoteException;

}

```

7.9.10.3.2 HelloIF_Stub.java

```

// This class was generated by 172 StubGenerator.
// Contents subject to change without notice.
// @generated

package staticstub;

import javax.xml.rpc.JAXRPCException;
import javax.xml.namespace.QName;
import javax.microedition.xml.rpc.Operation;
import javax.microedition.xml.rpc.Type;
import javax.microedition.xml.rpc.ComplexType;
import javax.microedition.xml.rpc.Element;

public class HelloIF_Stub implements staticstub.HelloIF,
javax.xml.rpc.Stub {
    private String[] _propertyNames;
    private Object[] _propertyValues;

    public HelloIF_Stub() {
        _propertyNames = new String[]
{ENDPOINT_ADDRESS_PROPERTY};
        _propertyValues = new Object[]
{"REPLACE_WITH_ACTUAL_URL"};
    }

    public void _setProperty(String name, Object value) {
        int size = _propertyNames.length;
        for (int i = 0; i < size; ++i) {
            if (_propertyNames[i].equals(name)) {
                _propertyValues[i] = value;
                return;
            }
        }
        // Need to expand our array for a new property
        String[] newPropNames = new String[size + 1];
        System.arraycopy(_propertyNames, 0, newPropNames, 0,
size);
        _propertyNames = newPropNames;
        Object[] newPropValues = new Object[size + 1];
        System.arraycopy(_propertyValues, 0, newPropValues,
0, size);
        _propertyValues = newPropValues;

        _propertyNames[size] = name;
        _propertyValues[size] = value;
    }
}

```



```

public Object _getProperty(String name) {
    for (int i = 0; i < _propertyNames.length; ++i) {
        if (_propertyNames[i].equals(name)) {
            return _propertyValues[i];
        }
    }
    if (ENDPOINT_ADDRESS_PROPERTY.equals(name) ||
    USERNAME_PROPERTY.equals(name) ||
    PASSWORD_PROPERTY.equals(name)) {
        return null;
    }
    if (SESSION_MAINTAIN_PROPERTY.equals(name)) {
        return new java.lang.Boolean(false);
    }
    throw new JAXRPCException("Stub does not recognize
property: "+name);
}

protected void _prepOperation(Operation op) {
    for (int i = 0; i < _propertyNames.length; ++i) {
        op.setProperty(_propertyNames[i],
        _propertyValues[i].toString());
    }
}

//
// Begin user methods
//

public java.lang.String sayHello(java.lang.String string_1)
throws java.rmi.RemoteException {
    // Copy the incoming values into an Object array if
    needed.
    Object[] inputObject = new Object[1];
    inputObject[0] = string_1;

    Operation op = Operation.newInstance(_qname_sayHello,
    _type_sayHello, _type_sayHelloResponse);
    _prepOperation(op);
    op.setProperty(Operation.SOAPACTION_URI_PROPERTY,
    "");
    Object resultObj;
    try {
        resultObj = op.invoke(inputObject);
    } catch (JAXRPCException e) {
        Throwable cause = e.getLinkedCause();
        if (cause instanceof java.rmi.RemoteException)
        {
            throw (java.rmi.RemoteException) cause;
        }
        throw e;
    }
    java.lang.String result;

```



```

        // Convert the result into the right Java type.
        // Unwrapped return value
        Object resultObj2 = ((Object[])resultObj)[0];
        result = (java.lang.String)resultObj2;
        return result;
    }
    //
    // End user methods
    //

    protected static final QName _qname_String_1 = new
    QName("", "String_1");
    protected static final QName _qname_result = new QName("",
    "result");
    protected static final QName _qname_sayHello = new
    QName("urn:Foo", "sayHello");
    protected static final QName _qname_sayHelloResponse = new
    QName("urn:Foo", "sayHelloResponse");
    protected static final Element _type_sayHello;
    protected static final Element _type_sayHelloResponse;
    static {
        // Create all of the Type's that this stub uses,
        once.
        Element _type_String_1;
        _type_String_1 = new Element(_qname_String_1,
        Type.STRING);
        ComplexType _complexType_sayHello;
        _complexType_sayHello = new ComplexType();
        _complexType_sayHello.elements = new Element[1];
        _complexType_sayHello.elements[0] = _type_String_1;
        _type_sayHello = new Element(_qname_sayHello,
        _complexType_sayHello);
        Element _type_result;
        _type_result = new Element(_qname_result,
        Type.STRING);
        ComplexType _complexType_sayHelloResponse;
        _complexType_sayHelloResponse = new ComplexType();
        _complexType_sayHelloResponse.elements = new
        Element[1];
        _complexType_sayHelloResponse.elements[0] =
        _type_result;
        _type_sayHelloResponse = new
        Element(_qname_sayHelloResponse,
        _complexType_sayHelloResponse);
    }
}

```

7.9.10.3.3 SayHello.java

```

// This class was generated by the JAXRPC SI, do not edit.
// Contents subject to change without notice.
// JSR-172 Reference Implementation wscompile 1.0, using:
JAX-RPC Standard Implementation (1.1, build R59)

```



```
package staticstub;

public class SayHello {
    protected java.lang.String string_1;

    public SayHello() {
    }

    public SayHello(java.lang.String string_1) {
        this.string_1 = string_1;
    }

    public java.lang.String getString_1() {
        return string_1;
    }

    public void setString_1(java.lang.String string_1) {
        this.string_1 = string_1;
    }
}
```

7.9.10.3.4 SayHelloResponse.java

```
// This class was generated by the JAXRPC SI, do not edit.
// Contents subject to change without notice.
// JSR-172 Reference Implementation wscompile 1.0, using:
JAX-RPC Standard Implementation (1.1, build R59)
```

```
package staticstub;

public class SayHelloResponse {
    protected java.lang.String result;

    public SayHelloResponse() {
    }

    public SayHelloResponse(java.lang.String result) {
        this.result = result;
    }

    public java.lang.String getResult() {
        return result;
    }

    public void setResult(java.lang.String result) {
        this.result = result;
    }
}
```



7.10 Java™ APIs for Bluetooth™ Wireless Technology and Object Push Protocol

7.10.1 Overview



This API is only available on this handset.

Certain iDEN handsets provide access to accessories and networking via Bluetooth. Several Bluetooth services are provided via Sun's dedicated Bluetooth API, JSR 82. This section covers iDEN's implementation of JSR 82 with emphasis on device specific limitations.

JSR 82 provides the following Bluetooth capabilities:

- Device Discovery
- Service Discovery
- Service Registration
- Generic Access Profile
- Serial Port Profile
- Logical Link Control and Adaptation Protocol (L2CAP)
- Object Exchange Protocol

In addition, iDEN handsets featuring Bluetooth also provide an OEM implementation of the Object Push Protocol.

7.10.2 Device and Service Discovery

This API allows developers to discover other Bluetooth devices and the services that they offer. MIDlets can initiate queries for devices and services and receive callbacks when either are found through the `DiscoveryListener` interface. This section does not describe how to use this API in detail. For sample code and detailed explanations of the API, see the JSR 82 specification.

7.10.2.1 Package Description

APIs for Bluetooth Service Discovery API are all located in `javax.bluetooth` package.

Interface Summary	
DiscoveryListener	Allows an application to receive device discovery and service discovery events.

**Class Summary**

DiscoveryAgent

Provides methods to perform device and service discovery.

7.10.2.2 Platform Specific Limitations

1. iDEN handsets can only perform one device or service inquiry at a time.
2. Once `populateRecord()` is called in `LocalDevice`, a device/service inquiry will throw an exception.
3. The maximum cache devices found in the previously performed inquiry is 10.
4. The trusted devices are always pre-known devices.
5. The maximum length of a service search response is 512 bytes. If the response length exceeds 512 bytes the registered `DiscoveryListener`'s `servicesDiscovered` method will be called with `SEARCH_SERVICE_ERROR` passed as the `transID`.
6. One device inquiry can find at most 10 devices.
7. Inquiry scanning and service searching are not allowed during a connection.
8. Inquiry and service search are not allowed during a connection.

7.10.3 Service Registration

The structure and use of service records is specified by the Bluetooth specification in the Service Discovery Protocol (SDP) document. Most of the Bluetooth Profile specifications also describe the structure of the service records used by the Bluetooth services that conform to the profile.

An SDP Server maintains a Service Discovery Database (SDDB) of service records that describe the services on the local device. Remote SDP clients can use the SDP to query an SDP server for any service records of interest. A service record provides sufficient information to allow an SDP client to connect to the Bluetooth service on the SDP server's device.

There might be many service attributes in a service record, and the SDP protocol makes it possible to specify the subset of the service attributes that an SDP client wants to retrieve from a remote service record. The `ServiceRecord` interface treats certain service attribute IDs as default IDs, and, if present, these service attributes are automatically retrieved during service searches.

The Service Record API defines a subset of the server responsibilities having to do with advertising a service to client devices. It mainly includes the following functions: Create a service record that describes the service offered by the application; Add a service record to the server's SDDB to make potential clients aware of this service; Update the service record in the server's SDDB if characteristics of the service change; Remove or disable the service record in the server's SDDB when the service is no longer available.

For additional details about the Service Record API, please refer to SDP document and the Bluetooth Profile specification.



7.10.3.1 Package Description

APIs for service registration are located in the `javax.bluetooth` package.

Interface Summary	
ServiceRecord	Implemented by stubs that describes characteristics of a Bluetooth service.
Class Summary	
DataElement	Defines the various data types that a Bluetooth service attribute value may have.
UUID	Defines universally unique identifiers.
LocalDevice	Defines the basic functions of the Bluetooth manager
Exception Summary	
ServiceRegistrationException	Thrown when there is a failure to add a service record to the local Service Discovery Database (SDDB) or to modify an existing service record in the SDDB.

7.10.3.2 Platform Specific Limitations

1. SDDB server supports at most 13 Java service records.
2. Memory allocated for a service record is limited by the Java heap.
3. When providing a service record, first set the discoverable mode of the device.
4. When a device/service inquiry is running, the `populateRecord()` method will always return false.

7.10.3.3 Sample Code

```

/** Describes this service name. */
private static final String SERVICE_NAME1 = new String("Service
Name1");
private static final String SERVICE_NAME2 = new String("Service
Name2");

/** Describes this server. */
private static final UUID SERVICE_UUID = new UUID(0x6789);

/** The attribute id of the record item with service names. */
private static final int SERVICE_ATTRIBUTE_ID = 0x2345;

/** Keeps the local device. */
private LocalDevice localDevice;

/** Accepts new connections. */
private StreamConnectionNotifier notifier;

```



```

/** Keeps the information about this service. */
private ServiceRecord record;

public void createServiceRecord()
{
    /*
     * Create Service Record
     */
    try {
        // get the local device
        localDevice = LocalDevice.getLocalDevice();

        // set device are discoverable
        if (!localDevice.setDiscoverable(DiscoveryAgent.GIAC)) {
            throw new IOException("Can't set discoverable
mode...");
        }

        // prepare a URL to create a notifier
        StringBuffer url = new StringBuffer("btspp://");

        // indicate this is a server
        url.append("localhost").append(':');

        // add the UUID to identify this service
        url.append(SERVICE_UUID.toString());

        // add the name for our service
        url.append(";name=Service1");

        // request all of the client to be authenticate
        url.append(";authenticate=true");

        // create notifier now
        notifier = (StreamConnectionNotifier) Connector.open(
            url.toString());

        // remember the service record for the later updates
        record = localDevice.getRecord(notifier);

        // create a special attribute with service names
        DataElement base = new DataElement(DataElement.DATSEQ);
        DataElement serviceName = new DataElement(DataElement.STRING, SERVICE_NAME1);
        base.addElement(serviceName);

        record.setAttributeValue(SERVICE_ATTRIBUTE_ID, base);

        // get the records Attribute ID array list
        attrIDs = record.getAttributeIDs();

        if ((attrIDs == null)) {

```




```

        System.out.println ("attribute ID array is null");
    }

    } catch (Exception e) {
        System.err.println("Can't initialize bluetooth: " + e);
    }
}

/**
 * Update Service Record
 */
public void updateServiceRecord()
{
    try {
        // get the record from service to update
        base = record.getAttributeValue(SERVICE_ATTRIBUTE_ID);

        // check the DataElement object is created already
        DataElement de = (DataElement)
dataElements.get(SERVICE_NAME1);

        // if no, then create a new DataElement
        de = new DataElement(DataElement.STRING, SERVICE_NAME2);

        base.addElement(de);
        record.setAttributeValue(IMAGES_NAMES_ATTRIBUTE_ID, base);
        localDevice.updateRecord(record);

    } catch (Exception e) {
        System.err.println("Can't Update service record: " + e);
    }
}

/**
 * Print Service Record
 */
public static void printServiceRecord( ServiceRecord r )
{
    // get the records Attribute ID array list
    int[] attrIDs = r.getAttributeIDs();

    System.out.println("Print          Service          Record          URL
"+r.getConnectionURL( ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false )
);

    for ( int i=0; i < attrIDs.length; i++ )
    {
        DataElement el = r.getAttributeValue(attrIDs [i] );
        printDataElement( el);
    }
}

```



```

/**
 * Print DataElement
 */
public static void printDataElement( DataElement e)
{
    Enumeration enum;
    if((de.getDataType() == DataElement.DATALT) || (de.getDataType()
== DataElement.DATSEQ))
    {
        System.out.println("it is DATALT or DATSEQ,size is " +
de.getSize());
        enum = (Enumeration)de.getValue();
        while(enum.hasMoreElements())
        {
            printDataElement((DataElement) enum.nextElement());
        }
    }
    else
    {
        switch(de.getDataType()){
            case DataElement.U_INT_4:
            case DataElement.U_INT_1:
            case DataElement.U_INT_2:
            case DataElement.INT_4:
            case DataElement.INT_1:
            case DataElement.INT_2:
            case DataElement.INT_8:
                System.out.println(de.getLong());
                break;
            case DataElement.U_INT_8:
            case DataElement.U_INT_16:
            case DataElement.INT_16:
                byte[] byteArray = (byte[])de.getValue();
                StringBuffer sbuffer = new StringBuffer();
                for(int count =0; count<byteArray.length;count++)
                {
                    sbuffer.append( (new
Byte(byteArray[count])).toString());
                }
                break;
            default:
                System.out.println(de.getValue().toString());
                break;
        }
    }
}
}
}

```



7.10.4 Generic Access Profile

Bluetooth's Generic Access Profile is made available by a small subset of classes defined by JSR 82. These classes are essential to the APIs described above: device discovery, service discovery, and service registration.

The `LocalDevice` class, one of the defining classes of GAP, has been described above. This section will describe its counterpart, `RemoteDevice`.

For more detailed information on the classes that implement the GAP, see the JSR 82 specification.

7.10.4.1 Package Description

APIs for GAP are located in the `javax.bluetooth` package.

Class Summary	
---------------	--

<code>RemoteDevice</code>	Represents a remote Bluetooth device.
---------------------------	---------------------------------------

7.10.4.2 Platform Specific Limitations

1. The `authorize()` method is not supported currently. However, if the device is trusted, `authorize()` will return `true` or `false`.
2. Before the `getFriendlyName()` method is called, set the discoverable mode of the device.

7.10.4.3 Sample Code

```
try {  
  
    // Retrieve the connection string to connect to  
    // the server  
    LocalDevice local =  
        LocalDevice.getLocalDevice();  
  
    DiscoveryAgent agent = local.getDiscoveryAgent();  
  
    String connString = agent.selectService(  
        new UUID("86b4d249fb8844d6a756ec265dd1f6a3", false),  
        ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);  
  
    if (connString != null) {  
        try {  
            // Connect to the server and send 'Hello, World'  
            StreamConnection conn = (StreamConnection)  
                Connector.open(connString);  
        }  
    }  
}
```



```

        /* Get remote device */
        RemoteDevice remDev =
RemoteDevice.getRemoteDevice(conn);
        /* Get remote device addr */
        String remoteAdd = remDev.getBluetoothAddress();
        /* Get remote device name */
        String deviceName = remDev.getFriendlyName(true);
        /* If the remote device trusted */
        if(!remDev.isTrustedDevice()) {
            System.out.println("Remote Device:" + remoteAdd +
                " not is trusted.");
        }
        else
            System.out.println("Remote Device:" + remoteAdd +
                " is trusted.");
        /* If the remote device Authenticated */
        if (!remDev.isAuthenticated()) {
            System.out.println("Remote Device:" + remoteAdd +
                " is not Authenticated.");
            /* The connection to remDev is not currently
            * Authenticated, so turn on Authenticated.
            */
            if (!remDev.authenticate()) {
                System.out.println("Remote Device:" + remoteAdd
+" set Authenticated failed");
            }
            else {
                System.out.println("Remote Device:" + remoteAdd
+" set Authenticated sucessfully.");
            }
        }

        /* If the remote device Authorized */
        if (!remDev.isAuthorized(conn)) {
            System.out.println("Remote Device:" + remoteAdd +
                " is not Authorized.");

            /* The connection to remDev is not currently
            * authorize, so turn on authorize.
            */
            if (!remDev.authorize(conn)) {
                System.out.println("Remote Device:" + remoteAdd +
                    " set Authorized failed");
            }
            System.out.println("Remote Device:" + remoteAdd +
                " set Authorized successfully.");
        }

    } catch (IOException e) {
        System.err.println("Can't initialize bluetooth: " + e);
    }

```



7.10.5 Serial Port Profile

In the JSR 82 specification, the Serial Port Profile is defined to establish an RFCOMM connection and supports wireless data communication between Bluetooth devices by providing a stream-based Java API to the RFCOMM connection. No new methods or classes are introduced for the RFCOMM API. Instead, the API reuses existing classes and interfaces from the Generic Connection Framework (GCF) in CLDC.

For more detailed information on the classes that implement the GAP, see the JSR 82 specification.

7.10.5.1 Package Description

Refer to the `javax.microedition.io` package for detailed explanations of the classes that define J2ME's GCF.

7.10.5.2 Using `javax.microedition.io.Connector` For RFCOMM

All RFCOMM connections are initiated with `javax.microedition.io.Connector.open (String name)` where *name* is a valid USL of form `{scheme}:{target}{params}`

{scheme} is "btspp" (RFCOMM link only)

{target} is the networking address starting with "/"

{params} are formed as series of equates of the form ";encrypt=false"

To open a server connection, the target specified is localhost concatenated with the UUID.

example: `localhost:102030405060740A1B1C1D1E100`

To open a client connection the target specified is the server address concatenated with the server channel identifier.

example: `FFFF7777FFFF:7`



The following table describes valid parameter strings for RFCOMM.

Name	Description	Values	Client or Server
authenticate	Remote device must be authenticated	true, false	both
authorize	All connections to this device must receive authorization	true, false	server
encrypt	Link must be encrypted	true, false	both
master	This device must be the Master	true, false	both
name	Service-Name attribute in service record	Valid string	server

example: *name=SPP;encrypt=true;authenticate=true;authorize=true*

7.10.5.3 Sample Code

The following is a client and server code example of JSR-82 RFCOMM API:

```
import java.io.*;
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import javax.bluetooth.*;

/* This is RFCOMM client test code */
public final class SPP5Client extends MIDlet implements CommandListener,
Runnable
{
    /** Soft button for exiting the demo. */
    private boolean SIMPLE_TEST = false;

    private final Command EXIT_CMD = new Command("Exit", Command.EXIT, 2);

    /** Soft button for launching a client or sever. */
    private final Command OK_CMD = new Command("Start", Command.SCREEN,
1);

    /** fix server Bluetooth address and channel id */
    private TextField addrTF = new TextField("Destination",
"FFFF7777FFFF:7", 14, TextField.ANY);

    private StringItem strItem = new StringItem(null, "");

    /** A menu list instance */
    private final Form menu = new Form("SPP Client ");

    /** data strings: 4 X 78 characters */
    private String input_data = " (1) Hello Hello World! This is the first
RFCOMM message I sent out. Do you get it? (2) Hello Hello World! This is
the first RFCOMM message I sent out. Do you get it? (3) Hello Hello World!
```



This is the first RFCOMM message I sent out. Do you get it? (4) Hello
Hello World! This is the first RFCOMM message I sent out. Do you get it?";

```
private boolean stop = false;

private StreamConnection conn;

int k = 0;

/**
 * Constructs main screen of the MIDlet.
 */
public SPP5Client()
{
    menu.addCommand(OK_CMD);
    menu.append(strItem);
    menu.append(addrTF);
    menu.setCommandListener(this);
}

/**
 * Creates the demo view and action buttons.
 */
public void startApp() {
    Display.getDisplay(this).setCurrent(menu);
}

/**
 * Destroys the application.
 */
protected void destroyApp(boolean unconditional) {
}

/**
 * Does nothing. Redefinition is required by MIDlet class.
 */
protected void pauseApp() {}

/**
 * Responds to commands issued on "client or server" form.
 *
 * @param c command object source of action
 * @param d screen object containing actioned item
 */
public void commandAction(Command c, Displayable d)
{
    if (c == EXIT_CMD)
    {
        System.out.println("Send " + k + " Bytes");
        destroyApp(true);
        notifyDestroyed();
        return;
    }
}
```



```
        else if (c == OK_CMD)
        {
            new Thread(this).start();
        }
    }

    /** Shows main menu of MIDlet on the screen. */
    public void run()
    {
        int i = 0;
        int j = 0;
        String result = "";

        while(true)
        {
            try
            {
                StringBuffer url = new StringBuffer("btspp://");

                url.append(addrTF.getString());

                strItem.setText("url:" + url);
                System.out.println("url:" + url);

                /* open RFCOMM connection from client side */
                conn = (StreamConnection)
                    Connector.open(url.toString(),
                        Connector.READ_WRITE, true);

                strItem.setText("Connected");

                System.out.println("(1) Connected");

                OutputStream out = conn.openOutputStream();
                InputStream in = conn.openInputStream();

                /* send data to server */
                out.write(input_data.getBytes());
                out.write(input_data.getBytes());

                System.out.println("(2) Write");

                out.flush();

                System.out.println("(3) flush");

                Thread.sleep(10000);

                System.out.println("(4) sleep");

                out.close();
                in.close();
            }
        }
    }
}
```




```

        catch (Exception e)
        {
            System.out.println("Exception " + e.toString());
            strItem.setText(e.toString());
        }

        try {
            System.out.println("(5) close");
            conn.close();
        }
        catch (Exception e)
        {
            System.out.println("Exception " + e.toString());
            strItem.setText(e.toString());
        }

        try {
            Thread.sleep(5000);
        }
        catch (Exception e)
        {}
    }
}

} // end of class

```

```

import java.io.*;
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import javax.bluetooth.*;

/*    This is RFCOMM server test code */
public final class SPP5Server extends MIDlet implements CommandListener,
Runnable {

    /** Soft button for exiting the demo. */
    private boolean SIMPLE_TEST = false;

    private final Command EXIT_CMD = new Command("Exit", Command.EXIT, 2);

    /** Soft button for launching a client or sever. */
    private final Command OK_CMD = new Command("Start", Command.SCREEN,
1);

    private StringItem strItem = new StringItem(null, "");

    /** A menu list instance */

```



```
private final Form menu = new Form("SPP Server ");

int i = 0;
int j = 0;
String result = "";

int buffersize = 201, counter = 0, ch = -5, timeout = 0;

private byte[] buffer = new byte[1500];
private int count = 0;
private StringBuffer url;
private LocalDevice localDevice;

private StreamConnectionNotifier notifier = null;

/**
 * Constructs main screen of the MIDlet.
 */
public SPP5Server() {
    menu.addCommand(OK_CMD);
    menu.append(strItem);
    menu.setCommandListener(this);
}

/**
 * Creates the demo view and action buttons.
 */
public void startApp() {
    Display.getDisplay(this).setCurrent(menu);
}

/**
 * Destroys the application.
 */
protected void destroyApp(boolean unconditional) {
}

/**
 * Does nothing. Redefinition is required by MIDlet class.
 */
protected void pauseApp() {}

/**
 * Responds to commands issued on "client or server" form.
 *
 * @param c command object source of action
 * @param d screen object containing actioned item
 */
public void commandAction(Command c, Displayable d)
{
    if (c == EXIT_CMD)
    {
        System.out.println(result + "\nReceived " + i + " Bytes\nLast
```



```
one is" + j);
        destroyApp(true);
        notifyDestroyed();
        return;
    }
    else if (c == OK_CMD)
    {
        new Thread(this).start();
    }
}

/** Shows main menu of MIDlet on the screen. */
public void run() {

//    (1) start server
    try
    {
        localDevice = LocalDevice.getLocalDevice();

        if (!localDevice.setDiscoverable(DiscoveryAgent.GIAC))
        {
            throw new IOException("Can't set discoverable mode...");
        }

        url = new StringBuffer("btspp://localhost:");

        // add the UUID to identify this service
        url.append(new UUID(0x1101).toString());

        // add the name for the service
        url.append(";name=SPP Server");

        strItem.setText("url:" + url);
        System.out.println("url:" + url);

        notifier = (StreamConnectionNotifier)
            Connector.open(url.toString(),
Connector.READ_WRITE, true);

    }
    catch (Exception e)
    {
        System.out.println("Can't open notifier" + e.toString());
    }

//    (2) start connection object
    while (true)
    {
        try {
            strItem.setText("url:" + url + "\nAddress:" +
localDevice.getBluetoothAddress()
                + "\nWaiting...");
        }
    }
}
```



```

System.out.println(localDevice.getBluetoothAddress());
    System.out.println("Waiting...");
    StreamConnection conn = notifier.acceptAndOpen();

    strItem.setText("Connected");
    System.out.println("Connected");

    OutputStream out = conn.openOutputStream();
    System.out.println("openOutputStream");

    InputStream in = conn.openInputStream();
    System.out.println("openInputStream");

    while (true)
    {
        counter = 0;
        timeout = 0;
        ch = 0;

        System.out.println("Reading...");

        try {
            while (counter <= 620)
            {
                ch = in.read();
                System.out.print((char)ch);
                counter++;
                if (ch == -1) break;
            }
            System.out.println("\n");

        } catch (Exception e) {
            System.out.println("\n");
            System.out.println("Server: Error while" + "
available()/read() " + e);
            break;
        }

        System.out.println(" counter = " + counter + "
timeout = " + timeout);

        /* just got 620 characters from client */
        if ( counter >= 620 || ch == -1 ) break;
    }

    System.out.println("close");
    in.close();
    out.close();
    conn.close();
    System.out.println("pass close");
}
catch (Exception e)
{

```



```

        System.out.println("Exception in conn object" +
e.toString());
    }

    }

}

} // end of class

```

7.10.6 Logical Link Control and Adaptation Protocol (L2CAP)

The Bluetooth L2CAP API allows a MIDlet to open a connection and send or receive data over the Logical Link Control and Adaptation Protocol.

For more detailed information on the classes that implement the L2CAP, see the JSR 82 specification.

7.10.6.1 Package Description

APIs for L2CAP are located in the `javax.bluetooth` package.

Interface Summary

L2CAPConnection	Represents a connection-oriented L2CAP channel
L2CAPConnectionNotifier	Provides an L2CAP connection notifier.

7.10.6.2 Using `javax.microedition.io.Connector` For L2CAP

All L2CAP connections are initiated with `javax.microedition.io.Connector` methods. URLs must start with “`btl2cap://`”, and contain a Bluetooth address plus a psm value to open a client connection, or “`localhost`” plus a UUID to open a server connection.

7.10.6.3 Platform Specific Limitations

1. When opening a connection, the `ReceiveMTU` and `TransmitMTU` parameters specified in the name must not be less than 48, and must not be greater than 672. Otherwise an `IllegalArgumentException` will be thrown.
2. The Bluetooth implementation on iDEN handsets supports only one connection at a time. There can be several server connections opened waiting for remote connection requests, but only one connection between two devices can be created successfully at any time.
3. Bluetooth L2CAP connections only support timeout when receiving data.



4. The following Connector open methods do not support L2CAP connections:

```
openDataInputStream
openDataOutputStream
openInputStream
openOutputStream
```

5. L2CAP connections do not provide flow control. When sending data, it is the responsibility of upper application to do flow control. Otherwise some data may be lost when transferring large blocks of data. The data transfer speed of L2CAP is about 36,000 bits per second.

6. When upper application closes a connection immediately after sending some data, the link between the two devices is broken. According to JSR 82, it is illegal to read data from a closed connection. So another device may not be able to receive the last block of data. To avoid this problem, flow control must be implemented in the upper applications.

7.10.7 Object Exchange Protocol

The OBEX protocol provides a high-level API for transferring data between hosts. JSR 82 provides an implementation of the OBEX protocol over Bluetooth. In iDEN handsets OBEX resides on top of two reliable transports, TCP/IP stream sockets and Bluetooth RFCOMM connection.

For more detailed information on the classes that implement the OBEX, see the JSR 82 specification.

7.10.7.1 Package Description

While defined by JSR 82, the OBEX API is located in the `javax.obex` package.

Class	
PasswordAuthentication	This class holds user name and password combinations.
ResponseCodes	The <code>ResponseCodes</code> class contains the list of valid response codes a server may send to a client.
ServerRequestHandler	The <code>ServerRequestHandler</code> class defines an event listener that will respond to OBEX requests made to the server.
Interface	
Authenticator	This interface provides a way to respond to authentication challenge and authentication response headers.
ClientSession	The <code>ClientSession</code> interface provides methods for OBEX requests.
HeaderSet	The <code>HeaderSet</code> interface defines the methods that set and get the values of OBEX headers.
Operation	The <code>Operation</code> interface provides ways to manipulate a



	single OBEX PUT or GET operation.
SessionNotifier	The <code>SessionNotifier</code> interface defines a connection notifier for server-side OBEX connections.

7.10.7.2 Using `javax.microedition.io.Connector` For OBEX

To create an OBEX client or server connection object, the application uses the GCF, following the same format as other connection strings in that framework:

```
{protocol}:{target} [{params}]
```

The definition of {protocol}, {target}, and {params} depends on the transport layer that OBEX uses. In general, {protocol} is defined to be {transport}obex, but OBEX over RFCOMM is an exception to this rule and is discussed next.

These protocols should be implemented based on the actual transport mechanisms available on the device. In our device, the supported protocol is “btgoep” and “tcpobex”. Calling `Connector.open()` on an unsupported transport protocol throws a `ConnectionNotFoundException`.

7.10.7.2.1 OBEX Over RFCOMM

The {protocol} for OBEX over RFCOMM is defined as `btgoep` because this is the implementation of the Generic Object Exchange Profile (GOEP) defined by the Bluetooth SIG. The {target} for client connections is the Bluetooth address and channel identifier of the device that the client wishes to connect to, separated by a colon (for example, `0050C000321B:4`). The {target} for a server always is `localhost` followed by a colon and the service class UUID. The valid {params} for OBEX over RFCOMM are `authenticate`, `encrypt`, `authorize`, and `master`. The default value for all of these {params} is “false”. The only other valid value is “true”.

The following is a valid client connection string for OBEX over RFCOMM:

```
btgoep://0050C000321B:12
```

The following is a valid server connection string for OBEX over RFCOMM:

```
btgoep://localhost:12AF51A9030C4B2937407F8C9ECB238A
```



When an application passes a valid OBEX over RFCOMM server connection string to `Connector.open()`, a Bluetooth service record is created. These attributes will be added by the device automatically: `ServiceRecordHandle` (generated and added by SDDb), `ServiceClassIDList` (the UUID retrieved from Server URL), `ProtocolDescriptorList` (added by our implementation), and `ServiceName` (retrieved from the Server URL parameter).

7.10.7.2.2 OBEX Over TCP/IP

If OBEX uses TCP/IP as its transport protocol, the `{protocol}` is `tcpobex`. For an OBEX client, the `{target}` is the IP address of the server followed by a colon and port number. (for example, `12.34.56.100:5005`). If no port number is specified, port number 650 is used (this is the port number reserved for OBEX by IANA, the Internet Assigned Numbers Authority). A server's `{target}` is a colon followed by the port number (for example, `:5005`). If no port number is given, port number 650 is opened by default. There are no valid `{params}` for OBEX over TCP/IP.

The following are valid client connection strings for OBEX over TCP/IP:

```
tcpobex://132.53.12.154:5005
```

```
tcpobex://132.53.12.154
```

The first string creates a client that connects to port 5005. The second string creates a client that connects to port 650.

The following are valid server connection strings for OBEX over TCP/IP:

```
tcpobex://:5005
```

```
tcpobex://
```

The first string creates a server that listens on port 5005. The second string creates a server that listens on port 650.

7.10.7.3 Creating An OBEX Client

To create a client connection for OBEX, the client application uses the appropriate string and passes this string to `Connector.open()`. `Connector.open()` returns a `javax.obex.ClientSession` object.

To establish an OBEX connection, the client creates a `javax.obex.HeaderSet` object using the `createHeaderSet()` method in the `ClientSession` interface. Using the `HeaderSet` object, the client can specify header values for the `CONNECT` request. An OBEX `CONNECT` packet also contains the OBEX version number, flags, and maximum packet length, which are maintained by the implementation. To complete a `CONNECT` request, the client supplies the `HeaderSet` object to the `connect()` method in the `ClientSession` interface. After the `CONNECT` request finishes, the OBEX headers received from the server are returned to the application. If no header object is provided as an input parameter, a `javax.obex.HeaderSet` object still is returned from the `connect()` method. To determine whether or not the request succeeded, the client calls the `getResponseCode()` method in the `HeaderSet` interface. This method returns the response code sent by the server, defined in the `javax.obex.ResponseCodes` class.

A `DISCONNECT` request is completed in the same way as a `CONNECT` request except that the `disconnect()` method is called instead of `connect()`. If the `javax.obex.HeaderSet` object contains more headers than can fit in one OBEX packet, a `java.io.IOException` is thrown.



To complete a SETPATH operation, the client calls the `setPath()` method in the `ClientSession` object. To specify the name of the target directory, set the name header to the desired target by calling `setHeader()` on the `HeaderSet` provided to `setPath()`. The client also may specify whether or not the server should back up one directory level before applying the name and whether or not the server should create the directory if it does not already exist. If the header is too large to send in one OBEX packet, a `java.io.IOException` is thrown.

To complete a PUT or GET operation, the client creates a `javax.obex.HeaderSet` object with `createHeaderSet()`. After specifying the header values, the client calls the `put()` or `get()` method in the `javax.obex.ClientSession` object. The implementation sends the headers to the server and receives the reply. The `put()` and `get()` methods return the `javax.obex.Operation` object. With this object, the client can determine whether or not the request succeeded. If the request succeeded, the client may put or get a data object using output or input streams, respectively. When the client is finished, the appropriate stream should be closed. To ABORT a PUT or GET request, the client calls the `abort()` method in the `javax.obex.Operation` object. The `abort()` method closes all input and output streams and ends the operation by calling the `close()` method on the `Operation` object.

7.10.7.4 Creating An OBEX Server

To create a server connection, the server provides a string to `Connector.open()`. `Connector.open()` returns a `javax.obex.SessionNotifier` object. The `SessionNotifier` object waits for a client to create a transport layer connection by calling `acceptAndOpen()`. A single server may serve multiple clients by calling `acceptAndOpen()` multiple times. The `acceptAndOpen()` method returns a `javax.obex.Connection` object. This object represents a connection to a single client. The server specifies the request handler that will respond to OBEX requests from the client by passing the `javax.obex.ServerRequestHandler` object to `acceptAndOpen()`.

The server must create a new class that extends the `javax.obex.ServerRequestHandler` class. The server needs to implement only those methods for the OBEX requests that it supports. For example, if the server does not support SETPATH requests, it need not override the `onSetPath()` method. As requests are received, the appropriate methods are called and the server processes the requests. When the server is finished, it must return the appropriate final response code defined in the `javax.obex.ResponseCodes` class.

Server applications should not call the `abort()` method; if a server applications calls `abort()` the `javax.obex.Operation` argument that is part of the `onGet()` and `onPut()` methods throws a `java.io.IOException`.

If the server implementation is not able to pass all the headers that are specified by the server application in a reply, then the server implementation returns an `OBEX_HTTP_REQ_TOO_LARGE`. If the server application returns a response code that is not defined in the `javax.obex.ResponseCodes` class, then the server implementation sends an `OBEX_HTTP_INTERNAL_ERROR` response to the client.



7.10.7.5 Platform Specific Limitations

For both the Object Exchange over RFCOMM and TCP/IP, only one connection at a time is supported. That means one server can communicate with one client at one time. Additionally, for for Object Exchange over TCP/IP, MIDlets must handle SessionNotifier one by one. For example, if a MIDlet calls acceptAndOpen in a loop, once a client connects to the server, the server should create a thread to handle that. When this thread closes the session, the acceptAndOpen will throw an IOException. It is recommended that MIDlets handle SessionNotifier one by one instead of in multiple thread.

7.10.8 Object Push Protocol

The OPP API allows a MIDlet to exchange data with another Bluetooth device. This package defines classes and interfaces for both clients and servers.

The main features of the OPP API are:

1. With a single command, it's possible to establish a connection, transfer data and disconnect.
2. OPP handles segmentation and reassembly of OBEX packages automatically, as needed.
3. Can act both as a client and a server, limited to only one session at a time.

In the following section the concept "session" is used. From the OPP API's point of view, a session is started when a client requests to push or pull a new object. The session is completed when the last confirmation for that object is received, regardless how many requests and confirms have been sent in between.

Detailed information on OBEX response codes is outside the scope of this document. For more information, see Infrared Data Association (IrDA) Object Exchange Protocol OBEX version 1.3.3 at <http://www.irda.org>.

7.10.8.1 Package Description

The OPP API is located in the com.motorola.iden.bluetooth.opp package.

Class	
OppClient	Provides APIs for OPP clients.
OppObject	Defines a structure for data transferred in OPP sessions.
OppServer	Provides APIs for OPP servers.
Interface	
OppClientRequestHandler	The OppClientRequestHandler interface defines an event listener that will respond to OPP response made to the client.
OppObjectFormat	The interface defines the formats be used in OPP.
OppResponseCodes	Defines result codes used in OPP sessions.
OppServerRequestHandler	Defines an event listener for responding to OPP requests made to a server.



7.10.8.1.1 Class OppClient

This class defines the APIs for a J2ME OPP client. Requests are sent via the APIs defined here.

```
7.10.8.1.1.1 public void openClient(OppClientRequestHandler  
    reqHandle) throws IllegalArgumentException,  
    IOException
```

This method initializes the OPP client. Push and pull calls will throw an IOException if the client has not been opened.

Calling openClient with a null reqHandle will cause an IllegalArgumentException to be thrown. An IOException will be thrown if the Bluetooth stack can not be opened successfully, for example if an OPP client is already open.

```
7.10.8.1.1.2 public boolean close(boolean friendlyClose) throws  
    InterruptedException, IOException
```

This method will be called to close the current OPP connection.

This method will return false and leave the connection open if friendlyClose is set to true and the connection has pending pushes or pulls.

This method will return true and close the connection if friendlyClose is set to true and the connection has no pending pushes or pulls or if friendlyClose is false. Note that any pending pushes or pulls may not complete if this method is called with friendlyClose set to false.

```
7.10.8.1.1.3 public void pushReq(byte[] serverAddr, OppObject  
    oppObject, int totalLen, boolean isLastSession) throws  
    IllegalArgumentException, IOException
```

This method pushes an object to a specified server.

If the size of serverAddr is not 6 bytes an IllegalArgumentException will be thrown.

If isLastSession is true the connection will be closed automatically after the current push operation finished. Clients setting isLastSession to true should not call disconnectReq(). Calling disconnectReq() when isLastSession is true will cause an IOException to be thrown. It is recommend to set isLastSession to true in the last pushReq() of an object.

An object can be divided and sent in several PUSH operations. In this case, do not need to fill the same NAME&DESCRIPTION in the following fragment.



```
7.10.8.1.1.4 public void pullReq(byte[] serverAddr, boolean  
isLastSession) throws IllegalArgumentException,  
IOException
```

Try to pull vCard data from the specified server.

If size of serverAddr is not 6 bytes an IllegalArgumentException will be thrown.

If isLastSession is true the connection will be closed automatically after the current pull operation finishes. Clients setting isLastSession to true do not need to call disconnectReq(). Calling disconnectReq() when isLastSession is true will cause an IOException to be thrown. If one object should be pulled in several pull request, only the "isLastSession" of first pull request can be accepted.

```
7.10.8.1.1.5 public void abortReq(String description) throws  
IOException, IllegalArgumentException
```

Send out an abort request for the current transfer operation.

The description parameter is optional and can be null.

The abortReq() method can only be called while data is being transferred, otherwise an IOException will be thrown and the abort request will not be sent out.

```
7.10.8.1.1.6 public void disconnectReq()throws IOException
```

Closes the connection.

If the isLastSession parameter in a call to pushReq() or pullReq was set to true this method should not be called, otherwise an IOException will be thrown and the disconnect request will not be sent out.

This method can't be called while data is being transferred, otherwise an IOException will be thrown and the disconnect request will not be sent out.

```
7.10.8.1.2      Class OppClientRequestHandler
```

The OppClientRequestHandler interface defines a set of callback methods for OPP related events that a typical OPP client would be interested in.

```
7.10.8.1.2.1 public void onConnectInd()
```

This method defines a callback for client connections.



7.10.8.1.2.2 `public void onDisconnectInd(int oppResult)`

This method defines a callback for client disconnections made automatically via a push or pull request whose `isLastSession` parameter was `true`.

`oppResult` is the result of the disconnect. Possible values are

- `OPP_OK`
- `OPP_PADAPT_FAILED`

7.10.8.1.2.3 `public void onDisconnectCfm(int oppResult)`

This method defines a callback for client disconnections as a result of a disconnect request.

`oppResult` is the result of the disconnect. Possible values are

- `OPP_OK`
- `OPP_PADAPT_FAILED`

7.10.8.1.2.4 `public void onPushCfm(int oppResult, byte
responseCode, String description)`

This method defines a callback indicating a server response to a push request.

The `oppResult` parameter holds the result of the push session. Positive results are:

- `OPP_OK`
- `OPP_CONTINUE`

Negative results are:

- `OPP_PADAPT_FAILED`
- `OPP_SDAP_FAILED`
- `OPP_OBEX_FAILED`
- `OPP_OBEX_HEADER_FAILED`
- `OPP_NO_MATCHING_SERVICE`
- `OPP_CALL_ERROR`

If the result is `OPP_OBEX_FAILED`, the `responseCode` parameter will contain one of the OBEX response codes below specifying the error:

- `OBEX_NOT_ACCEPTABLE`
- `OBEX_UNSUPPORTED_MEDIA_TYPE`
- `OBEX_SERVICE_UNAVAILABLE`
- `OBEX_REQUEST_TIME_OUT`



```
7.10.8.1.2.5 public void onPullCfm(int oppResult, byte  
                                responseCode, OppObject obj, int totalLen)
```

This method defines a callback indicating the result of a pull from a server.

The oppResult parameter holds the result of the pull session. Positive results are

OPP_OK	This is the final package of data.
OPP_CONTINUE	The server has more data to send. Use another pullReq() to request more.

Negative results are

- OPP_PADAPT_FAILED
- OPP_SDAP_FAILED
- OPP_OBEX_FAILED
- OPP_OBEX_HEADER_FAILED
- OPP_NO_MATCHING_SERVICE
- OPP_CALL_ERROR

If the result is OPP_OBEX_FAILED, the responseCode parameter will contain one of the OBEX response codes below specifying the error:

- OBEX_NOT_ACCEPTABLE
- OBEX_UNSUPPORTED_MEDIA_TYPE
- OBEX_SERVICE_UNAVAILABLE
- OBEX_REQUEST_TIME_OUT



```
7.10.8.1.2.6 public void onAbortCfm(int oppResult, byte  
                      responseCode, String description)
```

This method defines a callback indicating confirmation of an abort request.

The OppResult parameter holds the result of the abort request. Positive results are:

- OPP_OK

Negative results are:

- OPP_OBEX_FAILED
- OPP_OBEX_HEADER_FAILED
- OPP_CALL_ERROR

If the result is OPP_OBEX_FAILED, the client application should terminate the OBEX connection. If the result is OPP_OBEX_FAILED the responseCode parameter will contain the OBEX response code specifying the error.

The OBEX response code. Notable negative response code is

- OBEX_REQUEST_TIME_OUT



7.10.8.1.3 Class OppServer

This interface defines the APIs for a J2ME OPP server. Responses from servers to clients should be sent out via these APIs.

```
7.10.8.1.3.1 public void openServer(OppServerRequestHandler
                                handle, byte[] serviceName, byte incomingSecLevel)
                                throws IOException, IllegalArgumentException
```

This method allows an OPP server to be opened. The service record will be registered enabling clients to find the server. The openServer method behaves differently from the openClient method in that the client is opened immediately after the method returns. However, the server is opened after onOpenCfm() in handle is called.

If handle is null an IllegalArgumentException will be thrown. An IOException will be thrown if the OPP server can not be opened successfully, for example if there is already an open server or the Bluetooth device is busy.

The serviceName parameter is optional. If serviceName is null the default service name value is set to "Java-OBEX-Object-Push". The maximum length of the string is 100.

incomingSecLevel is a bit map field. Set the following bits as needed:

- Bit 0: SEC_INCOMING_AUTHENTICATION_REQUIRED
- Bit 1: SEC_INCOMING_AUTHORIZATION_REQUIRED
- Bit 2 SEC_INCOMING_ENCRYPTION_REQUIRED
- Bit 6 SEC_INCOMING_CONNECTIONLESS_ALLOWED

```
7.10.8.1.3.2 public boolean close(boolean friendlyClose) throws
                InterruptedException, IOException
```

This method closes the current OPP connection.

This method will return false and leave the connection open if friendlyClose is set to true and the connection has pending pushes or pulls.

This method will return true and close the connection if friendlyClose is set to true and the connection has no pending pushes or pulls or if friendlyClose is false. Note that any pending pushes or pulls may not complete if this method is called with friendlyClose set to false.



```
7.10.8.1.3.3 public void pushRsp(byte responseCode, String  
                descriptionString) throws IOException,  
                IllegalArgumentException
```

This method sends a response to a client.

pushRsp can only be called once there is an incoming push indication.

The descriptionString parameter is optional. The maximum length is 100.

The responseCode parameter is the OBEX response code. Positive responses are:

- OBEX_SUCCESS

Shall be used if the isLastMessage parameter in onPushInd() was equal to true.

- OBEX_CONTINUE

Shall be used if the isLastMessage parameter in onPush was equal to false.

Any other response code is a negative response. Notable responses are:

- OBEX_NOT_ACCEPTABLE
- OBEX_UNSUPPORTED_MEDIA_TYPE
- OBEX_SERVICE_UNAVAILABLE



```
7.10.8.1.3.4 public void pullRsp(byte responseCode, oppObject obj,  
    int totalLength) throws IOException,  
    IllegalArgumentException
```

Try to respond to the client which sent the pull request.

This method can only be called once there is an incoming pull indication.

The response code is specified with the responseCode parameter. Positive responses are:

- OBEX_SUCCESS

Shall be used if this message is the last one from OPP's point of view. For example, the oppObject body field is pointing to a buffer which holds the complete object.

- OBEX_CONTINUE

Shall be used if this message isn't the last one from OPP's point of view. For example, the oppObject body field is pointing to a buffer which doesn't hold the complete object.

Any other response code is a negative response. Notable negative responses are:

- OBEX_NOT_ACCEPTABLE
- OBEX_UNSUPPORTED_MEDIA_TYPE
- OBEX_SERVICE_UNAVAILABLE

```
7.10.8.1.3.5 public void abortRsp(byte responseCode, String  
    description) throws IOException,  
    IllegalArgumentException
```

Sends an abort response.

This method can only be called once there is an incoming abort indication.

The response code is specified with the rspCode parameter. The only positive responses is OBEX_SUCCESS. Any other response code is a negative response which causes the client to disconnect.

```
7.10.8.1.3.6 public void changeSecLevelReq(byte outgoingSecLevel)  
    throws IOException, IllegalArgumentException
```

This method provides the application layer with the ability to modify the outgoing OPP security level. The method is intended to be invoked from within the context of the registered OPP server application layer task.

If a call to changeSecLevel is made while another call is pending an IOException will be thrown.



7.10.8.1.4 Class OppServerRequestHandler

The OppServerRequestHandler interface defines an event listener that will respond to OPP requests made to the server.

A J2ME MIDlet acting as an OPP server should implement this interface.

7.10.8.1.4.1 `public void onConnectInd()`

This method defines a callback for server connections.

7.10.8.1.4.2 `public void onDisconnectInd(int oppResult)`

This method defines a callback for server disconnections.

oppResult is the result of the disconnect. Possible values are

- OPP_OK
- OPP_PADAPT_FAILED

7.10.8.1.4.3 `public void onPushInd(byte[] clientAddr, OppObject oppObject, int totalLength, boolean isLastFragment)`

This method defines a callback for client object pushes to the server.

The isLastFragment parameter indicates whether it is the last fragment being sent.

The object being pushed by the client is held in oppObject. This object may be null.

7.10.8.1.4.4 `public void onPullInd(byte[] clientAddr)`

This method defines a callback for client pull requests.

7.10.8.1.4.5 `public void onAbortInd(String description)`

This method defines a callback for operation aborts.

7.10.8.1.4.6 `public void onPullCompleteInd()`

This method defines a callback for the completion of an object pull from the client. The object being pulled by the client has been sent when this callback is made.



7.10.8.1.5 Class OppObject

This class represents an OPP object. One object may be split into several OppObject instances.

7.10.8.2 Sample Code

```
/**
 * OPP Client
 */
import java.io.*;
import java.lang.*;
import com.motorola.iden.bluetooth.opp.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.obex.ResponseCodes;

public class ClientTest extends MIDlet implements
CommandListener{
    Display display;
    List mainList;

    public OppClient OppImpl;
    public byte[] serverAddr =
{0x07, (byte)0xaa, (byte)0xff, 0x55, 0x0c, 0x00}; //bluetooth address
of PC

    public OppClientListener oppListener;
    public OppObject pushObj;
    public byte[] pushBody = {0x42 ,0x45 ,0x47 ,0x49 ,0x4e
,0x3a ,0x56 ,0x43 ,0x41 ,0x52 ,0x44 ,0x0d ,0x01 ,0x56 ,0x45
,0x52,
                                0x53 ,0x49 ,0x4f ,0x4e ,0x3a
,0x32 ,0x2e ,0x31 ,0x0d ,0x0a ,0x4e ,0x3a ,0x5a ,0x68 ,0x61
,0x6e,
                                0x67 ,0x3b ,0x43 ,0x69 ,0x6e
,0x0d ,0x0a ,0x45 ,0x4d ,0x41 ,0x49 ,0x4c ,0x3b ,0x49 ,0x4e
,0x54,
```



```

                                0x45 ,0x52 ,0x4e ,0x45 ,0x54
,0x3a ,0x64 ,0x61 ,0x69 ,0x73 ,0x79 ,0x7a ,0x40 ,0x6e ,0x6a
,0x2e,

                                0x73 ,0x63 ,0x2e ,0x6d ,0x63
,0x65 ,0x6c ,0x2e ,0x6d ,0x6f ,0x74 ,0x2e ,0x63 ,0x6f ,0x6d
,0x0d,

                                0x0a ,0x45 ,0x4e ,0x44 ,0x3a
,0x56 ,0x43 ,0x41 ,0x52 ,0x44 ,0x0d ,0x0a};

    public int bodyLen = 0x5c;

    private final static Command CMD_SEL = new
Command("Select",Command.SCREEN,1);

    private final static Command CMD_EXIT = new
Command("Exit",Command.SCREEN,1);

    private final static Command CMD_BACK = new
Command("Back",Command.SCREEN,1);

    private firstOpenFlag = true;


    /**
     * object push form
     */
    public Form spushForm;


    /**
     * object pull form
     */
    public Form spullForm;


    public ClientTest()
    {
        mainList = new List("Select Function", List.IMPLICIT);;
        mainList.append("Simple Push",null);
        mainList.append("Simple Pull",null);
        mainList.addCommand(CMD_SEL);
        mainList.addCommand(CMD_EXIT);
        mainList.setCommandListener(this);
    }

```



```
        spushForm = new Form("Push Object");
        spushForm.addCommand(CMD_BACK);
        spushForm.setCommandListener(this);

        spullForm = new Form("Push Object");
        spullForm.addCommand(CMD_BACK);
        spullForm.setCommandListener(this);

//open OPP client
        OppImpl = new OppClient();
        oppListener = new OppClientListener(this);
        try
        {
            OppImpl.openClient(oppListener);
        }
        catch(IOException e)
        {
            e.printStackTrace();
        }
    }

    public void startApp()
    {
        if(firstOpenFlag)
        {
            System.out.println("start");
            display = Display.getDisplay(this);

            display.setCurrent(mainList);
            firstOpenFlag = false;
        }
    }
}
```



```
public void pauseApp()
{
}

public void destroyApp(boolean unconditional)
{
    try
    {
        /** close OPP client */
        OppImpl.close(false);
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}

public void commandAction(Command c, Displayable s)
{
    if (CMD_EXIT == c)
    {
        destroyApp(false);
        notifyDestroyed();
    }
    else if (CMD_BACK == c)
    {
        display.setCurrent(mainList);
    }
    else if (CMD_SEL == c)
    {
        workMode = mainList.getSelectedIndex();
        //OppImpl.wmode = workMode;
    }
}
```



```
        switch (workMode)
        {
            case 0:
                simplePush();
                break;
            case 1:
                simplePull();
                break;
            default:
                break;
        }
    }

private void simplePush()
{
    spushForm.deleteAll();

    // fill object
    String strName = new String("jimtest.vcf");
    String strDes = new String("jimtestdes");
    pushObj = new
OppObject(pushBody, 0, bodyLen, OPP_FORMAT_VCARD, strName, strDes);

    //push object
    try
    {
        OppImpl.pushReq(serverAddr, pushObj, bodyLen, true);
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
```




```
public void simplePushConInd(boolean flag)
{
    if(flag)
    {
        spushForm.append("-->Connect Ind: (true)");
    }
    else
    {
        spushForm.append("-->Connect Ind: (false)");
    }
    display.setCurrent(spushForm);
}

public void simplePushCfm(String str)
{
    spushForm.append("-->Push Cfm: "+str);
    display.setCurrent(spushForm);
}

public void simplePushDisconInd(int tp)
{
    spushForm.append("-->Disconnect :"+tp);
    display.setCurrent(spushForm);
}

private void simplePull()
{
    spullForm.deleteAll();

    spullForm.append("-->Send PullReq");
    display.setCurrent(spullForm);
    //push object
    try
    {
```



```
        /* send a pull request */
        OppImpl.pullReq(serverAddr,true);
    }
    catch(IOException e)
    {
        e.printStackTrace();
    }
}

public void simplePullConInd(boolean flag)
{
    if(flag)
    {
        spullForm.append("-->Connect Ind: (true)");
    }
    else
    {
        spullForm.append("-->Connect Ind: (false)");
    }
    display.setCurrent(spullForm);
}

public void simplePullCfm(String str)
{
    spullForm.append("-->Pull Cfm: "+str);
    display.setCurrent(spullForm);
}

public void simplePullDisconInd(int tp)
{
    spullForm.append("-->Disconnect :"+tp);
    display.setCurrent(spullForm);
}
}
```



```
/** event listener */
import java.io.*;
import java.lang.*;
import com.motorola.iden.bluetooth.opp.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.obex.ResponseCodes;

public class OppClientListener implements OppClientRequestHandler
{
    public ClientTest parent;

    public OppClientListener(ClientTest p)
    {
        parent = p;
    }

    /** handle connect ind */
    public void onConnectInd()
    {
        switch(parent.workMode)
        {
            case 0:
                parent.simplePushConInd(isOpen);
                break;
            case 1:
                parent.simplePullConInd(isOpen);
                break;
            default:
                break;
        }
    }
}
```



```

/** handle disconnect ind */
public void onDisconnectInd(int OppResult)
{
    System.out.println("OppClientListener.java:
onDisconnectInd");
    switch(parent.workMode)
    {
        case 0:
            parent.simplePushDisconInd(OppResult);
            break;
        case 1:
            parent.simplePullDisconInd(OppResult);
            break;
        default:
            break;
    }
}

/** handle push cfm*/
public void onPushCfm(int oppResult,byte rspCode,String
desStr)
{
    parent.simplePushCfm("OppCode="+oppResult+";
OBEXCode="+rspCode+" "+desStr);
}

public void onPullCfm(int oppResult,byte rspCode,OppObject
obj,int totalBodyLen)
{
    parent.simplePullCfm("OppCode="+oppResult+"; OBEXCode="
+rspCode+
                                " \n Obj Type =
"+obj.objType+
                                " \n Obj Name =
"+obj.objName+
                                " \n Obj Desc =
"+obj.objDes+

```



```

" \n Obj Size =
"+obj.obj.length);

    }
}

/**
 * OPP server (PUSH) example.
 */
import java.io.*;
import java.lang.*;
import com.motorola.iden.bluetooth.opp.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.obex.ResponseCodes;

public class OppServer extends MIDlet implements
CommandListener, OppObjectFormat, ResponseCodes
{
    Display display;
    public Form mainForm;

    public com.motorola.iden.bluetooth.opp.OppServer OppImpl;

    public boolean onImplFlag = false;
    public int workMode;
    public OppServerListener oppListener;
    public OppObject lObj;
    public byte[] phoneName =
{0x4d, 0x6f, 0x74, 0x41, 0x46, 0x35, 0x32, 0x31};
    public byte[] serviceName = {0x4a , 0x69 , 0x6d , 0x4f , 0x62
, 0x65 , 0x78};

```



```
/**
 *commands
 */

    private final static Command CMD_EXIT = new
Command("Exit",Command.SCREEN,1);

    private final static Command CMD_BACK = new
Command("Back",Command.SCREEN,1);

    private final static Command CMD_OPEN = new
Command("Open",Command.SCREEN,1);

    private final static Command CMD_CLOSE = new
Command("Close",Command.SCREEN,1);

    public int fragnum=0;

    public OppServer()
    {
        mainForm = new Form("OPP Server");
        mainForm.addCommand(CMD_EXIT);
        mainForm.setCommandListener(this);

        //open OPP SERVER

        OppImpl = new
com.motorola.iden.bluetooth.opp.OppServer();
        oppListener = new OppServerListener(this);
        try
        {
            OppImpl.openServer(oppListener,null,(byte)0);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```



```
public void startApp()
{
    display = Display.getDisplay(this);
    display.setCurrent(mainForm);
}

public void pauseApp()
{
}

public void destroyApp(boolean unconditional)
{
}

public void commandAction(Command c, Displayable s)
{
    if (CMD_EXIT == c)
    {
        try
        {
            /** close OPP */
            OppImpl.close(false);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }

        destroyApp(false);
        notifyDestroyed();
    }
    else if (CMD_BACK == c)
    {

```



```
        display.setCurrent(mainForm);
    }
}

public void onConInd()
{
    mainForm.append("Connect Ind:");
    display.setCurrent(mainForm);
}

public void onDisconInd(int oppCode)
{
    mainForm.append("Disconnect Ind:"+oppCode);
    display.setCurrent(mainForm);
}

public void onPushInd(byte[] clientAddr, OppObject pObj, int
totalLen, boolean last)
{
    if(last)
    {
        if(pObj!=null)
            mainForm.append("Push Ind:\n objName
:"+pObj.objName+ "objLen:"+pObj.obj.length);
        display.setCurrent(mainForm);
        try
        {
            OppImpl.pushRsp(ResponseCodes.OBEX_HTTP_OK, null);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```




```
        fragnum = 0;
    }
    else
    {
        try
        {
            mainForm.append("Push Ind("+fragnum+") :\n
objName :"+pObj.objName+ "objLen:"+pObj.obj.length);
OppImpl.pushRsp(Opp.OBEX_HTTP_CONTINUE,null);

        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}

import java.io.*;
import java.lang.*;
import com.motorola.iden.bluetooth.opp.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.obex.ResponseCodes;

public class OppServerListener extends OppServerRequestHandler
{
    public OppServer parent;

    public OppServerListener(OppServer p)
    {
        parent = p;
    }
}
```



```
}  
public void onOpenCfm(boolean isOpen)  
{  
    System.out.println("OppServerListener.java: opencfm  
"+isOpen);  
  
}  
  
public void onConnectInd(boolean isOpen)  
{  
    System.out.println("OppServerListener.java:  
onConnectInd");  
    parent.onConInd();  
  
}  
  
public void onDisconnectInd(int OppResult)  
{  
    System.out.println("OppServerListener.java:  
onDisconnectInd");  
    parent.onDisconInd(OppResult);  
  
}  
  
public void onPushInd(byte[] clientAddr, OppObject obj, int  
totalLen, boolean last)  
{  
    System.out.println("OppServerListener.java: onPushInd");  
    parent.onPushInd(clientAddr, obj, totalLen, last);  
  
}  
}
```



8

Handset Features

8.1 Overview

This section will present the handset features of the iDEN Multi-Communication Device not previously covered in other chapters. These features include:

- MIDP 2.0 Platform Request
- Datebook
- Status Manager
- Location API
- Javax Location Package
- Customer Care

8.2 MIDP 2.0 Platform Request

8.2.1 Overview

The Platform Request API allows a Java application to pass a URL to the phone to have it handled by one of the phone's native applications. The iDEN Platform Request implementation supports only one type of URL: initiating a telephone call.

8.2.2 Class Description

The Platform Request API is located in package `javax.microedition.midlet`.

```
java.lang.Object
|
+ - javax.microedition.midlet.MIDlet
```



8.2.2.1 Method Description

8.2.2.1.1 `platformRequest` method

Passes a URL to the device to be handled by one of the phone's native applications.

```
public final boolean platformRequest(String URL)
throws ConnectionNotFoundException
```

The URL must begin with either "call:" or "tel:". If the URL begins with "call:", the rest of the URL should contain a valid phone number. If the URL begins with "tel:", the rest of the URL must be formatted according to RFC2806, which can be found at <http://www.ietf.org/rfc/rfc2806.txt>. When you pass this method a URL with "call:" or "tel:", this method launches the native phone application, automatically entering the phone number from the URL. The user must press the Send key to complete the phone call.

8.2.3 Code Examples

```
public class platReq extends MIDlet implements CommandListener
{
    Display myDisplay;
    List myList;

    public void startApp() throws MIDletStateChangeException
    {
        myDisplay = Display.getDisplay(this);
        myList = new List("Select test:", List.IMPLICIT);

        myList.append("Call Test", null);
        myList.append("Tel Test", null);

        myList.append("Empty", null);
        myList.append("Invalid", null);
        myList.setCommandListener(this);
        myDisplay.setCurrent(myList);
    }

    public void pauseApp()
    {
    }

    public void destroyApp(Boolean unconditional)
    {
    }

    public void commandAction(Command c, Displayable s)
    {
    }
}
```



```
if (s == myList)
{
    try
    {
        switch (((List)s).getSelectedIndex())
        {
            case 0:
                platformRequest("call:5552313");
                break;
            case 1:
                platformRequest("tel:5552312;postd=10101010");
                break;
            case 2:
                platformRequest("");
                break;
            case 3:
                platformRequest("this is an invalid URL");
                break;
        }
    }
    catch(Exception e)
    {
    }
}
}
```

8.2.4 Tips /

Once the native application receives the request, the application should be suspended and the user should be asked if he or she wants to follow through with the action.



8.3 DateBook

8.3.1 Overview

Java-based DateBook APIs provide methods to access the user's datebook data stored within the native database. The methods support such functionality as opening the datebook, adding an except date, removing an except date, getting except dates, retrieving dates of an event, setting dates for an event, getting repeat times, setting repeat times, getting number of events in the datebook, creating datebook events, importing datebook events, getting the elements of the datebook, removing datebook events, deleting all datebook events, determining the available storage, determining the event count, and choosing a MIDlet to launch when the event times out.

8.3.2 Class Descriptions

APIs for DateBook are all located in package class `com.motorola.iden.udm`.

The following will be the class hierarchy for the UDM API:

```
java.lang.Object
|
+--com.motorola.iden.udm.UDM
|
+--com.motorola.iden.udm.DateBook
|
+--com.motorola.iden.udm.DateBookEvent
|
+--com.motorola.iden.udm.DateBookRepeatEvent
|
+--java.lang.Throwable
|
+--java.lang.Exception
    |
    +--com.motorola.iden.udm.UDMException
```

The following is the Interface Hierarchy for the UDM and DateBook API:

```
com.motorola.iden.udm.UDMEntry
com.motorola.iden.udm.UDMList
```



8.3.3 Method Descriptions

8.3.3.1 UDM Method

8.3.3.1.1 `openDateBook`

Creates a `DateBook` with the phone's native datebook entries.

```
public static DateBook openDateBook(int mode) throws UDMException
```

mode must be either `READ_ONLY` or `READ_WRITE`.

The first time a MIDlet calls this method, it creates a new `DateBook` object with all the entries from the device's native datebook. When a MIDlet calls it subsequently, it returns the same `DateBook` object, after repopulating the object with the entries from the active datebook. Note that if your MIDlet has changed any `DateBookEvents` and hasn't committed them (with the `DateBookEvent.commit()`), those changes are lost.

To determine whether your application has modified a `DateBookEvent` without committing the change (with `DateBookEvent.commit()`), use `DateBookEvent.isModified()`. To determine whether the phone's native datebook database has been changed since the `DateBook` was created, use `DateBook.isCurrent()`.

8.3.3.2 `DateBookEvent` Methods

8.3.3.2.1 `commit`

Writes the data in the `DateBookEvent` to the phone's native datebook.

```
public void commit() throws UDMException
```

This method locks the phone's native datebook, writes the data, and then unlocks the datebook.

If this `DateBookEvent` is invalid, this method throws a `UDMException`. The `DateBookEvent` is invalid if its summary is null, its start time or end time is unspecified, its alarm is before current time, or it has an alarm but is an untimed event.

8.3.3.2.2 `isModified`

Returns true if any of this element's fields have been modified since the element was retrieved or last committed.

```
public boolean isModified ()
```

8.3.3.2.3 `getFieldDataType`

Returns the data type for the given field ID.

```
public int getFieldDataType(int fieldID) throws UDMException
```

If `fieldID` is `START`, `END`, `ALARM`, or `REVISION`, this method returns a `UDMEntry.DATE`. If `fieldID` is `SUMMARY`, `LOCATION`, `STYLE`, `MIDLET_SUITE`, or `MIDLET`, this method returns `UDMEntry.STRING`. If `fieldID` is `RINGER`, this method returns `UDMEntry.INT`.



8.3.3.2.4 getDate

Returns the value of the specified date field.

```
public long getDate(int fieldID) throws UDMException
```

The date is returned in milliseconds.

If you use this method with any field other than `START`, `END`, `ALARM` or `REVISION`, this method throws a `UDMException` with the string "Not supported field ID".

8.3.3.2.5 setDate

Sets the value of the specified date field.

```
public void setDate(int fieldID, long value) throws UDMException
```

If you use this method with any field other than `START`, `END`, `ALARM` or `REVISION`, this method throws a `UDMException` with the string "Not supported field ID".

Keep the following pointers in mind when setting these values:

- The phone's native datebook contains only events that occur between a month in the past and a year in the future. If you try to set the `START` or `END` fields to a value outside those bounds, this method throws an `IllegalArgumentException` with either the string "invalid start time" or "invalid end time."
- The event's `START` field must earlier than its `END` field. Otherwise the method throws `IllegalArgumentException` with the string "start time should be before end timer."
- The event's `ALARM` field must between 0 and 10080. (Max minutes of alarm – 7 days). Otherwise the method throws `IllegalArgumentException` with the string "invalid alarm value."
- The `REVISION` field is read-only. If you try to set it, this method throws a `UDMException` with the string "Revision is read only field".

8.3.3.2.6 getInt

Returns the value of the specified integer field.

```
public int getInt(int fieldID) throws UDMException
```

If the `fieldID` is not `RINGER`, this method throws a `UDMException` with the string "Not supported field ID".

8.3.3.2.7 setInt

Sets the value of the specified integer field.

```
public void setInt(int fieldID, int value) throws UDMException
```

If the `fieldID` is not `RINGER`, this method throws a `UDMException` with the string "Not supported field ID". To read the available ringers, use

`com.motorola.iden.call.CallReceive.playRinger(int index)`. The value for `RINGER` is an integer from 0 to the 250, which maps to one of the ringers stored on the phone. The value of the default ringer is `0xff`.



8.3.3.2.8 getString

Returns the value of the specified string field.

```
public String getString(int fieldID) throws UDMException
```

If `fieldID` is not `SUMMARY`, `LOCATION`, `STYLE`, `MIDLET_SUITE`, or `MIDLET`, this method throws a `UDMException` with the string "Not supported field ID".

8.3.3.2.9 setString

Sets the value of the specified string field.

```
public void setString(int fieldID, String value) throws  
UDMException
```

If `fieldID` is not `SUMMARY`, `LOCATION`, `STYLE`, `MIDLET_SUITE`, or `MIDLET`, this method throws a `UDMException` with the string "Not supported field ID".

Keep the following pointers in mind when setting these values:

The `SUMMARY` and `LOCATION` fields can contain a maximum of 64 characters if the strings have no Unicode characters, or a maximum of 32 characters if the strings do have Unicode characters.

The `MIDLET_SUITE` and `MIDLET` fields let you specify a MIDlet that is launched when this event times out. Always set the `MIDLET_SUITE` field before setting the `MIDLET` field. Note that the names of the suite and MIDlet are case sensitive. If this method cannot find a suite or MIDlet with the specified name, this method does not set the values and throws an `IllegalArgumentException`.

8.3.3.2.10 getTypedString / setTypedString

Return or set the value of the specified typed string field.

```
public String getTypedString(int fieldID, int typeID)  
throws UDMException
```

```
public void setTypedString(int fieldID, int typeID, String value)  
throws UDMException
```

The `DateBookEvent` class does not contain any typed string fields. This method always throws a `UDMException` with the string "Not supported field ID".

8.3.3.3 DateBookRepeatEvent Methods

This class represents a description for a repeating pattern for a `DateBookEvent` element. The fields are a subset of the capabilities of the `RRULE` field in `VEVENT` defined by the vCalendar 1.0 specification from the Internet Mail Consortium (<http://www.imc.org>). It is used on a `DateBookEvent` to determine how often the Event occurs.

The following table specifies the valid values for the settable fields in `DateBookRepeatEvent`.



Field Ids	Set Method	Valid Values
COUNT	setInt	Any positive int
FREQUENCY	setInt	DAILY, WEEKLY, MONTHLY, YEARLY
INTERVAL	setInt	Any positive int
END	setDate	Any valid Date
MONTH_IN_YEAR	setInt	JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY, AUGUST, SEPTEMBER, OCTOBER, NOVEMBER, DECEMBER
DAY_IN_WEEK	setInt	SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
WEEK_IN_MONTH	setInt	FIRST, SECOND, THIRD, FOURTH, FIFTH
DAY_IN_MONTH	setInt	1-31
DAY_IN_YEAR	setInt	1-366

8.3.3.3.1 addExceptDate

Adds a date to the repeat pattern's list of dates on which the event will not occur.

```
public void addExceptDate(long date)
```

The date should be greater than date offset in milliseconds from January 1, 1970, to January 1, 1999, that means date should be greater than 915148800000L. Otherwise, this method throws an `IllegalArgumentException`.

8.3.3.3.2 removeExceptDate

Removes a date from the repeat pattern's list of dates on which the event will not occur.

```
public void removeExceptDate(long date)
```

The date should be greater than date offset in milliseconds from January 1, 1970, to January 1, 1999, that means date should be greater than 915148800000L. Otherwise, this method throws an `IllegalArgumentException`.

8.3.3.3.3 getInt

Returns the value of the specified integer field.

```
public int getInt(int fieldID)
```

If the `fieldID` is not COUNT, FREQUENCY, MONTH_IN_YEAR, WEEK_IN_MONTH, DAY_IN_WEEK or DAY_IN_MONTH, this method throws a `UDMException` with the string "Not supported field ID".



8.3.3.3.4 setInt

Sets the value of the specified integer field.

```
public void setInt(int fieldID, int value)
```

If the `fieldID` is not `COUNT`, `FREQUENCY`, `MONTH_IN_YEAR`, `WEEK_IN_MONTH`, `DAY_IN_WEEK` or `DAY_IN_MONTH`, this method throws a `UDMException` with the string "Not supported field ID".

The value of the `FREQUENCY` field must be `DAILY`, `WEEKLY`, `MONTHLY` or `YEARLY`. If not, this method throws an `IllegalArgumentException` with the string "value is not valid."

8.3.3.3.5 getDate

Returns the value of the specified date field.

```
public long getDate(int fieldID)
```

If the `fieldID` is not `END`, this method throws a `UDMException` with the string "Not supported field ID".

8.3.3.3.6 setDate

Sets the value of the specified date field.

```
public void setDate(int fieldID, long value)
```

If the `fieldID` is not `END`, this method throws a `UDMException` with the string "Not supported field ID".

The phone's native datebook contains only events that occur between a month in the past and a year in the future. If you try to set the `END` field to a value outside those bounds, this method throws an `IllegalArgumentException`.

8.3.3.4 DateBook Methods

8.3.3.4.1 createDateBookEvent

Creates a `DateBookEvent` for this `DateBook`.

```
public DateBookEvent createDateBookEvent() throws UDMException
```

If there are not enough slots in the native database for a new `DateBookEvent`, this method throws a `UDMException` with the string "DateBook is full".

8.3.3.4.2 importDateBookEvent

Adds the `DateBookEvent` to this `DateBook`

```
public DateBookEvent importDateBookEvent(DateBookEvent element)  
throws UDMException
```

If you opened the `DateBook` in read-only mode, this method throws a `UDMException` with the string "DateBook is Read only".



8.3.3.4.3 isCurrent

Returns true if a DateBookEvent object has been created since the last native datebook update.

```
public static boolean isCurrent ()
```

8.3.3.4.4 isSupportedField

Returns true if this DateBook supports the given field.

```
public boolean isSupportedField(int fieldID) throws UDMException
```

Only these fields are supported: DateBookEvent.START, DateBookEvent.END, DateBookEvent.ALARM, DateBookEvent.SUMMARY, DateBookEvent.LOCATION, DateBookEvent.REVISION, DateBookEvent.RINGER, DateBookEvent.STYLE, DateBookEvent.MIDLET_SUITE, and DateBookEvent.MIDLET.

8.3.3.4.5 elements

Returns an Enumeration of DateBookEvents in this DateBook.

```
public Enumeration elements() throws UDMException
```

This method returns an Enumeration of all DateBookEvents in this DateBook. The order is not defined.

```
public Enumeration elements(long startDate, long endDate)  
throws UDMException
```

This method returns an Enumeration of all the DateBookEvents in this DateBook whose START field is greater than the given start date and whose END field is less than the given end date. The order is undefined.

If the startDate is greater than the endDate, this method throws a UDMException.

Note that the phone's native datebook contains only events that occur between a month in the past and a year in the future. If the startDate or endDate don't fall within those bounds, this method throws an IllegalArgumentException.

8.3.3.4.6 getEventCount

Returns an integer array of the number of used and empty slots in the native datebook database.

```
public int[] getEventCount() throws UDMException
```

Each non-repeat event occupies 1 slot. Each repeat event occupies two slots. To access the cells in the array, use the constants NUM_OF_REPEAT_EVENTS, NUM_OF_NON_REPEAT_EVENTS and NUM_OF_EVENTS. If the native database is closed or is no longer accessible, this method throws a UDMException.



8.3.3.4.7 removeDateBookEvent

Removes the specified DateBookEvent from the DateBook.

```
public void removeDateBookEvent(DateBookEvent element)
throws UDMException
```

If the specified DateBookEvent is not in the DateBook, this method throws a UDMException.

8.3.4 Code Examples

The following is the code example of DateBook:

```
/**
 * Demo program of Motorola iDEN SDK DateBook APIs
 * Filename: MyDateBook.java
 * <p></p>
 * <hr/>
 * <b>MOTOROLA and the Stylized M Logo are registered trademarks of
 * Motorola, Inc. Reg. U.S. Pat. & Tm. Off.<br>
 * &copy; Copyright 2003 Motorola, Inc. All Rights Reserved.</b>
 * <hr/>
 *
 * @version iDEN Datebook demo 1.0
 * @author Motorola, Inc.
 */

import com.motorola.iden.udm.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.Enumeration;

public class MyDateBook extends MIDlet implements CommandListener
{
    private Form textform;
    private Command exitCommand, checkCommand;
    private DateBook calendars;
    private DateBookEvent dateEvent;
    private StringItem userName;

    public MyDateBook()
    {
        textform = new Form("Hello, DateBook!");
        exitCommand = new Command("exit", Command.EXIT, 2);
        checkCommand = new Command("check", Command.OK, 1);
        textform.addCommand(exitCommand);
        textform.addCommand(checkCommand);
        textform.setCommandListener(this);

        try
        {
            /* Create a datebook with read and write mode. */
            calendars = UDM.openDateBook(UDM.READ_WRITE);
        }
    }
}
```



```

if (calendars != null)
{
    /* Get number of entries in DateBook. */
    int no = calendars.getNumOfEntries();
    System.out.println(
        "Number of entries in this DateBook is " + no);
}

Enumeration e;

for ( e = calendars.elements(); e.hasMoreElements(); )
{
    dateEvent = (DateBookEvent)e.nextElement();

    int[] type;
    type = dateEvent.getFields();

    /* Get the event's detail information. */
    userName = new StringItem("subject",
        dateEvent.getString(DateBookEvent.SUMMARY));
    textform.append(userName);
    userName = new StringItem("location",
        dateEvent.getString(DateBookEvent.LOCATION));
    textform.append(userName);

    for (int j= 0; j<type.length; j++)
    {
        System.out.println("Fields " + type[j] + " " +
            dateEvent.getFieldLabel(type[j]));
        if (dateEvent.getFieldDataType(type[j]) ==
            UDMEEntry.STRING)
            System.out.println(dateEvent.getString(type[j]));
        if (dateEvent.getFieldDataType(type[j]) ==
            UDMEEntry.DATE)
            System.out.println(dateEvent.getDate(type[j]));
        if (dateEvent.getFieldDataType(type[j]) ==
            UDMEEntry.INT)
            System.out.println(dateEvent.getInt(type[j]));
    }

    /* Get how often and when this event occurs. */
    DateBookRepeatEvent rpevent = dateEvent.getRepeat();
    if (rpevent != null)
    {
        int data = rpevent.getInt(
            DateBookRepeatEvent.FREQUENCY);
        System.out.println("FREQUENCY " +
            Integer.toString(data, 16) );

        data = rpevent.getInt(
            DateBookRepeatEvent.MONTH_IN_YEAR);
        System.out.println("MONTH_IN_YEAR " +
            Integer.toString(data, 16) );
    }
}

```



```

        data = rpevent.getInt(
            DateBookRepeatEvent.WEEK_IN_MONTH);
        System.out.println("WEEK_IN_MONTH " +
            Integer.toString(data, 16) );
        data = rpevent.getInt(
            DateBookRepeatEvent.DAY_IN_WEEK);
        System.out.println("DAY_IN_WEEK " +
            Integer.toString(data, 16) );
        data = rpevent.getInt(
            DateBookRepeatEvent.DAY_IN_MONTH);
        System.out.println(data);
        System.out.println("Repeat End "+
            rpevent.getDate(DateBookRepeatEvent.END));
        long[] except = rpevent.getExceptDates();
        for (int d = 0; d < except.length; d++ )
        {
            System.out.println("Except date is " + except[d]);
        }
    }
}

/* Get an array of integers representing the amount of
   additional slots that can be stored on the native database.
   * Each non-repeat event occupies 1 slot.
   * Each repeat event occupies two slots.
   */
int[] entryField;
entryField = calendars.getAvailableStorage();
for (int i= 0; i < entryField.length; i++)
{
    System.out.println(String.valueOf(entryField[i]));
}

entryField = calendars.getEventCount();
for (int i= 0; i < entryField.length; i++)
{
    System.out.println(String.valueOf(entryField[i]));
}

/* Create one event */
System.out.println("Current phone time is "+
    System.currentTimeMillis());
long currentTime = 0;
dateEvent = calendars.createDateBookEvent();
dateEvent.setString(DateBookEvent.SUMMARY,
    "Non-Repeat Event");
currentTime = System.currentTimeMillis()+ 60*60000;
dateEvent.setDate(DateBookEvent.START, currentTime);
dateEvent.setDate(DateBookEvent.END,  currentTime + 600000);

/* Associate a midlet_suite_name and a midlet_name
   * with this event
   */

```



```

        dateEvent.setString(MIDLET_SUITE, "midlet_suite_name");
        dateEvent.setString(MIDLET, "midlet_name");
        dateEvent.commit();

    }
    catch (Exception ex)
    {
        ex.printStackTrace();
    }
}

public void startApp()
{
    Display.getDisplay(this).setCurrent(textform);
}

public void pauseApp()
{
}

public void destroyApp(boolean unconditional)
{
}

public void commandAction(Command c, Displayable d)
{
    if(c == exitCommand)
    {
        try
        {
            calendars.close();
        }
        catch (Exception e) {
        }
        notifyDestroyed();
    }
    else if (c == checkCommand)
    {
        System.out.println(DateBook.isCurrent());
    }
}
}

```

8.3.5 Compiling & Testing Datebook MIDlets

- Method `DateBook.getEventCount()` always returns an empty array since there is no native support for this method.
- Method `DateBook.isCurrent()` always returns true since there is no native support for this method.
- Method `DateBook.entryIsModified(PhoneBookEntry entry)` always returns false since there is no native support for this method.



8.4 Status Manager

8.4.1 Overview



This API is only available on these handsets.

The StatusManager class lets an application query the status of various features on the device. These features include: battery level, signal strength, pack data registration, mobile IP registration, whether call forwarding is enabled, the current active line, number of unread text messages, total number of text messages, unheard voice mail messages, total number of voice mail messages, number of unread NetAlert messages, the total number of NetAlert messages, and the state of the high audio speaker.

8.4.2 Class Description

The API for the StatusManager is located in package `com.mot.iden.device`.

```
java.lang.Object
|
+ - com.mot.iden.device.StatusManager
```

8.4.3 Method Descriptions

8.4.3.1 StatusManager Methods

8.4.3.1.1 getStatus

Returns the status of the specified feature.

```
public static int getStatus(int feature)
throws IllegalArgumentException
```

`feature` must be one of the following:

- BATTERY_LEVEL
- SIGNAL_STRENGTH
- CALL_FORWARD_STATE
- CURRENT_ACTIVE_LINE
- NUM_UNREAD_TEXT_MSG



- NUM_TEXT_MSG
- NUM_UNHEARD_VOICE_MAIL
- NUM_VOICE_MAIL
- NUM_UNREAD_NET_ALERT
- NUMD_NET_ALERT
- HIGH_AUDIO_STATE
- CURRENT_VIBE_STYLE_LINE1
- CURRENT_VIBE_STYLE_LINE2
- CURRENT_VIBE_STYLE_ALERT

If `feature` is not one of those values, this method throws an `IllegalArgumentException`.

8.4.3.1.2 `isRegistered`

Returns true if the specified feature has registered; false, otherwise.

```
public static boolean isRegistered(int feature)  
throws IllegalArgumentException
```

`feature` must be one of the following:

```
PD_REGISTRATION  
MIP_REGISTRATION
```

If `feature` is not one of those values, this method throws an `IllegalArgumentException`.



8.5 Location API

8.5.1 Overview

The Location API lets users and developers access GPS position information such as latitude, longitude, altitude, speed, and so on. This feature is provided as a built-in application in the phone's standard ergonomics and as a J2ME™ API developers can use to create custom AGPS-based applications. This section describes some of the phone's features that affect AGPS accuracy and availability from a J2ME™ MIDlet. This API provides access to NMEA stream of messages and can turn on the GPS chip set's NMEA capabilities.

- *Accuracy*—The GPS receiver is designed to receive location fixes within a preset level of geographic accuracy as determined by the network provider. Using the Location API, J2ME™ developers can retrieve a fix; however, the location value is not guaranteed to be within this level of accuracy. The API provides methods to determine whether a given fix is accurate or not.

Motorola strives to achieve the highest possible accuracy; however, no GPS system can provide perfect accuracy in all situations. GPS accuracy can be affected by a multitude of potential error-introducing factors, including GPS satellite signal conditions and packet data availability. Position accuracy is not guaranteed nor implied.

- *Assist Data*—AGPS uses cellular assisted data to retrieve a location fix. The Location API provides J2ME™ developers with a method to determine whether cellular assisted data is used for a given fix.

The API provides the location functionalities required for Java applications to access GPS position information such as the following:

- Latitude
- Longitude
- Altitude
- Time Stamp
- Travel Direction
- Speed
- Altitude Uncertainty
- Speed Uncertainty

The Location API uses the GPS Privacy setting in the Main Menu of the phone when a MIDlet invokes the API. Based on the GPS Privacy setting value, the MIDlet does or does not have the access to the position information. The API will use the user's Privacy setting accordingly before providing position information. Some examples include:

- If the user's GPS Privacy setting is set to "Restricted", the Java API will return the position with all the attributes set to `UNAVAILABLE` and with the `PositionConnection`'s status code set to `POSITION_RESPONSE_RESTRICTED`.



- If the user's GPS Privacy setting is set to "Unrestricted", the Java API will be able to access GPS data and will return the position.
- If the user's GPS Privacy setting is set to "By Permission", the application is suspended, as the Java API brings up a system screen to prompt the user for permission to grant position access for this application. If the user does not grant permission, the Java API will return the position with all the attributes set to `UNAVAILABLE` and with the `PositionConnection`'s status code set to `POSITION_RESPONSE_RESTRICTED`. After selecting one of the permission options, the user needs to resume the application.

After permission is granted, the Java API brings up a system screen to prompt the user if the Almanac data in the phone is out of date or invalid, and the phone is not provisioned for packet data service. This is done only once after the phone powers up. If the user gives permission to override Almanac data, the Java API tries to retrieve position data. If user does not grant the Almanac override, the Java API returns the position with its attributes set to `UNAVAILABLE` and the status of `PositionConnection` set to `POSITION_RESPONSE_NO_ALMANAC_OVERRIDE`.

8.5.2 Class Description

The API for the NMEA output messages is located in package `com.motorola.iden.position`.

```
java.lang.Object
```

```
|
```

```
+ - com.motorola.iden.position.PositionConnection
```

The `PositionConnection` interface supports the creation of a connection to the GPS receiver (driver). GPS position can be retrieved and status can be obtained after creating a connection. Only one connection is allowed at a time. This API must be called from a separate thread from the main application thread.

To get a `PositionConnection`, the `MIDlet` must use the generic `Connector` class. For example:

```
com.motorola.iden.position.PositionConnection sc =
(com.motorola.iden.PositionConnection)Connector.open(String name);
```

String name should be one of the following:

- `name = "mposition:delay=no"`
- `name = "mposition:delay=low"`
- `name = "mposition:delay=high"`

The following descriptions of delay values are based on the default settings. These settings are carrier definable and can differ among carriers. Java has no access to change these values.

- `delay=no` This option is designed to provide the serving cell latitude and longitude to an application immediately after it requests them. Because all other attributes in the `AggregatePosition` class may be set to `UNAVAILABLE`, an application should use this connection only to access the serving cell latitude and longitude. This request does not make use of the GPS chipset. If the handset is outside of the network coverage area, the serving cell latitude and longitude will be set to 0.
- `delay=low` This option provides a response to the application in a few seconds. New assist data is retrieved only if no assist data exists or if the assist data is older than the Maximum Assist Data Age (MADA). This operation is transparent to the application. This



option is designed to provide all the position attributes with assistance from the Location Enhanced Service (LES) Server. To exercise this option, the device needs to have packet data service. Currently the maximum response time for this type of request is 32 seconds. If the API times out, the position will be returned with appropriate status and error code. If a low-delay request is made outside of the network coverage area, then the API will not get the assist data from the LES. The fix will proceed without assist data, and the timeout will remain at the low-delay value of 32 seconds.

- `delay=high` This option provides a response to the application where delay is longer than a `delay=low` setting. It provides for an assisted or autonomous fix for the application. The phone uses existing assist data only if it is available and valid; otherwise, the location fix shall proceed autonomously. Currently, maximum response time for this type of request is 180 seconds. If the response times out, position will be returned with the appropriate status and error code.

Only one request of `getPosition()` can be made or be pending at any time. If the application makes multiple requests without getting a response to the previous request, a null position value is returned or an exception is thrown. The next section provides more detail on this method.

8.5.3 Method Descriptions

8.5.3.1 PositionConnection Methods

8.5.3.1.1 `getPosition`

Returns a position.

```
public AggregatePosition getPosition()
```

This method returns a position using the same delay setting used for `Connector.open()`.

This method is a synchronous, blocking method, which means it blocks until a response, error, or timeout occurs. Closing the `PositionConnection` from a separate thread can unblock these calls. Once the connection is closed, it needs to be opened again using `Connector.open()`.

If the `PositionConnection` is closed while a call to this method is pending or a second call has been made to this method, then this method returns a null position. Unknown errors may occur during a location fix, which may also cause null position value to be returned.

```
public AggregatePosition getPosition(String name)
```

This method returns a new position with the delay parameters specified by `name`. This method also allows an application to obtain a fix with an accurate velocity and heading direction. Note that obtaining an accurate velocity and heading direction may cause a significant delay with weak GPS signal strength. In strong GPS signal coverage this operation may take no longer than a standard fix.

The argument required for accurate velocity and heading direction is as follows:

```
String name = "delay=low;fix=extended";    // or
String name = "delay=high;fix=extended";
```

This method is a synchronous, blocking method, which means it blocks until a response, error, or timeout occurs. Closing the `PositionConnection` from a separate thread can



unlock these calls. Once the connection is closed, it needs to be opened again using `Connector.open()`.

If the `PositionConnection` is closed while a call to this method is pending or a second call has been made to this method, then this method returns a null position. Unknown errors may occur during a location fix, which may also cause null position value to be returned.

8.5.3.1.2 requestPending

Returns true if there is a pending position request.

```
public boolean requestPending()
```

Call this method on a connection before making a new request from another thread.

8.5.3.1.3 getStatus

Returns the status for the last `getPosition()` call.

```
public int getStatus()
```

Call this method only after calling `getPosition()`. Use the position obtained only if `getStatus()` returns `POSITION_RESPONSE_OK`. The following is a list of the possible return values for this method:

- `POSITION_NO_RESPONSE` indicates that the device is not responding. No position information will be available, and all the attributes of the position will be set to `UNAVAILABLE`.
- `POSITION_RESPONSE_ERROR` indicates that an error occurred while retrieving the position. If possible, the cell latitude and longitude will be available, but all position's attributes will be set to `UNAVAILABLE`.
- `POSITION_RESPONSE_OK` indicates that the obtained position is a valid position. All position's attributes will be available.
- `POSITION_RESPONSE_RESTRICTED` indicates that the user has set the device so it does not provide the position information. No position information will be available, and the position's attributes will be set to `UNAVAILABLE`.
- `POSITION_WAITING_RESPONSE` indicates that the API is waiting for a response from the position device. `POSITION_WAITING_RESPONSE` will be returned if `getStatus()` method is called before `getPosition()` method.
- `POSITION_RESPONSE_NO_ALMANAC_OVERRIDE` indicates that the Almanac is outdated, and the user is restricted to override. No position information will be available, and all the attributes of the position will be set to `UNAVAILABLE`.

8.5.3.1.4 getNMEASentence()

Returns an NMEA Sentence for the specified type.

```
public String getNMEASentence (int type)
    throws IllegalArgumentException
```

Following are the valid NMEA message types.

- `PositionDevice.GPGGA`
- `PositionDevice.GPGLL`



- PositionDevice.GPGSA
- PositionDevice.GPGSV1
- PositionDevice.GPGSV2
- PositionDevice.GPGSV3
- PositionDevice.GPRMC
- PositionDevice.GPVTG

If the message type is other than above, this method throws an `IllegalArgumentException`.

If the method cannot fulfill the request for an NMEA sentence, this method returns a null string.

This first time you call this message, it turns on the GPS chip for NMEA messages. It's the application's responsibility to stop the NMEA request once it is done using it.

8.5.3.1.5 stopNMEASentence

Stops the NMEA request and turns off the GPS chip after 10 seconds.

```
public void stopNMEASentence()
```

This method stops only the NMEA access and keeps the connection open so the application can use the connection to retrieve the position fix or reuse it for NMEA messages.

8.5.3.2 AggregatePosition Methods

8.5.3.2.1 getResponseCode

Returns the response code for this position.

```
public int getResponseCode ()
```

The following is a list of returned response codes:

- `POSITION_OK` indicates that the obtained position is valid and accurate.
- `ACC_NOT_ATTAIN_ASSIST_DATA_UNAV` indicates that the location fix has timed out. The fix could not be accurately obtained since assistance data was not unavailable.
- `ALMANAC_OUT_OF_DATE` indicates that the Almanac is out of date.
- `ACCURACY_NOT_ATTAINABLE` indicates that the location fix has timed out, and the requested accuracy is not attainable.
- `BATTERY_TOO_LOW` indicates that the battery is too weak to retrieve a fix.
- `FIX_NOT_ATTAIN_ASSIST_DATA_UNAV` indicates that the location fix has timed out because a fix is not attainable, and assist data is unavailable.
- `FIX_NOT_ATTAINABLE` indicates that the location fix has timed out because a fix is not attainable.
- `GPS_CHIPSET_MALFUNCTION` indicates that the GPS chipset is malfunctioning.



- **UNAVAILABLE** indicates that an unknown error has occurred. This is the default response code.

These response codes are used in conjunction with `PositionConnection.getStatus()` to determine the quality of the retrieved position. These values are valid only when either `POSITION_RESPONSE_ERROR` or `POSITION_RESPONSE_OK` have been returned.

The following table shows the possible combinations of response codes for these two methods:

PositionConnection Status Values	Response Codes
<code>POSITION_RESPONSE_OK</code>	<code>POSITION_OK</code> <code>ACCURACY_NOT_ATTAINABLE</code> <code>ACC_NOT_ATTAIN</code> <code>ASSIST_DATA_UNAV</code>
<code>POSITION_RESPONSE_ERROR</code>	<code>FIX_NOT_ATTAINABLE</code> <code>FIX_NOT_ATTAIN</code> <code>ASSIST_DATA_UNAV</code> <code>BATTERY_TOO_LOW</code> <code>GPS_CHIPSET_MALFUNCTION</code> <code>ALMANAC_OUT_OF_DATE</code> , <code>UNAVAILABLE</code>

8.5.3.2.2 `getAssistanceUsed`

Checks if a fix has been retrieved using assistance.

```
public boolean getAssistanceUsed ()
```

8.5.4 Code Examples

```
void getViaPositionConnection() throws IOException {
    PositionConnection c = null;
    String name = "mposition:delay=low";
    try{
        c = (PositionConnection)Connector.open(name);
        AggregatePosition oap = c.getPosition();
        // Returns the AggregatePosition which contains the position
        // using the parameter passed when connection was opened.
        // Application should only check status by calling getStatus()
        // after getPosition() or getPosition(String name) returns.
        // Otherwise, it returns the same status and is
        // considered an invalid call of getStatus().
        // check the status code for permission and almanac over ride
        if(c.getStatus() ==
            PositionConnection.POSITION_RESPONSE_RESTRICTED)
        {
```




```
// means user has restricted permission to get position
}
else if(c.getStatus() ==
        PositionConnection.POSITION_RESPONSE_NO_ALMANAC_OVERRIDE)
{
    // means device has Almanac out of date and
    //the user has not granted to override
}
else if(c.getStatus() ==
        PositionConnection.POSITION_NO_RESPONSE)
{
    // means no response from device
}
if (oap != null ) {
    if(c.getStatus() ==
        PositionConnection.POSITION_RESPONSE_OK)
    {
        // Good position
        // Check for any error from device on position
        // Application needs to check for null position
        if(oap.getResponseCode() == PositionDevice.POSITION_OK) {
            // no error in the position
            if(oap.hasLatLon()) {
                // int value of Latitude and Longitude of the position in
                // arc minutes multiplied by 100,000 to maintain accuracy
                // or UNAVAILABLE if not available
                int lat = oap.getLatitude();
                int lon = oap.getLongitude();
                // String representation of the Latitude and Longitude.
                String LATDEGREES = oap.getLatitude(Position2D.DEGREES);
                String LONGDEGREES = oap.getLongitude(Position2D.DEGREES);
            }
            if(oap.hasSpeedUncertainty()) {
                // speed and heading value are valid
                int speed = oap.getSpeed();
```



```
        if (hasTravelDirection()) {
            // heading is available
            int travelDirection = oap.getTravelDirection();
        }
    }
    if(oap.hasAltitudeUncertainty()) {
        int alt = oap.getAltitude(); //altitude of position
                                    // in meters.
    }
}
// handle the errors...or request again for good position
// or display message to the user.
else if(oap.getResponseCode() ==
        PositionDevice.ACCURACY_NOT_ATTAINABLE) {
    // the position information was provided but enough
    // accuracy may not be attainable
}
else if(oap.getResponseCode() ==
        PositionDevice.ACC_NOT_ATTAIN_ASSIST_DATA_UNAV) {
    // the position information was provided but enough
    // accuracy, assistant data unavailable
}
} // end of position response ok
else if(c.getStatus() ==
        PositionConnection.POSITION_RESPONSE_ERROR)
{
    // indicate an error occurred while getting the position
    if(oap.getResponseCode() ==
        PositionDevice.FIX_NOT_ATTAINABLE) {
        // means position information not provided (timeout)
    }
    else if(oap.getResponseCode() ==
        PositionDevice.FIX_NOT_ATTAIN_ASSIST_DATA_UNAV) {
        // means position information not provided (timeout) and
        // assistant data unavailable
    }
}
```



```
}
else if(oap.getResponseCode() ==
        PositionDevice.BATTERY_TOO_LOW) {
    // means battery is too low to provide fix
}
else if(oap.getResponseCode() ==
        PositionDevice.GPS_CHIPSET_MALFUNCTION) {
    // means GPS chipset malfunction
}
else if(oap.getResponseCode() ==
        PositionDevice.ALMANAC_OUT_OF_DATE) {
    // means almanac out of date to get fix
    // This scenario occurs when user overrides almanac but
    // device is not packet data provisioned
}
else {
    // Unknown error occurs
}
} // end of position response error
// position is null
} finally {
    if ( c != null)
        c.close();
}
```

New positions can be obtained using the following method on the same PositionConnection object until the close() method is called.

```
AggregatePosition cell = c.getPosition("delay=no");
```

Or

```
AggregatePosition oap = c.getPosition("delay=low");
```

Or

```
AggregatePosition oap = c.getPosition("delay=high");
```

In addition, to obtain better accurate speed and direction



```
AggregatePosition oap = c.getPosition("delay=low;fix=extended");
```

Or

```
AggregatePosition oap = c.getPosition("delay=high;fix=extended");
```

The following is an NMEA code example:

```
try
{
    PositionConnection posCon =
        (PositionConnection)Connector.open("mposition:delay=low");

    String temp1 = posCon.getNMEASentence(PositionDevice.GPGGA);
    if(posCon.getStatus() == POSITION_RESPONSE_OK)
    {
        if(temp1 != null && temp1.equals(""))
        {
            // valid GPGGA string, parse it to extract
            // the required information
        }
        else if(posCon.getStatus() == POSITION_RESPONSE_RESTRICTED)
        {
            // User has not granted permission to access
            // its location information
        }
        else if (posCon.getStatus() ==
            POSITION_RESPONSE_NO_ALMANAC_OVERRIDE)
        {
            // User has not granted permission to override
            // its almanac information
        }
        else
        {
            // unusual error occurred
        }
    }
    catch(IllegalArgumentException ie) {
    }
    catch(Exception ex) {
    }
}
```

8.5.5 Tips /

- The GPS receiver requires access to both the iDEN network and GPS satellite signals to obtain rapid fixes. It is recommended that once the first fix is obtained, the application monitor the response codes and vary the times between position requests accordingly. This recommendation is to handle the real world case where an application requests fixes rapidly (less than 10 seconds apart) and then loses network and GPS coverage (by entering a parking structure, basement, etc.) The GPS system will continue to try to find the unit's position and will go into a longer integration or acquisition mode that, once started, may take so long to finish that it may miss GPS signals once back in coverage. The recommended practice is to make fixes rapidly until a response code of



`FIX_NOT_ATTAINABLE` or `FIX_NOT_ATTAIN_ASSIST_DATA_UNAV` is returned several times in a row (for about 10 requests for `delay=low` and about 5 requests for `delay = high`). After this occurs, the application may wish to start the acquisition over from the beginning in anticipation that the phone might be back in GPS coverage. To do so, the application must wait 15 to 20 seconds after receiving the last response code before requesting a new fix. After this pause, the application can continue requesting fixes at its normal frequency.

- The GPS subsystem requires about one second to calculate a new fix, so any request for a new fix during this one-second period may result in the exact same position information including the time stamp. Therefore it is recommended that an application request a new position no more than once per second.
- If an application needs continuous position, use "delay=low" once and "delay=high" thereafter even if the first fix does not succeed. The reason for this is because of network failures. When there is a network failure, there is a 12 to 24 second communication timeout from the LES.
- Use "delay=no" if the application needs only the cell latitude and longitude. This does not use the AGPS chip on the device.
- Applications must handle all response codes returned by the `AggregatePosition.getResponseCode()` method and the `PositionConnection.getStatus()` method. `getStatus()` provides the connection's status after the fix and user interaction status with regards to permission. `getResponseCode()` provides information about the position itself.
- Applications must always check the speed uncertainty value before using speed and heading. Although it is counter-intuitive, the presence of speed uncertainty denotes that the speed and heading value are accurate. Therefore, if a call to `hasSpeedUncertainty()` returns true, the speed and heading returned by the API are valid.
- If an application calls `getPosition(String name)` method with the "fix=extended" tag, this method will return accurate velocity and heading direction; however, there is a time penalty since it takes longer to calculate the accurate velocity and heading direction when the method is called.
- The method `PositionConnection.getStatus()` provides the status of the connection when the method `PositionConnection.getPosition()` was called. Whereas, `AggregatePosition.getResponseCode()` returns the detailed response code.
- Getting a position for the first time after the phone powers on is referred as a "cold start". A position retrieved within ten seconds of the previous fix is referred to as a "hot start". A position retrieved after ten seconds of the previous fix is a "warm start". After 1 hour since the last fix will set the device back to "cold start". Therefore, "hot start" is the quickest way of retrieving a fix.
- It is highly recommended that the antenna remain extended while getting fixes.
- There is a battery impact when the NMEA API is used heavily.
- If the application will need NMEA data again in less than 10 seconds, there is no value in calling `stopNMEASentence()` because the GPS chip will stay on for 10 seconds after calling `stopNMEASentence()`.



- First call of `getNMEASentence()` will turn on the GPS chip and it stays on until application calls `stopNMEASentence()`.



8.6 Javax Location Package

8.6.1 Overview



This API is only available on these handsets.

This feature provides APIs that allow a J2ME application to obtain information about the present geographic location and to access a database of known landmarks stored in the terminal.

Developers SHOULD read the JSR-179 spec before reading this guide. Not all classes and methods are addressed in this developer guide. For those classes and methods, please refer to JSR -179 1.0 document or <http://www.jsr.org>.

8.6.2 Package Description

The JSR-179 Location API is in package `javax.microedition.location`.

Classes	
AddressInfo	The AddressInfo class holds textual address information about a location.
Coordinates	The Coordinates class represents coordinates as latitude-longitude-altitude values.
Criteria	The criteria used for the selection of the location provider is defined by the values in this class.
Landmark	The Landmark class represents a landmark.
LandmarkStore	The LandmarkStore class provides methods to store, delete and retrieve landmarks from a persistent landmark store.
Location	The Location class represents the standard set of basic location information.
LocationProvider	This is the starting point for applications using this API and represents a source of the location information.
Orientation	The Orientation class represents the physical orientation of the terminal.
QualifiedCoordinates	The QualifiedCoordinates class represents coordinates as latitude-longitude-altitude values that are associated with an accuracy value.
Interfaces	
LocationListener	The LocationListener represents a listener that receives events associated with a particular LocationProvider.
ProximityListener	This interface represents a listener to events associated with detecting proximity to some registered coordinates.
Exceptions	
LandmarkException	The LandmarkException is thrown when an error related to handling landmarks has occurred.
LocationException	The LocationException is thrown when a location API specific error has occurred.



8.6.2.1 **javax.microedition.location.LandmarkStore**

```
public void createLandmarkStore(String storeName)
```

Creating new landmark stores is not supported, thus a `LandmarkException` will be thrown from this method.

Tip: Whenever this method is called, after `javax.microedition.location.LandmarkStore.management` permission is granted, a `LandmarkException` will be thrown with message "Creating landmark store not supported".

8.6.2.2 **public void deleteLandmarkStore(String storeName)**

Deleting landmark stores is not supported, thus a `LandmarkException` will be thrown from this method.

Tip: Whenever this method is called, after `javax.microedition.location.LandmarkStore.management` permission is granted, a `LandmarkException` will be thrown with message "Deleting landmark store not supported".

8.6.2.3 **public static String[] listLandmarkStores()**

This method always returns null because no more landmark stores other than the default can exist.

Tip: Whenever this method is called, after `javax.microedition.location.LandmarkStore.read` permission is granted, a null will be returned.

8.6.2.4 **public static LandmarkStore getInstance(String storeName)**

This method always returns the default `LandmarkStore` instance.

Tip: Only if the parameter `storeName` is null, a `LandmarkStore` instance can be retrieved.

8.6.2.5 **public void addLandmark(Landmark landmark, String category)**

Tip: Maximum acceptable landmark name length is 32. If the landmark name is longer than 32 characters, `IllegalArgumentException` will be thrown with message "Landmark name oversize". A maximum of 256 landmarks can be stored in landmark store, adding the 257th landmark into landmarkstore will result in `IOException` with message "There are no resources available to add the landmark".

8.6.2.6 **public void addCategory(String category)**

Tip: Maximum acceptable category name length is 32. If the landmark name is longer than 32 characters, it will be truncated to 32. A maximum of 64 categories can be supported in landmark store, adding the 65th category will meet `IOException` with message "There are no resources to add a new category".

8.6.2.7 **public void addCategory(String category)**

Tip: The maximum acceptable category name length is 32. If the landmark name is longer than 32 characters, it will be truncated to 32. A maximum of 64 categories can be supported in landmark store, adding the 65th category will meet `IOException` with message "There are no resources to add a new category".



8.6.3 javax.microedition.location.Location

8.6.3.1 public AddressInfo getAddressInfo()

Address info determination is not supported, so null will always be returned by this method.

8.6.3.2 public String getExtraInfo(String mime)

Three MIME types are supported.

When MIME type is "application/X-jsr179-location-nmea", the returned string is a sequence of GPGLL and GPGLL sentences representing this location, according to the syntax specified in the NMEA 0183 v3.1 specification.

When MIME type is "application/X-jsr179-location-lif", the returned string contains an XML formatted document containing the <pd> element defined in the LIF Mobile Location Protocol TS101 v3.0.0 as the root element of the document.

When MIME type is "text/plain", the returned string contains some info exposing the location fix status.

Tip: A sample returned string of `getExtraInfo("application/X-jsr179-location-nmea")` :

```
$GPGLL,140234,26:08.76784,N,-80:15.22240,W,1,6,109.181,,,7.0,M,,
$GPGLL,26:08.76784,N,-80:15.22240,W,000,A
```

A sample returned string of `getExtraInfo("application/X-jsr179-location-lif")` :

```
lif:<pd><time>140234</time><shape><Point><coord><X>26.146130666666668
</X><Y>-80.
25370666666666</Y></coord></Point></shape><alt>7.0</alt><alt_acc>20.1
73</alt_acc><speed>10000.0</speed><direction>90.0</direction></pd>
```

Returned string of `getExtraInfo("text/plain")` may be one the following.



<code>location.getExtrInfo("text/plain")</code>	<code>Location.is Valid()</code>
"Valid Location."	true
"Invalid Location: Time out, fix unattainable."	false
"Invalid Location: Time out, fix unattainable and assist unavailable."	false
"Invalid Location: Time out, accuracy unattainable."	false
"Invalid Location: Time out, accuracy unattainable and assist unavailable."	false
"Invalid Location: Battery too low."	false
"Invalid Location: GPS chipset malfunction."	false
"Invalid Location: Almanac out of date."	false
"Invalid Location: Request has been cancelled."	false
"Invalid Location: Unknown error."	false

In common cases, a location fix may be a invalid due to reasons listed above. Whether to make use of the invalid location depends on application developer. It is recommended to use an invalid fix if the returned string of `Location.getExtrInfo("text/plain")` is "Invalid Location: Time out, accuracy unattainable." or "Invalid Location: Time out, accuracy unattainable and assist unavailable."

8.6.3.3 `public int getLocationMethod()`

Returned location method can be one of the three:

```
MTE_CELLID,
MTE_SATELLITE | MTE_TERMINAL | MTA_ASSISTED,
MTE_SATELLITE | MTE_TERMINAL | MTA_UNASSISTED
```

8.6.4 `javax.microedition.location.LocationProvider`

8.6.4.1 `public static LocationProvider getInstance(Criteria criteria)`

Location provider can be returned by any criteria except the one with address info required. Different criteria will impact on provider obtaining location.

Tip: Address info determination is not supported, thus if address info is required by the parameter criteria, null will be returned by this method.



8.6.4.2 public Location getLocation(int timeout)

When timeout parameter is -1, the default get location time out of 30 seconds will be applied.

Tips:

- Inside a MIDlet, getLocation() methods are synchronized. Only one location request can be performed at a time, whether the requests are from the same provider instance or from different provider instances. Since concurrent MIDlets may be supported, viewing from across MIDlets, multiple requests may be performed at the same time.
- When a location instance is returned by getLocation(), location validation can be decided by Location.isValid() and some response info can be retrieved by using Location.getExtraInfo("text/plain"). In most cases, the location instance may not be valid with degraded fix or non-assisted fix.
- A location request's quality is decided by the criteria applied upon this location provider.
- As in iDEN AGPS API, a location request may be with one of the three quality levels: no delay, low delay and delay tolerant. JSR179 API location requests implicitly apply the quality level by the following rules:
 - If a location provider is required with some horizontal or vertical accuracy, and NOT allowed to cost, its location requests will be with delay tolerant level;
 - If a location provider is required with some horizontal or vertical accuracy, allowed to cost, and power consumption level is low or medium or no requirement, its location requests will be with delay tolerant level;
 - If a location provider is required with some horizontal or vertical accuracy, allowed to cost, and power consumption level is high, its 1st location request will be with low delay and subsequent requests will be with delay tolerant level;
 - If a location provider's horizontal and vertical accuracies are NOT required, and NOT allowed to cost, and power consumption is medium or high or no requirement, its location requests will be with delay tolerant level;
 - If a location provider's horizontal and vertical accuracies are NOT required, and allowed to cost, and power consumption is medium or no requirement, its location requests will be with low delay level;
 - If a location provider's horizontal and vertical accuracies are NOT required, and allowed to cost, and power consumption is high level, its 1st location request will be with low delay and subsequent requests will be with delay tolerant level;
 - If a location provider's horizontal and vertical accuracies are NOT required, and allowed to cost, and power consumption is low level, its location requests will be with no delay level.

8.6.4.3 public int getState()

Only two states are possible to be returned, AVAILABLE or TEMPORARILY_UNAVAILABLE.

Tip: If it is longer than 3 minutes from last location request, AVAILABLE state will be returned.



8.6.4.4 public void reset()

This method will cancel location requests from the same MIDlet, with no effect on location requests from other MIDlets.

Tip: This method can work on location update progress. For example, provided a location update is scheduled every 30 seconds, when the progress is making the 3rd update, some thread invokes reset() method, therefore this update will be aborted silently and wait for the 4th update later on.

On the other hand, reset() does not work on proximity detection progress. Proximity detection progress will not be interrupted by reset().

8.6.4.5 public void setLocationListener(LocationListener listener, int interval, int timeout, in maxAge)

Default location update interval is 60 seconds, default location update time out is 30 seconds, and default max age is 8 seconds.

8.6.5 javax.microedition.location.Orientation

8.6.5.1 public static Orientation getOrientation()

Pitch and Roll determination is not supported. Compass azimuth is related to true north, therefore isOrientationMagnetic returns false.



8.6.6 Code Examples

The following is the code example of `LandmarkStore.deleteLandmark()` and `LandmarkStore.removeLandmarkFromCategory()`.

```
public void deleteLm( int i)
{
    try {
        if((i >= 0) && (i <= lmItems.size()))
        {
            Landmark lm = (Landmark)lmItems.elementAt(i);
            if(lm != null)
            {
                lmItems.removeElementAt(i);
                lmList.delete(i);
                lmStore.deleteLandmark(lm);
            }
        }
    }
    catch (Exception e)
    {
        resultForm.deleteAll();
        resultForm.append("" + e);
        disp.setCurrent(resultForm);
    }
}

public void removeLmFromCat(int i)
{
    String categoryName;
    categoryName = catTF.getString();
    try{
        if((i >= 0) && (i <= lmItems.size()))
        {
            Landmark lm = (Landmark)lmItems.elementAt(i);
            if (lm != null)
            {
                lmStore.removeLandmarkFromCategory(lm,
categoryName);
            }
        }
        disp.setCurrent(lmList);
    }
    catch (Exception e)
    {
        resultForm.deleteAll();
        resultForm.append("" + e);
        disp.setCurrent(resultForm);
    }
}

// delete and remove landmark
public DelRemLandmarkTest()
```



```

{
    disp = Display.getDisplay(this);

    lmList = new List("Landmark List", List.IMPLICIT);

    try {
        lmStore = LandmarkStore.getInstance(null);
        if(lmStore == null)
        {
            throw new Exception("Can't get landmarkStore
Instance!");
        }
        landmarks = lmStore.getLandmarks();
    } catch (Exception e)
    {
        resultForm.deleteAll();
        resultForm.append("" + e);
    }

    if (landmarks!=null)
    {
        while (landmarks.hasMoreElements())
        {
            lm = (Landmark)landmarks.nextElement();
            qc = lm.getQualifiedCoordinates();
            lmItems.addElement(lm);
            if (qc!=null)
            {
                lmList.append(lm.getName()+"", "+qc.getLatitude()+"", "+qc.getLongitude()
                +""", "+qc.getAltitude(), null);
            }
            else
            {
                lmList.append(lm.getName(), null);
            }
        }
    }

    lmList.addCommand(deleteCommand);
    lmList.addCommand(removeCommand);
    lmList.addCommand(exitCommand);
    lmList.setCommandListener(this);

    catForm = new Form("Input the category");
    catForm.append(catTF);

    catForm.addCommand(okCommand);
    catForm.addCommand(cancelCommand);
    catForm.setCommandListener(this);

    resultForm = new Form("LandmarkStore Result");
    resultForm.addCommand(backCommand);

```



```

        resultForm.setCommandListener(this);
    }

```

The following is the code example of `LandmarkStore.AddLandmark()`.

```

public void addNewLmObj()
{
    String lmName;
    String catName;
    double latitude;
    double longitude;
    float altitude;
    float horizontalAccuracy;
    float verticalAccuracy;
    String description;

    //Landmark lm;

    try {
        lmName = lmNameTF.getString();
        catName = catNameTF.getString();

        latitude = Double.parseDouble(latitudeTF.getString());
        longitude = Double.parseDouble(longitudeTF.getString());
        altitude = Float.parseFloat(altitudeTF.getString());
        horizontalAccuracy =
Float.parseFloat(horizontalAccuracyTF.getString());
        verticalAccuracy =
Float.parseFloat(verticalAccuracyTF.getString());

        description = descriptionTF.getString();

        qc = new QualifiedCoordinates(latitude, longitude,
altitude, horizontalAccuracy, verticalAccuracy);
        lm = new Landmark(lmName, description, qc, null);
        lmStore.addLandmark(lm, catName);
        lmItems.addElement(lm);
        lmList.append(lmName + "," + longitude + "," + latitude +
", " + altitude, null);
        disp.setCurrent(lmList);
        /*
        if(getItemIndex(categoryName) < 0)
        {
            items.addElement(categoryName);
            categoryList.append(categoryName, null);
        }
        else
        {
            // add the same coordinates, nothing need to be done
        }*/
    }
    catch (Exception e) {
        resultForm.deleteAll();
        resultForm.append("" + e);
    }
}

```



```

        disp.setCurrent(resultForm);
    }
}

public void addLmToCat()
{
    int i = lmList.getSelectedIndex();
    int j = catList.getSelectedIndex();
    try{
        lmStore.addLandmark((Landmark)lmItems.elementAt(i),
(String)catItems.elementAt(j));
        disp.setCurrent(lmList);
    }catch (Exception e)
    {
        resultForm.deleteAll();
        resultForm.append("" + e);
        disp.setCurrent(resultForm);
    }
}

public AddLandmarkTest()
{
    disp = Display.getDisplay(this);

    cmdList = new List("Add Landmark Test", List.IMPLICIT);

    cmdList.append("Add Landmark Object", null);
    cmdList.append("Add Landmark to Category", null);

    cmdList.addCommand(selectCommand);
    cmdList.addCommand(exitCommand);
    cmdList.setCommandListener(this);
    //////////////////////////////////////

    newLmForm = new Form("Add Landmark Object");

    newLmForm.append(lmNameTF);
    newLmForm.append(catNameTF);
    newLmForm.append(latitudeTF);
    newLmForm.append(longitudeTF);
    newLmForm.append(altitudeTF);
    newLmForm.append(horizontalAccuracyTF);
    newLmForm.append(verticalAccuracyTF);
    newLmForm.append(descriptionTF);

    newLmForm.addCommand(addCommand);
    newLmForm.addCommand(cancelCommand);
    newLmForm.setCommandListener(this);

    //////////////////////////////////////
    lmList = new List("Landmark List", List.IMPLICIT);
    try {
        lmStore = LandmarkStore.getInstance(null);

```




```

        if(lmStore == null)
        {
            throw new Exception("Can't get landmarkStore
Instance!");
        }
        landmarks = lmStore.getLandmarks();
    }catch (Exception e)
    {
        disp.setCurrent(resultForm);
        resultForm.deleteAll();
        resultForm.append(" " + e);
    }

    if (landmarks!=null)
    {
        while (landmarks.hasMoreElements())
        {
            lm = (Landmark)landmarks.nextElement();
            qc = lm.getQualifiedCoordinates();
            lmItems.addElement(lm);
            if (qc!=null)
            {
lmList.append(lm.getName()+" "+qc.getLatitude()+" "+qc.getLongitude()
                +", "+qc.getAltitude(), null);
            }
            else
            {
                lmList.append(lm.getName(), null);
            }
        }
        lmList.addCommand(addtoCommand);
        lmList.addCommand(backCommand);
        lmList.setCommandListener(this);

        catList = new List("Category List", List.IMPLICIT);
        categories = lmStore.getCategories();
        if (categories!=null)
        {
            while (categories.hasMoreElements())
            {
                cat = (String)categories.nextElement();
                catItems.addElement(cat);
                catList.append(cat, null);
            }
        }

        catList.addCommand(addCommand);
        catList.addCommand(cancelCommand);
        catList.setCommandListener(this);
    }

```



```

        resultForm = new Form("AddLandmarkTest Result");
        resultForm.addCommand(backCommand);
        resultForm.setCommandListener(this);
    }

```

The following is the code example of `LocationProvider.getLocation`.

```

private void test()
{
    int hAccuracy;
    int vAccuracy;
    int timeout;
    String aStr = accTF.getString();
    String vStr = vaccTF.getString();
    String tStr = toTF.getString();
    String dispStr = "";
    Location loc;
    QualifiedCoordinates c;
    result.deleteAll();
    try
    {
        hAccuracy = Integer.parseInt(aStr);
        vAccuracy = Integer.parseInt(vStr);
        timeout = Integer.parseInt(tStr);
    }
    catch (NumberFormatException nfe)
    {
        result.append("Invalid input");
        disp.setCurrent(result);
        return;
    }
    try
    {
        Criteria cr = new Criteria();
        dispStr = "Criteria.setHorizontalAccuracy";
        cr.setHorizontalAccuracy(hAccuracy);
        cr.setVerticalAccuracy(vAccuracy);
        dispStr = "LocationProvider.getInstance";
        lp = LocationProvider.getInstance(cr);
        if (lp != null)
        {
            loc = lp.getLocation(timeout);
            if (loc != null)
            {
                if ( !loc.isValid() )
                {
                    dispStr += " got an invalid location instance
\n ";
                }
                dispStr += "\ntext/plain:";
                dispStr += loc.getExtraInfo("text/plain");
                c = loc.getQualifiedCoordinates();
                if (c != null)
                {

```



```

                                dispStr += "\nSpeed "+loc.getSpeed()+ " ";
                                dispStr += "\nHAcc "+
c.getHorizontalAccuracy()+ " ";
                                dispStr += "\nVAcc "+ c.getVerticalAccuracy() +
" ";
                                dispStr +=  "\nLat "+
Coordinates.convert(c.getLatitude(), Coordinates.DD_MM_SS) +" ";
                                dispStr +=  "\nLong "+
Coordinates.convert(c.getLongitude(), Coordinates.DD_MM_SS) +" ";
                                dispStr +=  "\nAlt "+ c.getAltitude() +" ";
                                dispStr += "\n we can get the qualified
coordinates!\n";
                                dispStr += "nmea:";
                                dispStr += loc.getExtraInfo("application/X-
jsr179-location-nmea");
                                dispStr += "\nlif:";
                                dispStr += loc.getExtraInfo("application/X-
jsr179-location-lif");
                                }
                                else
                                {
                                    dispStr = "getQualifiedCoordinates returns
null";
                                }
                                }
                                else
                                {
                                    dispStr = "getLocation returns null";
                                }
                                }
                                else
                                {
                                    dispStr = "LocationProvider.getInstance returns
null";
                                }
                                }
                                catch (Exception e)
                                {
                                    result.append("Exception:" + e + " at " + dispStr);
                                    disp.setCurrent(result);
                                    return;
                                }
                                result.append(dispStr);
                                disp.setCurrent(result);
                                return;
                            }
    }

```

Following is the code example of `LocationProvider.setLocationListener()`

```

public class LocListenerTest extends MIDlet implements Runnable{

    /**
     * Creates new SeaTest MIDlet.

```



```

*/
private LocationProvider lp=null;

public LocListenerTest() {}

public void startApp() {
    Thread t=new Thread(this);
    t.start();
    SThread sthread = new SThread();
    sthread.start();
}

public void run(){
    try{
        //create a Criteria for defining desired selection criteria
        Criteria cr = new Criteria();
        cr.setHorizontalAccuracy(500);

        lp = LocationProvider.getInstance(cr);
    }
    catch (Exception e ){
        System.out.println("Exception:"+e.getMessage());
    }
    LocListenerTestP2 p2 =new LocListenerTestP2();

    int interval = -1;
    int timeout = -1;
    int maxAge = -1;
    System.out.println("interval ="+interval+" timeout="+timeout+"
maxAge="+maxAge+"\n");
    lp.setLocationListener(p2, interval, timeout, maxAge);
} catch (Exception e ){
    System.out.println("exception "+e);
}

}

public void pauseApp() {}
public void commandAction(Command c, Displayable s) {}

class SThread extends Thread {
    public SThread() {}
    public void run() {
        try{
            Thread.sleep(120000);
        }
        catch (InterruptedException e) {}
        catch (IllegalMonitorStateException e) {}
        System.out.println("====SThread cancel the location
listener");
        try{
            lp.setLocationListener(null,3, 1,2);
        }
        catch ( Exception e )

```



```

        {
            System.out.println(e.getMessage());
        }
    }
}

class LocListenerTestP2 implements LocationListener{
    private static int count = 0;
    public void locationUpdated(LocationProvider provider, Location
location){
        count++;
        if ( location != null )
        {
            if ( location.isValid() )
            {
                System.out.println("getAddressInfo()="+location.getAddre
ssInfo());

                System.out.println("getCourse()="+location.getCourse());

                System.out.println("getLocationMethod()="+location.getLo
cationMethod());

                System.out.println("getTimestamp()="+location.getTimesta
mp());

                System.out.println("getSpeed()="+location.getSpeed());
                QualifiedCoordinates qc =
                location.getQualifiedCoordinates();
                if ( qc!=null)
                {

                    System.out.println("getHorinzontalAccuracy()="+qc.get
HorizontalAccuracy());

                    System.out.println("getVerticalAccuracy()="+qc.getVer
ticalAccuracy());

                    System.out.println("getLatitude()="+qc.getLatitude());
                    ;

                    System.out.println("getLongitude()="+qc.getLongitude(
));

                    System.out.println("getAltitude()="+qc.getAltitude())
                    ;
                }

                System.out.println("nmea:"+location.getExtraInfo("applic
ation/X-jsr179-location-nmea"));
            }
        }
    }
}

```



```

        System.out.println("lif:"+location.getExtraInfo("applica
        tion/X-jsr179-location-lif"));

        System.out.println("text/plain:"+location.getExtraInfo("
        text/plain"));
    }
    else
    {
        System.out.println(".....P2 location updated with INVALID
        location instance.....");

        System.out.println("text/plain:"+location.getExtraInfo("tex
        t/plain"));
    }
}
}
}
public void providerStateChanged(LocationProvider provider, int
newState){
    System.out.println("p2 providerStateChanged");
    System.out.println("new state" + newState);
}
}

```

8.6.7 Comparing with OEM AGPS API

8.6.7.1 Location Provider vs. Position Connection

The start point of getting location service is getting a `LocationProvider` instance in JSR-179 API and establishing a `PositionConnection` in OEM API respectively.

Example:

For JSR-179 API

```

try{
    //create a Criteria for defining desired criteria
    Criteria cr = new Criteria();
    cr.setHorizontalAccuracy(500);

    //get a LocationProvider instance with the defined criteria
    LocationProvider lp = LocationProvider.getInstance(cr);

    //get a location instance
    Location loc = lp.getLocation(-1);
}
catch (LocationException e){}

```



For OEM AGPS API:

```
try{
    AggregatePosition pos = null;

    PositionConnection posCon =
(PositionConnection) Connector.open("mposition:delay=low");

    pos = posCon.getPosition();
}
catch(Exception e) {}
```

8.6.7.2 Criteria vs. Quality level

For OEM AGPS API, a location request's quality is decided by the delay level and fix type.

Example of delay type: `PositionConnection sc = (PositionConnection) Connector.open(String name);` String name should be one of the following: `name = "mposition:delay=no"` `name = "mposition:delay=low"` `name = "mposition:delay=high"`.

Example of fix type: `AggregatePosition pos = posCon.getPosition(String name);` Possible delay type is `"delay=no"` or `"delay=low"` or `"delay=high"` or `"delay=low;fix=extended"` or `"delay=high;fix=extended"`. Using the `"fix=extended"` tag will return accurate velocity and heading direction.

While for JSR-179 API, a location request's quality is decided by the criteria applied upon this location provider. JSR179 API location requests implicitly apply the corresponding delay levels by the following rules:

- If a location provider is required with some horizontal or vertical accuracy, and NOT allowed to cost, its location requests will be with delay tolerant level;
- If a location provider is required with some horizontal or vertical accuracy, allowed to cost, and power consumption level is low or medium or no requirement, its location requests will be with delay tolerant level;
- If a location provider is required with some horizontal or vertical accuracy, allowed to cost, and power consumption level is high, its 1st location request will be with low delay and subsequent requests will be with delay tolerant level;
- If a location provider's horizontal and vertical accuracies are NOT required, and NOT allowed to cost, and power consumption is medium or high or no requirement, its location requests will be with delay tolerant level;
- If a location provider's horizontal and vertical accuracies are NOT required, and allowed to cost, and power consumption is medium or no requirement, its location requests will be with low delay level;
- If a location provider's horizontal and vertical accuracies are NOT required, and allowed to cost, and power consumption is high level, its 1st location request will be with low delay and subsequent requests will be with delay tolerant level;
- If a location provider's horizontal and vertical accuracies are NOT required, and allowed to cost, and power consumption is low level, its location requests will be with no delay level;



And the requests also apply the fix type by following rules:

- If a location provider is required with speed and course, its location requests will be with extended fix type;
- Or else, the requests will be with standard fix type.

8.6.7.3 Request capacity

For OEM AGPS API, only one request of `PositionConnection.getPosition()` can be made and pending at a time and this limitation works across concurrent applications.

While for JSR-179 API, only one request of `LocationProvider.getLocation()` can be made and pending at a time and this limitation works within an application. Therefore in the case of concurrent applications, totally 3 requests can be made and pending at a time.



8.7 Customer Care API

8.7.1 Overview

The Customer Care API lets J2ME™ applications access unit and user specific data. This data may be used to track and troubleshoot issues out on the field. Specifically it will provide access to such information as the unit info, system status, reset/error log, client info, my info, and Java System metrics. This feature is protected with the "System Information Access" or "Read User Data Access" functional groups of the Permission/Security Domain feature, which is described in "7.6 MIDP 2.0 Security API on page 384.

8.7.2 Class Description

The API for the Customer Care is located in package `com.mot.iden.customercare`

```
java.lang.Object
|
+ -- com.mot.iden.customercare.CustomerCare
```

8.7.3 Method Descriptions

8.7.3.1 CustomerCare Methods

8.7.3.1.1 `getUnitInfo`

Returns information about this phone such as the phone model, the version of the codeplug, the version of the CSD, the version of the software, the GPS version, and the version of the user file.

```
public static final String getUnitInfo(int fieldID)
throws IllegalArgumentException
```

`fieldID` must be one of the values in this table:

fieldID	Example of Data
DEVICE_MODEL	"i90cA"
CP_VERSION	"19.00/19.00"
CSD_VERSION	"C97.05.05"
SW_VERSION	"D76.01.09"
GPS_VERSION	"SiRF Cust SW Version Info"
USR_VERSION	"U00c.00.00"

If `fieldID` is not one of those values, this method throws an `IllegalArgumentException`.



8.7.3.1.2 `getSystemStatus`

Returns system status information such as signal quality, carrier channel, carrier color code (including extended color code for supporting devices), power cutback level and serving cell quality.

```
public static final String getSystemStatus(int fieldID)
throws IllegalArgumentException
```

`fieldID` must be one of the values in this table:

fieldID	Example of Data
SQE	"34.73"
CARRIER_CHNL	"2EB"
COLOR_CODE	"1"
PWR_CUTBACK	"00db"
SVG_CELL_QUALITY	"-66".

If `fieldID` is not one of those values, this method throws an `IllegalArgumentException`.

8.7.3.1.3 `getSystemInfo`

Returns system information such as the total Program and Data Space and the available Program and Data Space.

```
public static final int getSystemInfo(int fieldID)
throws IllegalArgumentException
```

`fieldID` must be one of the values in this table:

fieldID	Example of Data
SQE	"34.73"
CARRIER_CHNL	"2EB"
COLOR_CODE	"1"
PWR_CUTBACK	"00db"
SVG_CELL_QUALITY	"-66".

If `fieldID` is not one of those values, this method throws an `IllegalArgumentException`.



8.7.3.1.4 getMyInfo

Returns information on the various service ids for the device, such as private dispatch id (as a '*' delimited UFMI), the main phone number, the alternate line phone number for supporting devices, and the carrier IP address.

```
public static final String getMyInfo(int fieldID)
throws IllegalArgumentException
```

fieldID must be one of the values in this table:

fieldID	Example of Data
PRVT_ID	"902*43*12345"
LINE_1	"5551234567"
LINE_2	"5558901234"
CARRIER_IP	"123.45.67.89".

If fieldID is not one of those values, this method throws an IllegalArgumentException.

For all fieldIDs, a null will be returned for applications in unauthorized domains.

8.7.3.1.5 getClientInfo

Returns client information such as a unique identifier (or IMEI), the device's serial number, and the SIM identifier.

```
public static final String getClientInfo(int fieldID)
throws IllegalArgumentException
```

fieldID must be one of the values in this table:

fieldID	Example of Data
IMEI	"01010101010101"
SERIAL_NUMBER	"1234567890"
SIM_ID	"12345678901234"

For IMEI, SERIAL_NUMBER, and SIMID, the device provides hashed unique values so that applications can identify the phone with a unique identifier.



8.7.3.1.6 getErrors

Returns information on the errors that have occurred on the device.

```
public static final String getErrors(int fieldID)
```

throws `IllegalArgumentException`

The errors are kept in the reset log. This log stores only the last 24 errors.

`fieldID` must be one of the values in this table:

fieldID	Example of Data
RESET_NUMBER	"1", "2"
RESET	"R[0]:R0400Date:N/AX-&1019594CT100CFC", "R[0]:RXXXXDate:XX/XX&XXXXXXXXXXXXXXXXXX R[1]:RXXXXDate:XX/XX&XXXXXXXXXXXXXXXXXX"

If `fieldID` is not one of those values, this method throws an `IllegalArgumentException`.

8.7.4 Code Examples

```
public void test(){

    try {
        // Get unit info methods
        // 1) device model
        // 2) codeplug version
        // 3) csd version
        // 4) software version
        // 5) gps version
        // 6) usr version
        for(int x = 1; x < 7; x++){
            System.out.println("Unit Info Methods(" + x + ")
->" +
                CustomerCare.getUnitInfo(x));
            screen.append("Unit Info Methods(" + x + ") ->" +
                CustomerCare.getUnitInfo(x));
        }

        // Get system status methods
        // 7) sqe signal quality
        // 8) carrier channel
        // 9) carrier color code
        // 10) power cutback level
        // 11) serving cell quality
        for(int x = 7; x < 12; x++){
            System.out.println("System Status Methods(" + x +
") ->" +
                CustomerCare.getSystemStatus(x));
            screen.append("System Status Methods(" + x + ") -
>" +
```



```

        CustomerCare.getSystemStatus(x));
    }

    // Get my info methods
    // 12) private ID
    // 13) line 1 phone number
    // 14) line 2 phone number
    // 15) carrier IP address
    for(int x = 12; x <16; x++){
        System.out.println("My Info Methods(" + x + ") ->" +
            CustomerCare.getMyInfo(x));
        screen.append("My Info Methods(" + x + ") ->" +
            CustomerCare.getMyInfo(x));
    }

    // Get client info methods
    // 16) imei number
    // 17) serial number
    // 18) sim ID
    for(int x = 16; x <19; x++){
        System.out.println("Client Info Methods(" + x +
    ") ->" +
        CustomerCare.getClientInfo(x));
        screen.append("Client Info Methods(" + x + ") ->" +
            CustomerCare.getClientInfo(x));
    }

    // Get error/reset methods
    // 19) reset number
    // 20) reset log
    for(int x = 19; x <21; x++){
        System.out.println("Reset & Error Methods(" + x +
    ") ->" +
        CustomerCare.getErrors(x));
        screen.append("Reset & Error Methods(" + x + ") -
    >" +
        CustomerCare.getErrors(x));
    }

    // Get java system methods
    // 21) free data space
    // 22) total data space
    // 23) free program space
    // 24) total data space
    for(int x = 21; x <25; x++){
        System.out.println("Java System Methods(" + x +
    ") ->" +
        CustomerCare.getSystemInfo(x));
        screen.append("Java System Methods(" + x + ") ->" +
            CustomerCare.getSystemInfo(x));
    }
}

```



```
catch(Exception e){  
    System.out.println("Something went wrong! "+  
        e.toString());  
}  
}
```

8.7.5 Compiling & Testing Customer Care MIDlets

In the stub classes, the methods return either 0 or null since there is no native support for them.



Appendix A:

Specification Sheets

iDEN Multi-Communication Device Specifications

	Hardware	i325	i285	i730 / i710	i830 / i830e	i860	i265/ i275/i355	i605
GRIP	Form Factor	Monolith	Monolith	Clam	Clam	Clam	Monolith	Monolith
PERFORMANCE	Platform	CLDC 1.0 MIDP 2.0	CLDC 1.1 MIDP 2.0	CLDC 1.0 MIDP 2.0	CLDC 1.1 MIDP 2.0	CLDC 1.1 MIDP 2.0	CLDC 1.1 MIDP 2.0	CLDC 1.1 MIDP 2.0
	Graphics Accelerator	No	No	No	No	ATI W2250	No	ATI W2262
USER INTERFACE	External Display	N/A	N/A	96x32 Mono*	96x32 Mono	96x65 CSTN 12-bit	N/A	N/A
	Internal Display	96x65 CSTN 12-bit	96x65 CSTN 12-bit	130x130 TFT 16-bit	130x130 TFT 16-bit	176X220 TFT 16-bit Transmissive (176X206 Available to MIDlets)	130x130 TFT 16-bit	176X220 TFT 16-bit Transmissive (176X206 Available to MIDlets)
	Navigation/Joystick	8-way with Multikey Press	8-way with Multikey Press	8-way + Center Select with Multikey Press	8-way + Center Select with Multikey Press	8-way + Center Select with Multikey Press	8-way + Center Select with Multikey Press	8-way + Center Select with Multikey Press

* External Display is not available in the i710.



Continued:

		i325	i285	i730 / i710	i830 / i830e / i265 / i275 / i355	i860	i605
	Hardware						
MEDIA	Image File Formats	PNG w/ Transparency and Alpha, JPEG	PNG w/ Transparency and Alpha, JPEG	WBMP, PNG w/ Transparency and Alpha, JPEG	WBMP, PNG w/ Transparency and Alpha, JPEG	WBMP, PNG w/ Transparency and Alpha, JPEG	WBMP, PNG w/ Transparency and Alpha, JPEG
	Video Playback Formats	N/A	N/A	N/A	N/A	Motion JPEG, MP4, H.263	Motion JPEG, MP4, H.263
	Audio File Formats	MIDI, WAV, AU, VSELP	MIDI, WAV, AU, VSELP	MIDI, WAV, AU, VSELP	MIDI, WAV, AU, VSELP, AMBE	MP3, MIDI, WAV, AU, VSELP, AMBE, AMR	MP3, MIDI, WAV, AU, VSELP, AMBE, AMR
	Onboard Camera	No	No	No	i275 Only.	Yes	No
	Camera Capture Formats	N/A	N/A	Still Image: JPEG	Still Image: JPEG	Still Image: JPEG Video Capture: H.263	N/A
	3D Rendering	No	No	Micro3D	Micro3D	Micro3D, JSR-184	Micro3D, JSR-184
NETWORKING & DATA	AGPS	Yes	Yes	Yes	Yes	Yes	Yes
	UDP	Maximum of 21 sockets	Maximum of 21 sockets	Maximum of 24 sockets	Maximum of 20 sockets	Maximum of 24 sockets	Maximum of 24 sockets
	TCP	Maximum of 14 sockets	Maximum of 14 sockets	Maximum of 16 sockets	Maximum of 12 sockets	Maximum of 16 sockets	Maximum of 16 sockets
	HTTP/HTTPS	Maximum of 4 sockets	Maximum of 4 sockets	Maximum of 8 sockets	Maximum of 4 sockets	Maximum of 8 sockets	Maximum of 8 sockets
	Serial	Yes	Yes	Yes	Yes	Yes	Yes
	Bluetooth	No	No	No	No	No	Yes
STORAGE & MEMORY	Java Heap	1.1 MB	1.1 MB	1.1 MB	1.1 MB	2 MB	4 MB
	Data Space	1 MB	1 MB	2 MB	3 MB	25 MB	11 MB
	Program Space	1 MB	1 MB	2 MB	4 MB	4 MB	4 MB



Appendix B:

Java APIs

Feature Matrix for iDEN Multi-Communication Devices

Features	i285	i325	i730/ i710	i830/i830e/ i265/i275/ i355	i860	i605
Supported JSRs						
CLDC 1.0 (JSR-30)	✓	✓	✓	✓	✓	✓
CLDC 1.1 (JSR-139)	✓		✓ ¹	✓	✓	✓
MIDP 1.0 (JSR-37)	✓	✓	✓	✓	✓	✓
MIDP 2.0 (JSR-118)	✓	✓	✓	✓	✓	✓
PNG Transparency for MIDP (JSR-37 and JSR-118)	✓	✓	✓	✓	✓	✓
Alpha Blending for MIDP (JSR-37 and JSR-118)	✓	✓	✓	✓	✓	✓
JPEG for MIDP (JSR-37 and JSR-118)	✓	✓	✓	✓	✓	✓
Bold/Underline Fonts for MIDP (JSR-37 and JSR-118)	✓	✓	✓	✓	✓	✓
Key Repeat for MIDP (JSR-37 and JSR-118)	✓	✓	✓	✓	✓	✓
Datagram Connection for MIDP (JSR-37 and JSR-118)	✓	✓	✓	✓	✓	✓
Socket Connection for MIDP (JSR-37 and JSR-118)	✓	✓	✓	✓	✓	✓
Serial Port Interface for MIDP (JSR-37 and JSR-118)	✓	✓	✓	✓	✓	✓
HTTP 1.1 Persistency for MIDP (JSR-37 and JSR-118)	✓	✓	✓	✓	✓	✓
File Connection for MIDP (JSR-37 and JSR-118)	✓	✓	✓	✓	✓	✓
Secure File Connection for MIDP (JSR-37 and JSR-118)		✓	✓	✓	✓	✓
HTTPS/SSL 3.0 for MIDP (JSR-37 and JSR-118)	✓	✓	✓	✓	✓	✓

1: Later releases of the i730 include this feature. Check Java System for CLDC 1.1 support.



Features	i285	i325	i730/ i710	i830/i830e /i265/i275/ i355	i860	i605
Supported JSRs (continued)						
Location API for J2ME™ (JSR-179)					✓	✓
PDA Optional Packages for J2ME™ (JSR-75)					✓	✓
Mobile 3D Graphics API for J2ME™(JSR-184)					✓	✓
J2ME Web Services Specification (JSR-172)					✓	✓
XML Parsing with SAX for WSS (JSR-172)					✓	✓
Remote Procedure Calls (RPC) for WSS (JSR-172)						✓
Mobile Media API 1.1 (JSR-135)	✓	✓	✓	✓	✓	✓
Audio for MMAPI 1.1 (JSR-135)	✓	✓	✓	✓	✓	✓
Digital Camera Support for MMAPI 1.1 (JSR-135)			✓	✓	✓	✓
Image Utility Library for MMAPI 1.1 (JSR-135)					✓	✓
Video Capture/Playback Support for MMAPI 1.1 (JSR-135)					✓	✓ ³
Wireless Messaging API 1.1 (JSR-120)			✓	✓	✓	✓
Wireless Messaging API 2.0 MMS Extensions (JSR-205)					✓	✓
Java APIs for Bluetooth (JSR-82)						✓
iDEN Defined (OEM) APIs						
Call Initiation	✓	✓	✓	✓	✓	✓
Call Receive			✓	✓	✓	✓
Crypto		✓	✓	✓	✓	✓
Customer Care	✓	✓	✓	✓	✓	✓
Datebook	✓	✓	✓	✓	✓	✓
Distributed Speech Recognition (DSR)						✓
External Display			✓ ²	✓ ²	✓	✓
License						✓
Lighting	✓	✓	✓	✓	✓	✓
Lightweight Windowing Toolkit (LWT)			✓	✓	✓	✓
Location	✓	✓	✓	✓	✓	✓
Look and Feel (LnF)	✓	✓	✓	✓	✓	✓

2: External Display API is only available on the i730, i830, and i830e handsets.

3: Video playback support only.



Features	i285	i325	i730/ i710	i830/i830e i265/i275/ i355	i860	i605
iDEN Defined (OEM) APIs (continued)						
Math (With Floating Point Support)	✓	✓	✓	✓	✓	✓
Micro3D			✓	✓	✓	✓
MIDI	✓	✓	✓	✓	✓	✓
Object Push Protocol (OPP)						✓
PhoneBook	✓	✓	✓	✓	✓	✓
Private Call Receive				✓ ⁴	✓ ⁴	✓ ⁴
Realtime Protocol						✓
Recent Call		✓	✓	✓	✓	✓
Resource Bundles					✓	✓
SDG Call				✓ ^{4 5}	✓ ⁴	✓ ⁴
Status Manager			✓	✓	✓	✓
Vibrator	✓	✓	✓	✓	✓	✓
VoiceNotes	✓	✓	✓	✓	✓	✓
Zip	✓	✓	✓	✓	✓	✓
Java Related Platform Features						
Auto Install						✓
Customizable Keymaps					✓	✓
Internationalization Support for Application Management	✓	✓	✓	✓	✓	✓
JAID Install All In Jar	✓	✓	✓	✓	✓	✓
MIDlet Concurrency	✓	✓	✓	✓	✓	✓
MIDlet-Icon Support			✓	✓	✓	✓
Personalized Java (Main Menu Java Support)	✓	✓	✓	✓	✓	✓
Powerup App	✓	✓	✓	✓	✓	✓
Smart Text Entry (T9)	✓	✓	✓	✓	✓	✓
Time Zone Support	✓	✓	✓ ¹	✓	✓	✓

1: Later releases of the i730 include this feature. Check Java System for CLDC 1.1 support.

4: Private and SDG Call Receive functionality is limited by some carriers.

5: SDG is not available on the i830 handset.



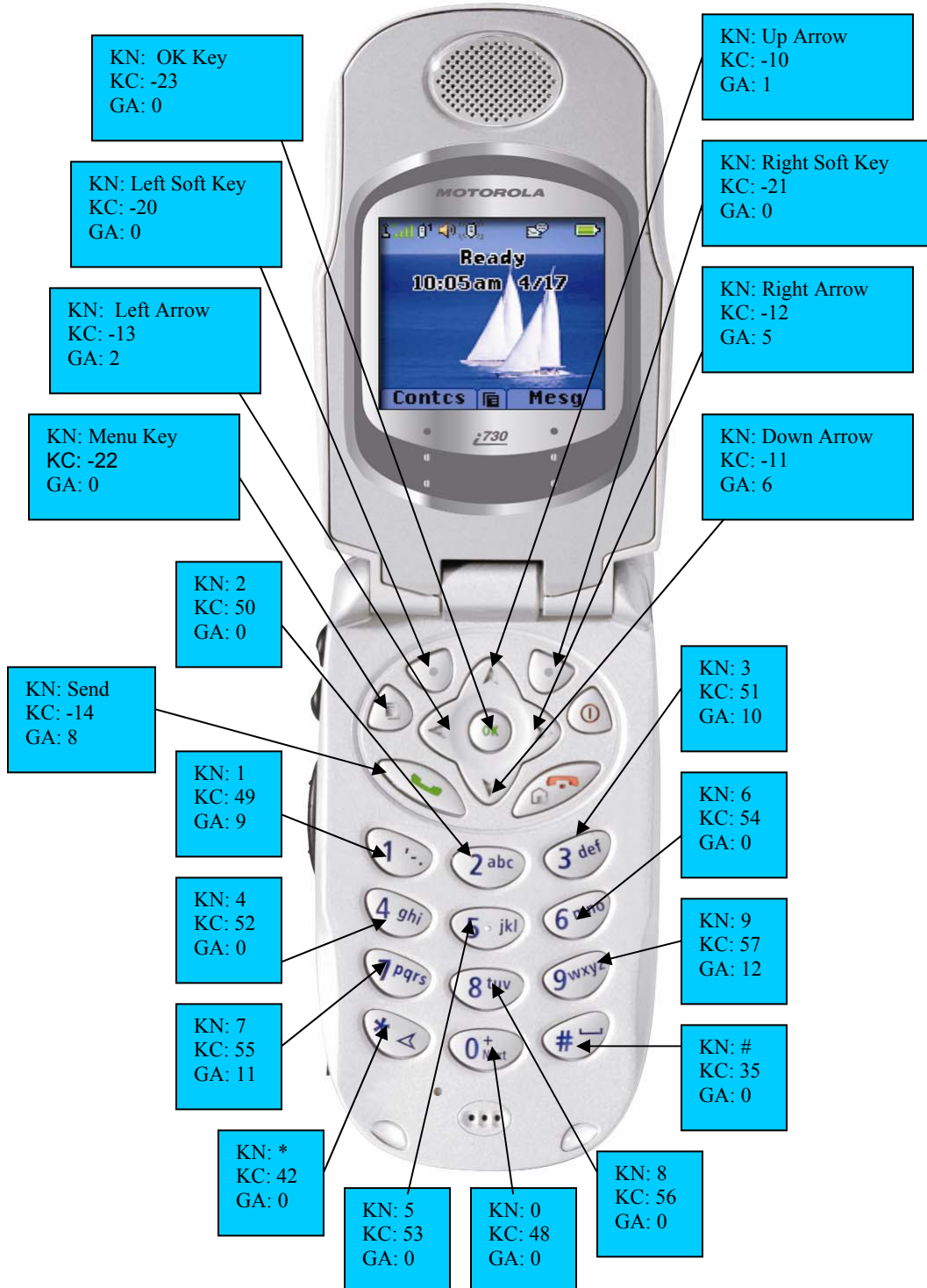
Appendix C: Key Maps for the iDEN Multi-Communication Devices

Overview

This section supplies specific key maps for the iDEN Multi-Communication devices. Note that the power and “end” keys are never sent to MIDlets. Not all handsets feature “OK” or “Smart” keys.



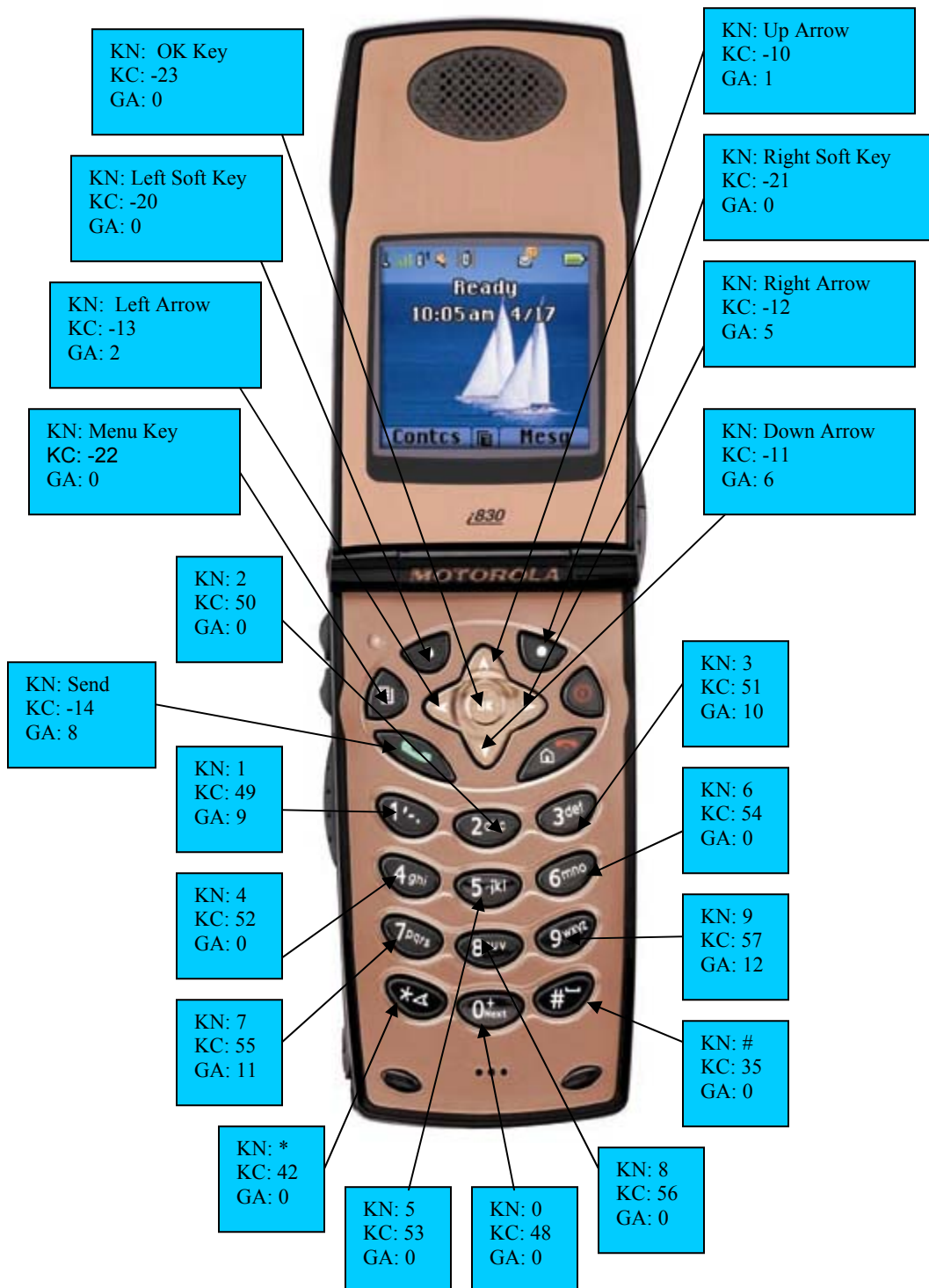
i730 / i710 Multi-Communication Device







i830 / i830e Multi-Communication Device







i285 Multi-Communication Device





i325 Multi-Communication Device



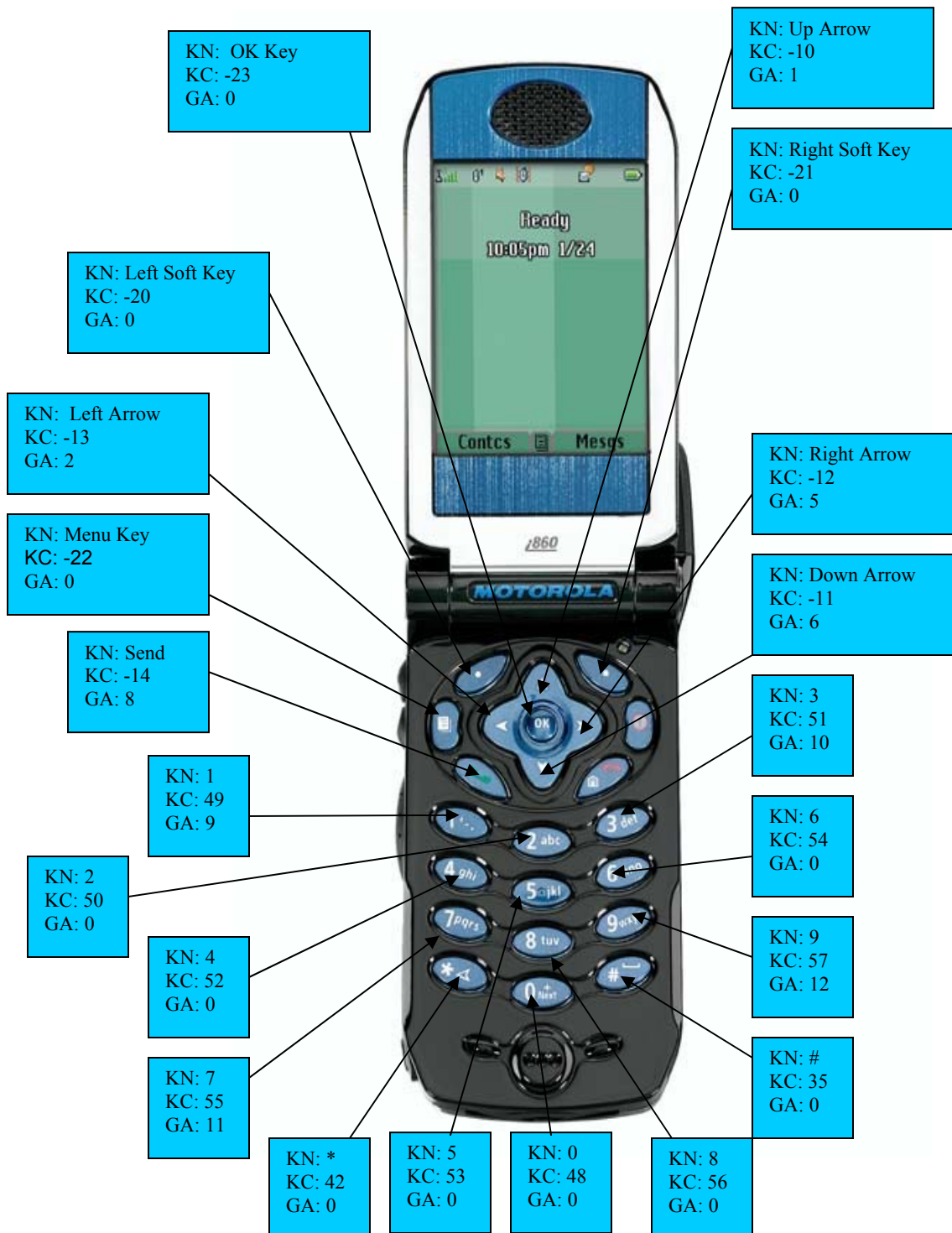


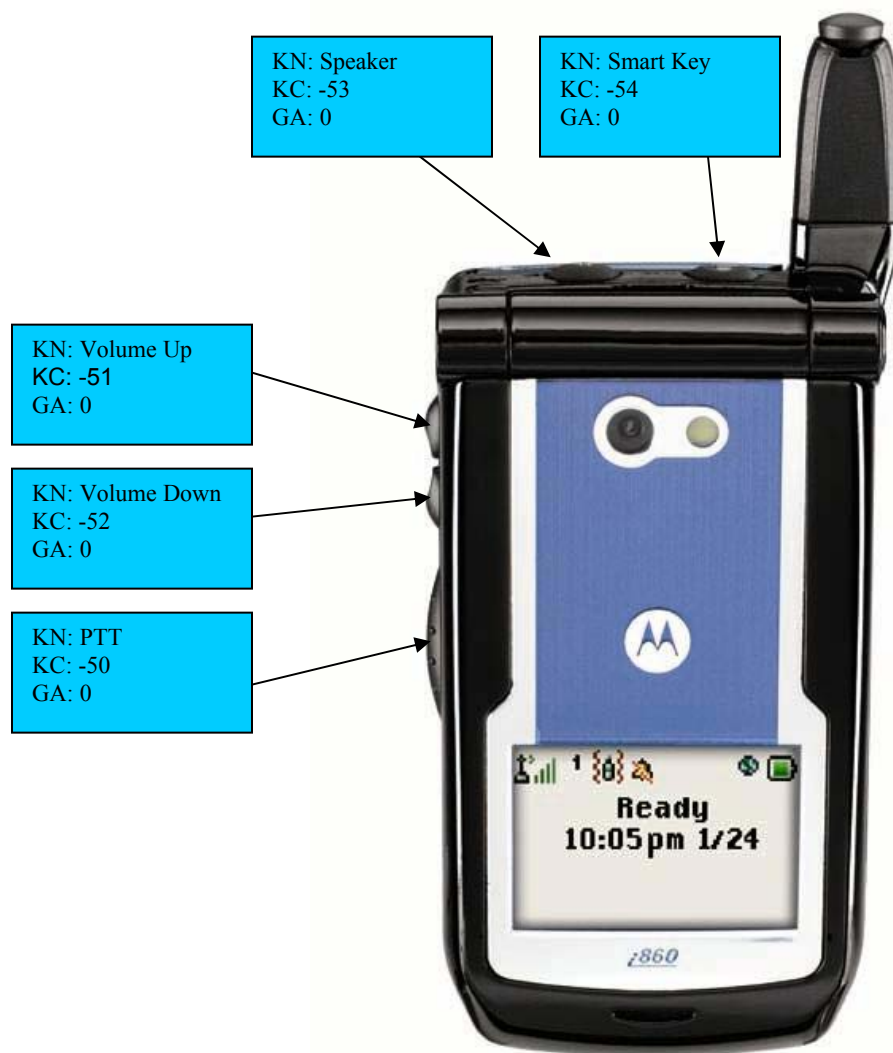
i355 Multi-Communication Device





i860 Multi-Communication Device







i265/i275 Multi-Communication Device





i605 Multi-Communication Device





MOTOROLA and the Stylized M Logo are registered in the U.S. Patent & Trademark Office. All other product or service names are the property of their respective owners. Java and all other Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

© Motorola, Inc. 2005.